Signing and security of Hue software

Technische Universiteit
Eindhoven
University of Technology

/ Department of
Mathematics and
Computer Science
/ PDEng Software
Technology

Signing and security of Hue software

Igor Anastasov

Signing and security of Hue software

Eindhoven University of Technology
Stan Ackermans Institute / Software Technology

Partners

Philips Lighting

Eindhoven University of Technology

Steering Group Luud Woltjer (Philips Lighting)
Leon Bouwmeester (Philips Lighting)
Walter Slegers (Philips Lighting)
Tanir Ozcelebi (TU/e)
Ad Aerts (TU/e)
Yanja Dajsuren (TU/e)

Date September 2017

Document Status Public

SAI report no. **2017/036**

The design described in this report has been carried out in accordance with the TU/e Code of Scientific Conduct.

Contact Address
Eindhoven University of Technology
Department of Mathematics and Computer Science
MF 5.080A, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
+31402474334

Published by
Eindhoven University of Technology
Stan Ackermans Institute

Printed by
Eindhoven University of Technology
UniversiteitsDrukkerij

SAI report no.
2017/036

Abstract

Developing software for the Hue devices poses plenty of challenges among the engineers at Philips Lighting. These challenges arise at each stage of the Software Development Life-Cycle (SDLC). Improvement of it is of immense importance to the Philips Lighting. This report describes a project which focus was to automate the SDLC, as well as to improve the security in it. The end result solves many challenges. It delivers a complete release management tool dedicated to the engineers at the Home Systems department. First, it visualizes release workflows in a simple user interface. Second, the core activities of the SDLC, such as the software signing, are fully automated. What is more important is that the signing is executed in a highly secure environment. This is very important for Philips Lighting not only because this automation saves a lot of time, but also because it reduces the risk of a human error. The same benefits are gained through an automation of other activities, such as approvals, distribution of the software to the factories, and deploying the software to the device cloud. Third, the system provides a traceability about each step executed in the process. Finally, the system is highly configurable, which makes it easy to be extended and adjusted to support different device types with different release workflows.

Keywords
Software signing, security, SDLC, release management workflow, software deployment, Philips Hue, Philips Lighting, TUE, Software technology, PDEng

Preferred reference
Signing and security of Hue software., SAI Technical Report, September 2017. (2017/036)

Partnership
This project was supported by Eindhoven University of Technology and Philips Lighting.

Foreword

Security is becoming more and more important in the IoT space. One of the most recent examples is the hack of a casino through an internet-connected fish tank [1] that clearly shows today's creativity of hackers. Like quality, security is not a department, but an attitude that should be embraced by the entire organization supported by integral processes to always keep further improving.

This report describes the results of one such important improvement: it provides a link between the software development projects on one hand versus the digital operation to deploy the software artefacts towards end-users on the other hand. It includes a fully automated release process in a highly secure environment that not only saves a lot of time, but also eliminates the possibility of human error to a large extend. Given the diversity and scale of products in combination with necessary speed of innovation, the resulting tool is an absolute must to keep providing secure solutions. The first demonstrations clearly showed the value of this tool.

Igor played a pivoting role in this project: he identified the (key) stakeholders in this project and what their (non) functional requirements were. He also setup the initial architecture and design which determined the foundation for the resulting implementation. During the entire development process Igor in particularly paid attention to the integral security aspects of his solution by working with the various security engineers. With this report, the project hasn't stopped: Igor continues to work on this, but now as an employee of Philips Lighting.

Ir. LHA Bouwmeester, Project Manager Hue system platform.
September 2017

Preface

This report summarizes the “Signing and security of Hue software” project carried out by Igor Anastasov as the final part of the Professional Doctorate in Engineering (PDEng) program in Software Technology, provided by the Eindhoven University of Technology, Stan Ackermans Institute. The project lasted for nine months and was conducted at Philips Lighting, Eindhoven.

This document describes the successful realization of the project and elaborates the software development and project management processes. Audience of this report can be both technical as well as non-technical readers. However, it is primarily intended for the engineers at Home Systems Department in Philips Lighting who have the greatest interest of this project. Readers who are interested in the existing software release management processes and challenges within the Home Systems Department, can read Chapters 1, 2 and 3. Readers who are willing to learn more details about the software development life-cycle and core challenges in Internet of Things environment in general are invited to read Chapter 4. Readers who are interested in the technical details, such as system requirements, system architecture, system design, and system implementation will be interested in Chapters 6-11. Readers mainly interested in the results or future continuation of the project can read Chapter 12. Project managers are referred to Chapter 2, 5, 13, and 14. Readers who are interested in the entire project, are welcome to read the entire report.

Igor Anastasov
September 2017

Acknowledgements

Firstly, I would like to express my sincere gratitude to my supervisors from Philips Lighting, Leon Bouwmeester, Luud Woltjer and Walter Slegers for providing me the opportunity to do my project in Philips Lighting. I would like to thank all of you for the continuous support, motivation and feedback. Your valuable guidance as well as the knowledge and necessary information you provided, helped me to reach the success of this project. I enjoyed working with you these nine months and thank you for the vast experience I have been able to have with you. Furthermore, I am excited and looking forward to continuing cooperating with you in future as colleague in Philips Lighting.

This project would not have been a success without the support from my university supervisor Dr. Tanir Ozcelebi. Thank you for guiding me throughout the project. Your critical thinking, ideas, feedback and necessary information provided encouraged me to explore different angles and views on the project. Thank you for the continuous support.

My gratitude goes to everybody involved in the PDEng program, especially Ad Aerts, Yanja Dajsuren, and Desiree van Oorschot for giving me the opportunity to be part of the PDEng program and their support and guidance during these past two years.

I would also like to thank my colleagues from the PDEng program for the great moments and experiences we shared together.

And last but not least, I would like to thank my family for their continuous support. A special thank you goes to my girlfriend, Maja, for her support and patience. A special thank you to my parents, Zoran and Keti, for everything they have done for me, and to my sister, Natasha, for her support.

Igor Anastasov,
September 2017

Executive Summary

The Internet of Things (IoT) is an inter-networking of physical devices embedded with electronics, software, sensors, actuators, and network connectivity that enable these devices to collect and exchange data. Developing software applications for the IoT devices poses plenty of challenges that arise at each stage of the Software Development Life-Cycle (SDLC). Enterprise customers expect their IoT systems to perform for many years, during which they will require regular attention and frequent upgrading to take advantage of advances in technology. This puts pressure on engineers to deliver applications quickly, but without compromising security and performance. That means that software engineers must anticipate what systems will require in the future and plan for the whole life cycle of devices and applications at the design stage, from development, configuration, and deployment through management, monitoring, and, ultimately, decommissioning.

Improvement of the complete SDLC is of immense importance to Home Systems department at Philips Lighting too. Engineers there aim for continuous integration, deployment, and delivery processes in each aspect of the SDLC. They tend to make the SDLC as automated as possible, without compromising security aspects of the systems. This project is part of this global movement and its successful delivery makes a step further in reaching these high-level company objectives. The project has two main goals. First goal is to optimize and automate the software release management workflow, with main emphasis on the code signing process. Second goal is to improve the security in these processes, such as storing and operating the highly sensitive software signing keys.

The end result of this project solves a lot of challenges in the SDLC. It is a release management tool dedicated for the engineers at the Home department at Philips Lighting. First, it visualizes ongoing software release workflows in a simple and easy to use web-based user interface. With a lot of searching and sorting possibilities, users are able to quickly find out the specific details about a particular software release process. Second, the core activities of the SDLC process are fully automated. The testing architects are now able to digitally sign software with just a few clicks on the web-based user interface. What is more important is that this signing is executed in a highly secure and protected environment. This is of huge importance for Philips Lighting not only because this automation saves a lot of time, but also because it reduces the risk of human error that was present before. The same benefits are gained through an automation of other activities in the SDLC, such as approving the steps, distributing the signed software to the manufacturing factories or deploying the software on the device in the field. Third, the system provides a traceability of each step executed in the process, for instance, who approved the software for signing or who uploaded the signed files to the Hue device portal. Finally, the system is highly configurable, which makes it easy to be extended and adjusted to support different device types with different release workflows.

From the software architecture point of view, the designed system exhibits properties, such as modularity and maintainability, which makes it easily extendable. One of the core non-functional requirements of this project was security. This was achieved by applying advanced security mechanisms in every aspect of the system.

Table of Contents

List of Figures

List of Tables

1.Introduction

This chapter introduces the project and its context. The scope of the project and its goals are briefly mentioned. The outline section of this chapter gives a brief overview of what is discussed in the following chapters.

1.1 Context

The "Signing and security of Hue software" project was conducted by Igor Anastasov, as part of his Professional Doctorate in Engineering (PDEng) program. The PDEng degree program in Software Technology is provided by the Department of Mathematics and Computer Science of Eindhoven University of Technology in the context of the 4TU.School for Technological Design, Stan Ackermans Institute. [2]

This Professional Doctorate in Engineering program (PDEng) is an accredited and challenging two-year, third-cycle (doctorate-level) engineering degree program during which its trainees focus on strengthening their technical and non-technical competencies related to the effective and efficient design and development of software for resource-constrained software-intensive systems, such as real-time embedded or distributed systems, in an industrial setting. PDEng focuses on large-scale project-based design and development of this kind of software. [3]

The various parts of the PDEng degree program aid in developing the capability of individuals to work within a professional context. It advocates a scientific research-based approach to solving problems, a systematic way of collecting evidence and a critical, reflective, and independent mind for the analysis and interpretation of evidence. The first fifteen months of the program consist of advanced training and education, including four small, industry driven training projects. During the last nine months, a major design project in a company takes place.

The current project was initiated by Philips Lighting, an organization that develops connected lighting systems and services. By leveraging the Internet of Things, they are transforming buildings, urban places and homes to increase energy efficiency and manage working environments in a more environmentally friendly way. The next section, gives a brief introduction to the Internet of Things paradigm.

1.2 Internet of Things

From the first emails to Web 2.0 to the cloud and mobile access, the Internet has made spectacular progress over the past decades. One of the things that the future has in store is an "Internet of things": connected appliances and machines that dynamically and intelligently adapt to users' needs and preferences. In these developments, light may play a crucial part. [4]

The Internet of Things is the inter-networking of physical devices, vehicles, buildings, and other items embedded with electronics, software, sensors, actuators, and network connectivity that enable these objects to collect and exchange data. These devices are also referred to as "connected devices" and "smart devices". In 2013 the Global Standards Initiative on Internet of Things (IoT-GSI) defined the IoT as "the infrastructure of the information society." The IoT allows objects to be sensed and / or controlled remotely across existing network infrastructure, creating opportunities for more direct integration of the physical world into computer-based systems and resulting in improved efficiency, accuracy and economic benefit in addition to reduced human intervention. When IoT is augmented with sensors and actuators, the technology becomes an instance of the more general class of cyber-physical systems, which also encompasses technologies such as smart grids, smart homes, intelligent transportation and

smart cities. Each thing is uniquely identifiable through its embedded computing system but is able to interoperate within the existing Internet infrastructure. Experts estimate that the IoT will consist of almost 50 billion objects by 2020. [5]

As the possibilities for digital, connected lighting develop and expand, Philips Lightings' smart lighting infrastructure can become the glue that connects the physical world to the digital realm, creating a true "Internet of Lights". In this brave new world of connected intelligence, lighting can become an integral and responsive part of our everyday environments. [4]

1.3 Philips Lighting

"Philips Lighting is the leading provider of lighting solutions and applications for both professional and consumer markets, pioneering in how lighting is used to enhance the human experience in the places where people live and work. Whether being at home, on the road, in the city, shopping, at work or at school, Philips Lighting is creating lighting solutions that transform environments, create experiences, and help shape identities. Philips Lighting serves its customers through a market segment approach, which encompasses Homes, Office and Outdoor, Industry, Retail, Hospitality, Entertainment, Healthcare and Automotive. The company employed approximately 33,600 people worldwide with sales of EUR 7.4 billion in 2015. In 2015, Philips Lighting spanned a full-service lighting value chain-from lamps, luminaires, electronics, and controls to connected and application-specific systems and services." [6]

1.4 Philips Hue System

In 2012, Philips Lighting launched the Philips Hue System. The Hue is a completely connected home lighting system of linked bulbs that can be controlled by smartphone or tablet via a ZigBee[1] bridge that connects to the home Wi-Fi network.

Philips Hue is an internet connected (wireless) lighting system designed to transform how users experience light inside their homes. It is one of the leading and most installed smart home / Internet of Things products in the world. Philips Hue transforms how users can experience light by enabling color tunable lights to be controlled from smartphones, web services or other control logic and devices running in the system. Furthermore, it is an open system, which means that other suppliers can add third-party components via standardized or published interfaces. [7]

The Hue system consists of various products: the Hue portal[2], the mobile applications, the bridges, various lamps and luminaires, sensors, and switches. Figure 1.1 shows a high-level overview of the Hue system and its main components. Several of these devices, such as lamps, luminaires, bridges, sensors, and switches contain software. It is essential that the respective software is securely signed before it is deployed to the production factories and installed on the device. Moreover, when there is an updated version of the software available, all consumer devices in the field need to be updated with the latest version of the software. This also requires the software to be securely signed. Additionally, the distribution channels through which the software is being distributed to the factories and to the consumer devices in the field, need to be highly secured and to conform to the highest security standards.

Within the Philips Lighting projects software engineers are working towards continuous integration and deployment. Once the Integration & Verification (I&V) team has given their approval on release of specific software, the software running on, for example, the bridge must be signed and uploaded to the Hue cloud for further distribution towards all consumer devices in the field. The bridge software must also be signed and securely sent to the factory as a running change on all devices that are being produced.

[1] http://www.zigbee.org/

[2] For the remainder of this report, the terms "Hue portal," "Hue device cloud," and "Hue cloud" are used interchangeably and refer to the same system

Currently, all these steps beginning from the 'approval' from the I&V are manual steps which are error prone. The aim of this project is to automate and optimize this process.

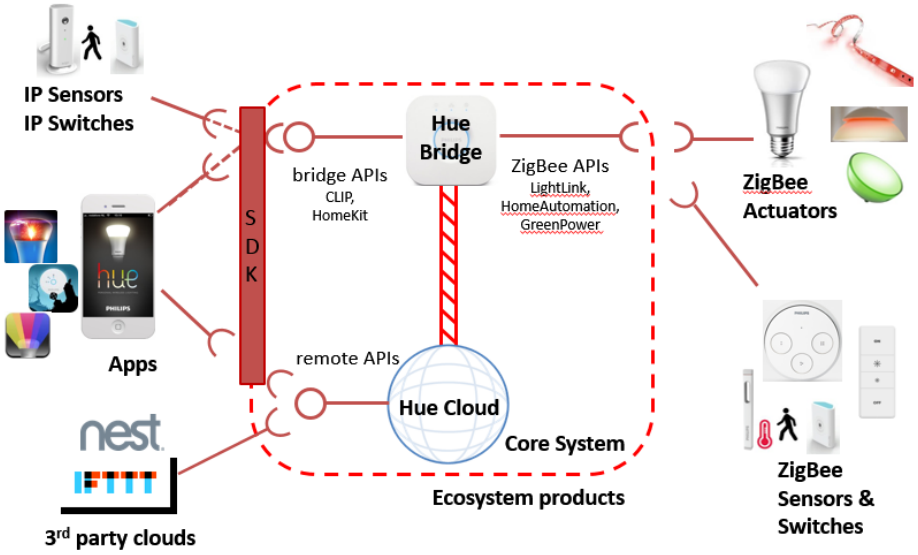

Figure 1.1 Hue system overview

1.5 Outline

The next chapter, Chapter 2, introduces the stakeholders of this project together with their interests and goals. A problem analysis is described in Chapter 3 that elaborates the problem that this project is focusing on and its current state in the ecosystem. Chapter 4 talks about the domain of the problem and gives directions towards the problem's solution. The issues and challenges encountered during the lifetime of the project as well as the risks that might arise are elaborated in Chapter 5.

The following chapters focus on the design and implementation phases of the project. Chapter 6 lists the user and functional requirements derived from the domain and problem analysis as well as from the many discussions we had with the stakeholders. The requirements together with the domain and problem analysis give an input for the system architecture that is elaborated in Chapter 7. After modeling the architecture, the design of the system is created. The system design is presented in Chapter 8. Chapter 9 describes the implementation of the system. The process of validation and verification of the system is discussed in Chapter 10. Explanation of the deployment of the solution is described in the Chapter 11. Finally, the conclusions and future work are addressed in Chapter 12.

The project management process during the lifetime of the project is described in Chapter 13. Finally, Chapter 14 gives the retrospective and reflection on the project from the author's perspective. ■

2.Stakeholder Analysis

This chapter gives an overview of the stakeholders involved in the project. The main concerned parties are Philips Lighting (supervisors and the Department of Home Systems) and the Eindhoven University of Technology (supervisor and PDEng trainee). For each concerned party, the representative stakeholders are listed together with their role, responsibilities, concerns, acceptance criteria, and involvement.

2.1 List of stakeholders

The following stakeholders were identified:

- Project Manager
- Product Owner
- Test Architect in the Integration & Validation team
- Security Officer in the Integration & Validation team
- Product Owner of the Hue bridge platform
- The Bridge Application team
- The Bridge Platform team
- The ZigBee Platform team
- The ZigBee Products team
- The Mobile Applications team
- The Device cloud (HSDP) team
- The Digital Operations department
- Quality Assurance Department
- The Integration & Validation team in China
- Factory employee – responsible for receiving and deploying the updated bridge firmware
- Eindhoven University of Technology (TU/e)
- PDEng trainee

2.2 Stakeholders Analysis

The following table gives an overview of each stakeholder's role, concerns, acceptance criteria, and their involvement during the project.

2.2.1. Project Manager and Project Owner

- ROLE
  - ✓ The project owner is the person who defines the goals of the project, funds it and looks for the business value
  - ✓ The project manager is the person who organizes the project and has to make sure that the project goals (defined by the owner) are met one time and within budget
- REPRESENTATIVE
  - ✓ Project owner: Luud Woltjer
  - ✓ Project manager: Leon Bouwmeester
- RESPONSIBILITIES
  - ✓ Monitor, evaluate, assess, and provide regular feedback on the project progress and deliverables
  - ✓ Provide relevant domain knowledge, references, and contacts
  - ✓ Provide relevant information regarding the needs and requirements of the project
  - ✓ Evaluate whether the solution meets the requirements
  - ✓ Review the final project report
- ACCEPTANCE CRITERIA
  - ✓ Timely report of the project deliverables

- CONCERNS
  - ✓ Project is delivered according to the defined timeline
  - ✓ System is developed by using approved licenses. For instance, MIT, BSD are preferred over GPL.
- INVOLVEMENT
  - ✓ During the entire project by continuous communication via daily meetings on ad–hoc basis, regular weekly progress update meetings, and regular monthly project steering group meetings.

2.2.2. Product Owner of the Hue platform

- ROLE
  - ✓ Software architect responsible for and owning the Hue platform.
- REPRESENTATIVE
  - ✓ Walter Slegers
- RESPONSIBILITIES
  - ✓ Monitor, evaluate, assess and provide regular feedback on the project progress and deliverables
  - ✓ Provide relevant domain knowledge, references, and contacts
  - ✓ Provide relevant information regarding the needs and requirements of the project
  - ✓ Evaluate whether the solution meets the requirements
  - ✓ Review the final project report
- ACCEPTANCE CRITERIA
  - ✓ Timely report of the project deliverables
  - ✓ Solution meets functional and non-functional requirements
- CONCERNS
  - ✓ Deploying the software to the wrong luminaires/lamps/bridges (a wrong bridge image deployed seemed to have happened in the early days of Hue)
  - ✓ Being able to stop a software deployment (has been necessary once because the software contained a problem discovered after deployment started)
  - ✓ Accidentally deploying to the wrong region / world or to the customers instead of alpha/beta groups.
  - ✓ Being able to see the deployment status (For example. version X detected on y% of connected bridges)
- INVOLVEMENT
  - ✓ During the entire project by continuous communication via daily meetings on ad–hoc basis, regular weekly progress update meetings, and regular monthly project steering group meetings.

2.2.3. Test Architect in the Integration & Validation team

- ROLE
  - ✓ Testing & Validation of the software
- REPRESENTATIVE
  - ✓ Luud Woltjer
- RESPONSIBILITIES
  - ✓ Provide relevant domain knowledge
  - ✓ Provide relevant information regarding the needs and requirements of the project
  - ✓ Evaluate whether the solution meets the requirements
- ACCEPTANCE CRITERIA
  - ✓ Solution meets functional and non–functional requirements
- CONCERNS
  - ✓ Giving approval on a specific release is automated and is a simple process
  - ✓ It is visible and clear which version of the release is under validation

- ✓ There is a clear and simple communication channel with the respective development teams
- INVOLVEMENT
  - ✓ During the entire project by continuous communication via daily meetings on ad–hoc basis, regular weekly progress update meetings

2.2.4. Security Officer in the Integration & Validation team

- ROLE
  - ✓ Digitally signing the production firmware version
  - ✓ Upload signed firmware to the device cloud
  - ✓ Approve deployment of the firmware on specific devices and/or factory
- REPRESENTATIVE
  - ✓ Luud Woltjer
- RESPONSIBILITIES
  - ✓ Provide relevant domain knowledge
  - ✓ Provide relevant information regarding the needs and requirements of the project
  - ✓ Evaluate whether the solution meets the requirements
- ACCEPTANCE CRITERIA
  - ✓ Solution meets functional and non–functional requirements
- CONCERNS
  - ✓ Signing certificates for production firmware are securely stored under highest possible security measures
  - ✓ Software signing is an automated, simple and straightforward process
  - ✓ Uploading the firmware to the device cloud is automated and is a simple process. It is visible which version is deployed on the test and production device cloud.
  - ✓ Approving deployment of the firmware on specific devices and /or factory is an automated and simple process
  - ✓ There is an archive of all production builds, i.e. it is clear and visible which version is released
- INVOLVEMENT
  - ✓ During the entire project by continuous communication via daily meetings on ad–hoc basis, regular weekly progress update meetings

2.2.5. The Bridge Platform team

- ROLE
  - ✓ Develop bridge firmware
- REPRESENTATIVE
  - ✓ Erik Maas
- RESPONSIBILITIES
  - ✓ Provide relevant domain knowledge
- ACCEPTANCE CRITERIA
  - ✓ Solution meets functional and non–functional requirements
- CONCERNS
  - ✓ Signing production firmware is done by using same development scripts which are maintained by developers
  - ✓ There is a clear procedure for signing and deployment with clear responsibilities about who is liable for the specific step
- INVOLVEMENT
  - ✓ Meetings on an ad–hoc basis

2.2.6. The Bridge Application team

- ROLE
  - ✓ Develop bridge application firmware
- REPRESENTATIVE

    - ✓ Ino Dekker
- RESPONSIBILITIES
    - ✓ Provide relevant domain knowledge
- ACCEPTANCE CRITERIA
    - ✓ Solution meeting functional and non–functional requirements
- CONCERNS
    - ✓ Signing production firmware is done by using the same development scripts that are constantly maintained by the software developers
    - ✓ The process of uploading firmware artifacts to the Hue portal is automated
- INVOLVEMENT
    - ✓ Meetings on an ad–hoc basis

2.2.7. The Integration & Validation team in China

- ROLE
    - ✓ Approve the release of the new firmware version for several devices such as the bridges, various lamps, sensors and switches
- REPRESENTATIVE
    - ✓ Paul Krekel
- RESPONSIBILITIES
    - ✓ Provide relevant domain knowledge and technical information regarding the needs and requirements of the project
- ACCEPTANCE CRITERIA
    - ✓ Solution meets functional and non–functional requirements
- CONCERNS
    - ✓ Signing certificates for production firmware are securely stored under highest possible security measures
    - ✓ There is an archive of all production builds and it is clear and visible which version is released on specific date
    - ✓ Specific configuration files related to particular products such as lambs and luminaires are archived and stored together with specific build artifacts
- INVOLVEMENT
    - ✓ Meetings on ad–hoc basis

2.2.8. Validation Engineer in Quality Assurance Department

- ROLE
    - ✓ Quality Assurance in the Software Development process
- REPRESENTATIVE
    - ✓ John Goor
- RESPONSIBILITIES
    - ✓ Provide relevant domain knowledge and requirements regarding the quality aspects of the delivered solution
- ACCEPTANCE CRITERIA
    - ✓ Solution meets functional and non–functional requirements
- CONCERNS
    - ✓ To have a clear explanation about the process for signing and deployment of Hue software (responsibilities, workflow)
    - ✓ Traceability in the process
- INVOLVEMENT
    - ✓ Meetings on ad–hoc basis

2.2.9. Digital Operations Department

- ROLE
    - ✓ Manages specific operations
- REPRESENTATIVE
    - ✓ Carlos Sierra

- RESPONSIBILITIES
  - ✓ Provide relevant domain knowledge and requirements related to the Digital Operations (DO) department
- ACCEPTANCE CRITERIA
  - ✓ Solution meeting functional and non–functional requirements
- CONCERNS
  - ✓ To distribute the software at a high level of security
  - ✓ To have traceability in the process
  - ✓ To be included in the specific steps in the process
- INVOLVEMENT
  - ✓ Meetings on ad–hoc basis

2.2.10. Eindhoven University of Technology (TU/e)

- ROLE
  - ✓ The university guards the educational interests of the university and the trainee
- REPRESENTATIVE
  - ✓ PDEng Program Director: Ad Aerts
  - ✓ TU/e supervisor: Tanir Ozcelebi
- RESPONSIBILITIES
  - ✓ Monitor, evaluate, assess and provide regular feedback on the project progress and deliverables
  - ✓ Provide relevant domain knowledge, references and contacts
  - ✓ Provide relevant information regarding the needs and requirements of the project
  - ✓ Monitor, evaluate, assess and provide feedback on the trainee's design process and qualities of the design
  - ✓ Review the final project report
- ACCEPTANCE CRITERIA
  - ✓ Timely report of the project deliverables
  - ✓ Design, implementation, project management and documentation that meet the level of a PDEng project
- INVOLVEMENT
  - ✓ During the entire project by continuous communication via meetings on ad-hoc basis with the PDEng trainee and regular monthly project steering group meetings.

2.2.11. PDEng trainee

- ROLE
  - ✓ Software designer, project manager
- REPRESENTATIVE
  - ✓ Igor Anastasov
- RESPONSIBILITIES
  - ✓ Responsible for complete project implementation
- ACCEPTANCE CRITERIA
  - ✓ Content of sufficient quality as to the level expected of a PDEng trainee

■

3.Problem Analysis

The first two chapters give a brief overview of the project's context and scope, and the stakeholders' interests and goals. This chapter focuses on the problem that the project is trying to solve, by analyzing not only the use cases and challenges of the software signing and deployment within Philips Lighting, but also the overall challenges of the security and software deployment processes.

3.1 Context

3.1.1. Software signing

Software signing is a process of digitally signing executables and scripts to confirm the software author and guarantee that the code has not been altered or corrupted since it was signed. The most common use of code signing is to provide security when deploying software. Almost every code signing implementation provides some sort of digital signature mechanism to verify the identity of the author. [8]

The efficacy of code signing as an authentication mechanism for software depends on the security of underpinning signing keys. As with other public key infrastructure (PKI) technologies[3], the integrity of the system relies on publishers or authors securing their private keys against unauthorized access.

Within Philips Lighting, the Security Officer inside the Integration & Validation team is responsible for storing the signing keys. Additionally, he is responsible for signing the production version of the appropriate Hue software. Storing the signing keys safely and securely is one of the main challenges and responsibilities of the Security Officer. There should be highest possible security measures in place in order to prevent unauthorized access to these keys. Another challenge that the Security Officer is facing now is the signing process itself. The current process includes many steps that are manual and therefore error prone. There is a specific software script that, combined with the secret keys, generates the signed production version of the Hue software. However, this execution is done manually with a real possibility of mistakes. Moreover, there are two versions of the production software that are generated and require diverse ways of signing. One version is needed for distribution to the device factories, and one version is uploaded to the Hue portal.

3.1.2. Software deployment in IoT world

The IoT combines smart devices and sensors with analytics and the cloud. This paradigm shift presents new challenges involving software distribution, updates, and security. The world is evolving into an "everything as a service" environment and the embedded industry is no different. Internet of Things applications make heavy use of the cloud and this new paradigm is essentially what differentiates IoT from traditional networked embedded systems. Software updates are essential. Within the context of IoT and cloud applications, the ability to soft-configure the system is critical and an essential part of the motivation for moving traditional networked embedded systems in this new emerging direction. These kinds of capabilities offer the ability to quickly deploy new features and capabilities at a fraction of the cost. New capabilities promise lower cost and increased revenue. The ability to quickly, securely, and flexibly update any cloud-based service is essential to take advantage of the benefits this environment provides. Further, within the cloud, adding new services can adversely affect the security of the existing hosted services. For these reasons, new tools, capabilities, and techniques are emerging to coordinate and synchronize software distribution. [9]

[3] https://en.wikipedia.org/wiki/Public_key_infrastructure

The software distribution process involves software developers, integrators, testers and software users. The developers create the software and utilize a variety of integrated development environments (IDEs), code repositories, automated test, and continuous integration tools. Once the production binaries are created and tested, these binaries need to be controlled, stored, and managed throughout the release. Sometimes these are called "binary artifacts." It is important for binary management solutions to integrate with popular repositories, build tools, and continuous integration servers. The other consideration is how developers store, publish, download, and distribute software. The cloud environment adds significant complexity. In many IoT instances, endpoints may be a variety of platforms with end users that may or may not upgrade in a timely manner. [9]

Within the Philips Lighting projects, the engineers are working toward continuous integration and deployment. Once the Integration & Verification (I&V) team has given their approval on release of specific software, the software running on, for example, the bridge, must be signed and uploaded to the Hue cloud for further distribution toward all consumer devices in the field. The bridge software must also be signed and securely sent to a factory as a running change on all devices that are being produced. There are several actions that need to be undertaken until the software is released. All these actions are consisted of manual steps.

3.1.3. Project goals

Improvement of the complete Software Development Life-Cycle (SDLC) is of immense importance to Philips Lighting. Engineers there tend to make the SDLC as automated as possible, without compromising security aspects of the systems. This project is part of this global movement and its successful delivery makes a step further in reaching these high-level company objectives. The project has two main goals. First goal is to optimize and automate the software release management workflow, with main emphasis on the automation of the code signing process. Second goal is to improve the security in these processes, such as storing and operating the highly sensitive software signing keys. The following section explains in detail the current process of Hue software deployment and pinpoints the possible issues that could happen with it.

3.2 Business Roadmaps

Improvement of the complete Software Development Life-Cycle (SDLC) is significant to the Philips Lighting too. Engineers there aim for continuous integration, deployment, and delivery processes in each aspect of the SDLC. They tend to make the SDLC as automated as possible, without compromising security aspects of the systems. This project is part of this global movement and its successful delivery makes a step further in reaching these high-level company objectives.

3.3 Design Opportunities

During the study of the Hue system domain and the problem analysis, the following design opportunities were identified, namely security, usability and reliability. First, the system shall have access to the keys used for signing the production Hue software. These keys are sensitive and the system needs to provide the highest possible security measures in order to prevent unauthorized access to them. Second, the system should replace a complete process. Therefore, the system should satisfy the requirements of the owner and the stakeholders of the project in a way that the system is made usable and intuitive. Finally, the system should be reliable. The speed of execution is not critical, but the correctness of the processes is crucial. It is preferred that processes run sequentially. Concurrency is not preferred since every action need to be atomic and therefore system should know its state in every moment of time. ■

4.Domain Analysis

The problem analysis discussed in the previous chapter reveals the domain in which the project resides, namely the Software Deployment Life-Cycle (SDLC) in the IoT domain and the software signing with specific emphasis on the security perspective of these processes. The objective of this chapter is to broaden the understanding of the domains by analyzing the core challenges in them and identify the relevant parts that provide insight for the solution to the problem.

4.1 Challenges in the SDLC in IoT world

Developing applications and devices for the Internet of Things (IoT) poses plenty of challenges. Because IoT systems may be expected to perform for many years, developers must plan for their entire lifecycle, from design through end-of-life. Demand for IoT solutions puts pressure on developers to deliver applications quickly without compromising security and performance. Enterprise customers expect their IoT systems to perform for many years, during which they will require regular attention and frequent upgrading to take advantage of advances in technology. That means developers must anticipate what systems will require in the future and plan for the whole lifecycle of devices and applications at the design stage, from development, configuration, and deployment through provisioning, management, monitoring, and, ultimately, decommissioning. [10]

Managing the IoT lifecycle presents a prominent level of complexity. IoT systems often entail connecting existing "brownfield" devices that were not designed for connectivity. Developers must also ensure that the connection is secure, which requires implementing protective measures at all levels – device, network, and cloud. Once devices are securely connected, operators need a way to provision them, which today often means literally going from device to device with a thumb drive and loading applications or performing upgrades manually. Finally, IoT systems face the challenge of integrating with enterprise systems that can aggregate, analyze, and act upon the data collected from devices, bridging operational technology (OT) and information technology (IT) systems that have historically been separated by air gaps. [10]

To bring greater simplicity and efficiency to SDLC process, developers and system operators need the means to configure, provision, and manage field devices remotely. Many organizations today have, at most, the means to connect devices and collect data via gateways, but lack the ability to "push" control instructions back to devices. With hundreds or even thousands of devices on a single IoT network, the cost of lifecycle management and the risk of failure can be extremely high. To better understand the capabilities needed to address these challenges, it is instructive to walk through the entire device lifecycle. [10]

4.1.1. Design and Development

The new challenges arising from IoT call for new, more flexible development methods that enable developers to build applications efficiently and deploy them to multiple devices. In the absence of a single standard of device connectivity, ease of integration with a variety of platforms is a necessity. A pre-configured development platform, optimized for IoT, which does not require the use of actual devices, can dramatically accelerate development and enhance efficiency, reducing the risk of errors and delays. Increasingly, software development is distributed among teams, often working in separate locations. With a cloud-based development platform that allows anytime, anywhere access, distributed development teams can collaborate more effectively and further speed the process. Security is a paramount concern in IoT development. An end-to end security strategy must be factored in across the application lifecycle at the design stage. Building in security functionality adds a layer of complexity that can be a drag

on development and interfere with the eventual performance of the application. Developers can mitigate this, however, by building on a platform using pre-configured, integrated software components in which many security issues have already been addressed. This reduces complexity, saves time, and lowers the risk of security gaps due to misconfiguration. [10]

4.1.2. Deployment

Once an application has been tested, developers need the means to deploy it easily to hundreds or thousands of devices in the field. A similar challenge is facing the Philips Lighting. This can be accomplished with a platform that includes built-in capabilities for the remote commissioning, provisioning, and customization of devices with new applications.

4.1.3. Solution

To overcome the complexity of IoT development, it is important to think of the system lifecycle not as a sequence of discrete steps, but in a holistic fashion. The key capabilities for managing the system lifecycle, therefore, need to be integrated.

These capabilities would include tools for:

- **Remote device management**: Sending out engineers to maintain and upgrade devices in the field is simply not viable in large-scale IoT deployments. System operators need to be able to manage devices from a central location in a closed-loop system from provisioning to end-of-life.
- **Virtualized application development**: Developers need the ability to customize and reconfigure deployed devices with new applications, using an abstracted target hardware platform that does not require proximity to an actual device.
- **Modeling and testing**: Development teams need to test applications thoroughly even during the debug phase, using simulation models capable of replicating the entire system across its entire lifecycle.

Finally, to enable distributed development teams to collaborate efficiently and effectively, it makes sense to house these tools in a secure cloud environment, whether on an internal server or through an external provider that allows anywhere, anytime access to authorized developers. [10]

This Section explores the challenges that arise at each stage of the lifecycle and how to address them, and outlines the advantages of having an integrated development environment for building, testing, deploying, and managing IoT applications.

4.2 Software signing

Software signing[4] is the method of using a certificate-based digital signature to sign executables and scripts in order to verify the author's identity and ensure that the code has not been changed or corrupted since it was signed by the author. This helps users and other software to determine whether the software can be trusted. [11]

Because of the potential damage that an executable or script can cause to a computer system, it is important that users are able to trust the code published on the Internet. There are two important ways that Code Signing increases trust, namely:

- **Authentication**. Verifying who the author of the software is.
- **Integrity**. Verifying that the software has not been tampered with since it was signed.

For example, every time a developer publishes a software application, he signs it with his own signing certificate. Before using the application, other users can clearly see

[4] Sometimes is referred to as *Code signing*

that it is signed by a known developer and they will know that it has not been changed by a hacker in the process of downloading it.

Another advantage that code signing provides is the ability to trust updates. If an update to a software application is released and signed using the same key as the original application, the update can be automatically trusted because it could not have come from anywhere other than the original trusted software developer or organization.

Almost every code signing implementation will provide some sort of digital signature mechanism to verify the identity of the author or build system, and a checksum to verify that the object has not been modified. The efficacy of code signing as an authentication mechanism for software depends on the security of underpinning signing keys. As with other public key infrastructure (PKI) technologies, the integrity of the system relies on publishers securing their private keys against unauthorized access. Keys stored in software on general–purpose computers are susceptible to compromise. Therefore, it is more secure, and best practice, to store keys in secured tamper–proof devices. Software is written code, and code can be read and analyzed. Once it has been analyzed, it can be modified to the requirements of an attacker. If a device is reprogrammed with modified software, the authentication process and system integrity can be broken. Another potential and severe weakness of software-based solutions is the inappropriate storage of secret keys during various process and production steps. Typically, in software–based protection systems, attackers can easily identify secret keys that are built into the software or otherwise stored in readable form. [8]

Setting up a secure storage for software signing keys is one of the challenges that the Philips Lighting faces. Therefore, one of the most important goals of this project is to find a more secure way of handling and storing these keys. ■

5.Feasibility Analysis

After elaborating the problem and domain analysis, a feasibility analysis is performed to identify and analyze issues and risks that exist or might appear. This chapter discusses issues and risks identified together with their mitigation strategies.

5.1 Issues

This section describes issues and challenges that we came across during the lifetime of the project.

i. Arranging meetings with all important stakeholders of this project is a challenging activity. The discussions with testing architects, security officers, different development teams, the product security department, and distributed teams in Bangalore are crucial input to the problem analysis and further definition of the use cases and requirements of the project. Given their busy everyday schedule, arranging discussions for exacting key requirements and evaluating the result is a challenging task that requires stakeholder management and time management skills.

ii. Having an appropriate deployment environment where the final results will be installed is critical for delivering the end product of this project as it affects the design, modeling, implementation, and validation steps of the project. Deployment of the end product introduces an extra level of complexity because of two reasons. First, there are several possibilities for deployment, such as internal Philips Lighting data center, existing IT infrastructure in the Home Department, and cloud providers. Second, if cloud providers are chosen, there are a lot of legal regulations that need to be satisfied in order to make a contract with some of these providers. Having complete deployment environment is a risk identified and elaborated in the following section.

iii. Finally, integrating the existing Hue software into the system-under-development means that several adjustments need to be implemented into the existing software for code signing. Currently, there are several types of Hue software, such as bridge and lamp. Some of these software files are signed with external signing software that runs on a specific operating system. To integrate completely into one system, some of the existing code signing scripts need to be migrated to a different operating system. Properly communicating those changes and defining who is liable for them now and in the future, adds additional complexity to the project itself.

5.2 Risks

5.2.1. Project risks

During the project, a number of risks are identified. Table 4 lists risks together with their impact on the project and corresponding mitigation strategy. A more proactive risk management is applied during the initial months of the project due to the uncertainties and not clearly defined milestones. Further through the lifetime of the project, this list was updated as the scope and complexity became clearer.

Risk ID	**R01**
Description	Stakeholders' unavailability
Impact	Some stakeholders might not be available for (detailed) discussions, so the input gathered might be limited.

Mitigation strategy	Make plans for meetings in advance. Always try to have a backup strategy in case important stakeholder is unavailable.
Risk ID	**R02**
Description	The Home department does not have dedicated servers for deployment of the final product of this project. This is understandable, since they are more oriented towards embedded development. Therefore, the author needs to properly communicate and choose a deployment option. There is a risk that this decision and implementation will take a longer time, having in mind all legal and security regulations that Philips Lighting has.
Impact	It will not be possible for the stakeholders to use the system.
Mitigation strategy	Plan deployment early. Specify acceptance criteria and start analyzing deployment options. Set a fixed deadline for the decision and implementation. Clearly communicate this risk with the stakeholders and together define a backup plan in case this process is not finished on time.
Risk ID	**R03**
Description	Not a feasible requirement. One requirement of the project stays that the system shall implement an automated process for software upload to device cloud. However, the device cloud system is owned by the Royal Philips and there is a chance that it is technically not possible to integrate with this system as stakeholder's desire.
Impact	It will not be possible for the stakeholders to automatically upload signed software artifacts to the device cloud.
Mitigation strategy	Analyze if this requirement is technically feasible. If not, then clearly communicate this to the stakeholders and together define a backup scenario.
Risk ID	**R04**
Description	Operating system incompatibility. As we briefly mentioned in the Section 5.1, there are diverse ways of software signing for different types of Hue devices. Some of them require Windows operating system and some of them are fully Linux dependent. Since the goal of the project is to integrate all signing processes into one automated system, some of those signing processes need to be migrated into a different operating system. However, this migration is a complete project and is not in the scope of this one. Therefore, there is a risk that the system will not be applicable for some of the device types.
Impact	The final product will not support all possible device types, such as Hue bridge and Hue lamps.
Mitigation strategy	Discuss with stakeholders who is liable for this migration. Define a proper timeline for the migration.

Table 5.1 Project risks

As addition to the risks listed in Table 5.1, a comprehensive Information Security Risk assessment was performed during the architecture, design and implementation phases on the project deliverable. The purpose of this assessment was to analyze how the system is exposed to the security risks happening in its environment. The output of this analysis was a risk register, that contains a list of information security threats that are relevant to this project. Each threat was given with the following details: vulnerability, threat source, threat explanation, likelihood of the threat, impact, mitigation strategy, risk level, and risk level after mitigation. The mitigation strategies proposed here influenced and steered some further decisions that were made.

6.System Requirements

This chapter describes the system requirements from the project "Signing and security of Hue software." After analysis of the domain and its problems, a set of requirements are extracted and formulated that must be satisfied for this project. This document presents these requirements, both functional and non-functional ones.

6.1 Introduction

Although this chapter is primarily intended for engineers, it does not specify each functionality with technical details. It is assumed that the reader is familiar with the Hue system, current project description, goals, domain and the problem statement.

6.1.1. Requirements gathering process

Both the problem and domain analysis reveal the core of the problem and its position in the bigger ecosystem. Still, a simple problem statement is not enough to proceed with solving the problem itself. What is necessary is a problem decomposition into small and traceable sub-problems that address a specific issue. This decomposition leads to the formalized requirements of the project.

The problem decomposition process is conducted by repeated discussions with relevant stakeholders and thorough analysis of the problem and domain. These repeated discussions reveal the fine details of the problem and its subcomponents, which further help to differentiate specific requirements for the project.

Two sets of requirements are identified: functional requirements and non-functional requirements. Each requirement has a priority assigned to it, which indicates the importance of that requirement. The following three categories are defined for priority, based on the descriptions defined by S. Brander [12]:

- MUST — absolute requirement of the specification.
- SHOULD — there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- OPTIONAL — the requirement is truly optional.

6.1.2. Scope

The scope of the project includes the processes of signing and deployment of the Hue software running on the following devices:

- Hue Bridges and
- Hue Lamps

The scope of the project starts after the moment the software is developed and the development phase is finished. There is no differentiation among different versions of the Hue devices. The project sees the software as a collection of files (artifacts) that are a complete product that need to be signed and further distributed.

6.1.3. Intended audience and reading suggestions

This chapter is intended for any individual user, developer, tester, project manager, or documentation writer who needs to understand the basic system specifications.

Here are the potential uses for each one of the reader types:

- **Developer**: The developer who wants to read, change, modify or add new requirements into the existing program must firstly consult this chapter and update the requirements in the appropriate manner so as not to destroy the

actual meaning of them and pass the information correctly to the next phases of the development process.

- **User**: The user of this system reviews the diagrams and the specifications presented in this chapter and determines if the software has all the suitable requirements and if the software developer has implemented all of them.
- **Tester**: The tester needs this chapter to validate that the initial requirements of this system correspond to the executable system correctly.

6.1.4. Definitions, acronyms, and abbreviations

Term	Definition
The software	In the current context, the term software is related to the specific Hue software developed for a device such as the bridge device and the Hue lamp. Sometimes it is referred as "firmware," "software version," "software artifacts," and "release."
Software signing	In the current context, it refers to the activity of running the signing script together with production keys to build a production signed version of the software.
The Hue portal	The cloud system where software artifacts are uploaded for future deployment. Sometimes referred to as "device cloud," or "Hue device cloud".
I&V	The Integration and Verification team.
SO	The Security Officer from the I&V team responsible for signing the production software.
MAC	A media access control address (MAC address) of a computer is a unique identifier assigned to network interfaces for communication at the data link layer of a network segment.
Software developer	In current context, the software developer term can relate to the developer working in the following teams: Bridge Application, Bridge Platform, ZigBee Platform and ZigBee Products.

Table 6.1 Definitions, acronyms, and abbreviations

6.2 Overall Description

6.2.1. Product features and use cases

Six major features are identified from the requirements gathering process, namely, centralized location for software released versions, automated software signing, automated software distribution to the Hue portal, automated software deployment, automated upload of the software to the production factories, and automated distribution of the software for mobile application release.

i. The first feature, centralized location for firmware versions, serves as an archive of all production releases. Moreover, the system shall provide information about specific data related to a release. For instance, when the software is released, who approved specific step in the process and which files are related to that release.
ii. The second feature, an automated process for software signing, deals with the process of digitally signing software artifacts. The automation of this process helps Security Officers to replace the current manual error prone procedure with a fast, automated and reliable process.
iii. The third feature, an automated process for software distribution to the Hue portal, focuses on the process of uploading the software files to the device cloud. The current process is manual and therefore is error prone. Moreover, it requires a lot of communication overhead between the Testing Architects and the employees who are responsible for uploading the files.

iv. The fourth feature, an automated process for software deployment, deals with the process of deploying the artifacts on specific devices. Devices can be specified by their MAC addresses and/or their physical location. This process is also known as regional deployment. At the moment, it is also a manual process.
v. The fifth feature, an automated process for uploading software to factories, will allow the Security Officer to directly upload artifacts to the production factories. At the moment, this is also a manual step and requires a lot of unnecessary E-mail communication.
vi. The sixth feature, an automated process for distributing software for mobile application release, will allow the Security Officer to directly send artifacts to the mobile application development team. This is needed for further creation of the mobile application version release. This last step is not in the scope of this project.
vii. It is important to understand that all features from second to the sixth, are part of a larger workflow that deals with the entire process of distribution and deployment of the software release. Therefore, another feature of the system is to manage this workflow. There should be clear explanation of the process and responsibilities inside it. Each action in the process will be traceable.

The refinement of the requirements revealed also the main actors or users of the product. Table 6.2 lists the identified users and their main scenarios / steps within the system. The presented scenarios in the table are expressed from an abstract point of view. After refinement of the main requirements and use cases, the list of functional requirements was identified. The functional requirements are discussed in Section 6.3.

User	**Main scenario**
Admin	• Configure and manage the workflows, user, roles and permissions
Security officer	• Approve a step in the release workflow • Sign software files • Upload signed files to the production factories • View release workflow details
Software developer	• Approve a step in the release workflow • Start a new release workflow • View release workflow details
Test architect	• Approve a step in the release workflow • Send files to the mobile app development team • Deploy software on end devices • Upload the software files to the device cloud • View release workflow details

Table 6.2 Main users and scenarios

Figure 6.1 shows the system level use cases, which are based on the identified major features and actors within the product. Actors are linked to different use cases they can perform within the system. The use cases marked orange represent the main use cases of the system.

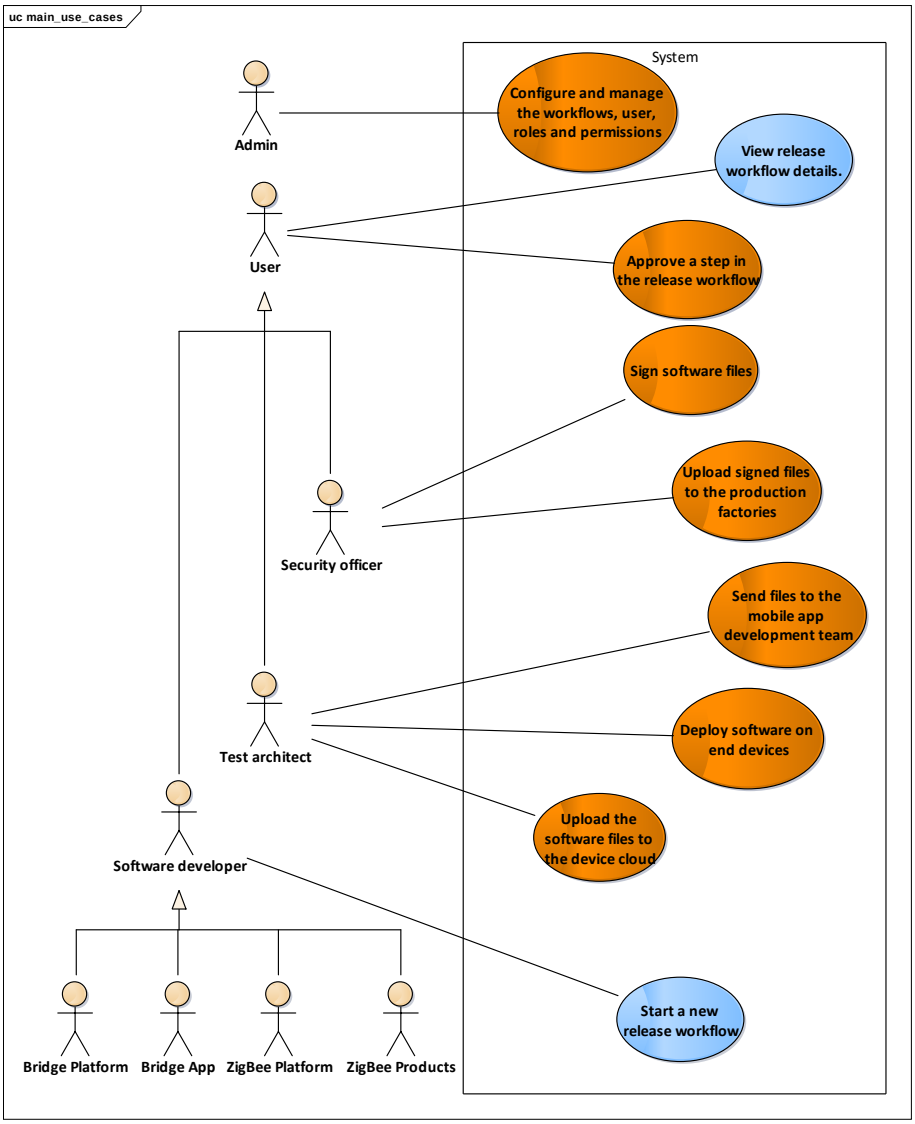

Figure 6.1 System level use-cases

6.2.2. Operating environment

One of the concerns and requirements that users of the system have is that it should be easily accessible via web browser from everywhere inside the Philips Lighting network. In this way, authorized users can sign software, approve specific step in the workflow, upload the software to the Hue portal or factory or send the artifacts to the mobile application team. Figure 6.2 depicts the envisioned deployment of the system.

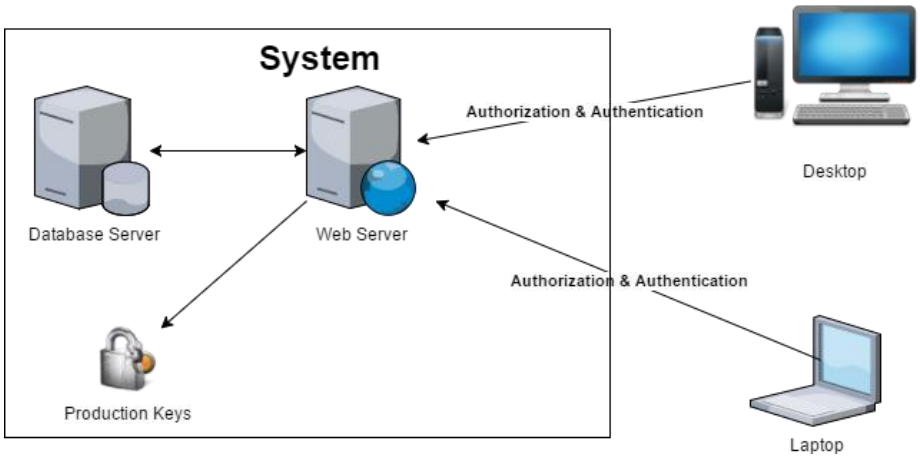

Figure 6.2 Envisioned deployment of the system

6.2.3. Design/Implementation constraints

The following technical constraints need to be taken into consideration during the design and implementation phases. First, a Linux[5] based operating system is preferred option for the system where the software signing processes will run, because the software signing applications are specifically developed for Linux operating system. Changing or adjusting these signing applications is not in the scope of this project.

The signing applications that are used to sign Hue software are coded in Python. Therefore, the application server where the system will run need to have the appropriate version of Python installed.

Regarding the licenses for the software libraries used in the implementation phase, open-source based licenses are preferred, specifically the MIT[6] and BSD[7] type of licenses.

6.3 Functional Requirements

This section includes the requirements that specify all the fundamental actions of the system. Each requirement has a brief description and an assigned priority.

ID	Description	Depends on	Priority
FR01	The system shall implement a process management tool (workflow) for managing releases of Hue software. This means approval / execution of all steps connected to the release process (signing, uploading, and distribution). The process should be clear about who is liable for specific step.		MUST
FR02	The system shall provide an interface to specify and customize the workflow. This means that authenticated and authorized users can define the workflow and who is liable for each step in it.	FR01	SHOULD
FR03	The system shall implement an authentication and authorization mechanisms for accessing	FR01	MUST

[5] https://en.wikipedia.org/wiki/Linux
[6] https://en.wikipedia.org/wiki/MIT_License
[7] https://en.wikipedia.org/wiki/BSD_licenses

	and for execution of specific step in the release process (user roles).		
FR04	The system shall implement an information overview regarding the status of each release process (what is the current state in the process)	FR01	MUST
FR05	The system shall provide traceability about each step executed in the process. For instance, who approved the Hue software for signing, who uploaded to the Hue portal, and /or who distributed to the mobile application team.	FR01 FR03	MUST
FR06	The system shall implement an interface that allows uploading the development software artifacts to it.	FR01 FR03	MUST
FR07	The system shall provide a historical overview of all signed production releases of the Hue software (archive). This includes: version number, type of device, release date, workflow overview (who did specific step of the release process), links to the software artifacts (files, configurations) and the I&V report.	FR01 FR03 FR05 FR15	SHOULD
FR08	The system shall implement an automated process for signing of Hue software production releases.	FR01 FR03	MUST
FR09	The system shall implement an automated process for software upload to the production Hue portal.	FR01 FR03	SHOULD
FR10	The system shall implement an automated process for software upload to the test Hue portal.	FR01 FR03	SHOULD
FR11	The system shall implement an automated process of deploying the artifacts on devices, specified by their MAC addresses.	FR01 FR03	SHOULD
FR12	The system shall implement an automated process of deploying the artifacts on specific region (regional deployment). This means that devices can be specified by their physical location.	FR01 FR03	OPTIONAL
FR13	The system shall implement an automated process for sending the software artifacts to factories.	FR01 FR03	MUST
FR14	The system shall implement an automated process for distributing software artifacts to the mobile application team.	FR01 FR03	SHOULD
FR15	The system shall implement an interface that allows uploading the I&V report to the specific Hue software that is to be signed.	FR01 FR03	SHOULD
FR16	The script for signing the production Hue software shall be same as the development signing script that is maintained by software developers.		MUST
FR17	The system shall provide notifications for specific actions or steps executed in the process. It should be configurable who receives notification regarding specific action.		SHOULD
FR18	The system shall provide an information overview of the deployment status. For instance, version X of the bridge software is detected on y% of connected bridges.		OPTIONAL

FR19	The system shall be able to stop a deployment. For instance, if the software is deployed in a wrong region, for wrong customers, or on wrong devices.		OPTIONAL
FR20	The system shall be integrated with the product database application. Integration will mean that each software has a link to the appropriate product database record and it is constantly maintained.		OPTIONAL
FR21	The system shall implement an interface for adding new signing production keys, by authorized and authenticated users.		MUST
FR22	The system shall implement an interface for revoking existing signing production keys.		MUST
FR23	The system shall maintain an audit trail (log) for each activity performed.		MUST
FR24	The system shall implement an automatic test process. This means that the system automatically checks that signed production software can be installed on a Hue bridge and a Hue bridge can be updated with the newest software release.		SHOULD
FR25	The system shall implement automatic notification actions for external systems for information purposes. For instance, interface to the Datadoc API that provides release information in Dashboards.		OPTIONAL

Table 6.3 Functional requirements

6.4 Non-functional requirements

Besides the functional requirements, several non-functional requirements are identified from the requirements gathering process.

6.4.1. Security requirements

The system shall have access to the production keys used for signing the production Hue software. These keys are sensitive and the system need to provide highest possible security measures to prevent unauthorized access to them.

6.4.2. Software quality attributes

Several software quality attributes are identified that the system should satisfy.

Usability

The system should satisfy the requirements of the owner and the stakeholders of the project in a way that the system is made usable and intuitive.

Maintainability

The system should be easily maintainable, meaning that the system can undergo changes with a degree of ease.

Extensibility

The system should be extendable. This means that it should be easy to add or modify features, for instance, adding new types of Hue software.

Reliability / Dependability

The system should be reliable. The speed of execution is not critical, but the correctness of the processes is crucial. It is preferred that processes run sequentially. Concurrency is not preferred since every action need to be atomic and therefore the system should know its state in every moment of time.

Minimum system that satisfies the requirements is defined as the system that implements all functional requirements marked with MUST and supports all non-functional requirements. ■

7.System Architecture

The previous chapter (Chapter 6) describes the functional and non-functional requirements of the system. This chapter explains the architectural reasoning and design decisions that are made and which resulted in a design of the system, based on the identified requirements. The purpose of designing an architecture for a system is solving a problem statement formalized with system requirements.

7.1 Architectural reasoning

The architecture of a system is a set of early key decisions that gives direction to the design, implementation and evolution of the system in its environment. The purpose of having a system architecture is to have one appropriate consistent system with balanced design decisions in line with the requirements of the stakeholders.

In agile processes, it is generally accepted that an early stage of the development process should be concerned with establishing an overall system architecture. Incremental development of architectures is not usually successful. While refactoring components in response to changes is usually relatively easy, refactoring a system architecture is likely to be expensive. [13]

A good starting point when designing an architecture is choosing a suitable architectural style or pattern. An architectural pattern is a set of principles that provides an abstract framework for solving frequently recurring problems. Formally, an architectural pattern is a description of element and relation types together with a set of constraints on how they may be used. [14]

An architectural pattern gives the backbone of the design. The pattern is typically chosen based on the non-functional requirements. Every pattern exhibits certain quality attributes, but not all can fit the non-functional requirements of a project or are not needed. For the purposes of this project, several key drivers steer the decision for an architectural pattern and later design of the system (ordered by importance):

1. High-level requirements
2. Non-functional requirements
3. Functional requirements.

One of the main requirements of the product (see Section 6.2.2) is for it to be a web application available to end users, such as security officers, test architects and software developers. Typically, web applications follow an n-tiered architecture pattern, where *n* is the number of tiers[8]. The simplest form of the n-tiered architecture is the 2-tiered web architecture or the client-server architectural style. Typically, when users browse the Internet, they use Web Browser software such as Internet Explorer, Google Chrome or Mozilla Firefox. The computer where the browser runs is called a client, whilst the machine which provides a Web page is called a server. [15]

When specific page is requested from the Web Browser, the user's computer forms a network connection to a Web server. In this situation, the computer is in effect a client, which is linked to a Web server. The web server, as the name suggests, serves the requesting browser with Web pages, such as HTML, ASPX, JSP and PHP pages. This simple scenario, where the Web server is connected to one or more clients, is known as a 2-tier architecture model. Figure 7.1 demonstrates how Web pages are accessed via a browser, using this 2-tier architecture.

[8] In the following text, terms *tier* and *layer* are used interchangeably

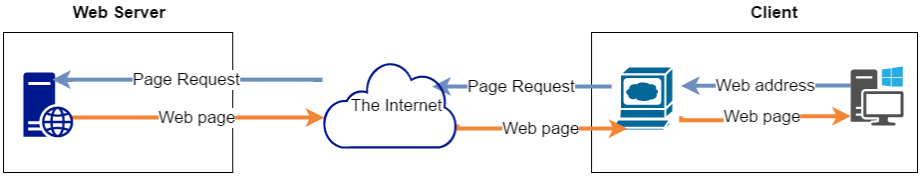

Figure 7.1 Client-server architecture

Generally computing applications consist of three different and distinct types of functionalities, namely presentation, logic, and data management. The difference with the client-server architecture is the addition of a specific layer that handles all business logic and computing. Every application includes some data processing and this may also involve database interactivity. For example, the user authentication module requires the logic unit to read user - role combinations from a database and check if the current user has permission to access a specific page. Starting from a client-server architecture, this addition of a new tier moves towards a 3-tier architecture, namely presentation, logic, and data tier. The 3-tier architecture is very common in today's systems, due to the qualities it exhibits, such as maintainability, scalability, flexibility, and availability. The basic idea behind this architecture is to separate the functionality into segments or tiers that can be deployed (not necessarily) on different physical computing platforms.

Since the nature of the project is that the system performs many actions in an automated way, naturally there is a need for a specific layer that handles all business logic, such as automation of software signing, approval of workflow steps and / or access rights management. Although it is not an explicit requirement for this project, there is a possibility to integrate this system with one existing system / service from Royal Philips. This system is used for release management of the software components that are not part of the Hue system software stack. This integration could possibly mean that this system will need to have access to the specific data from the database. Having a separate layer that only handles database related topics, will make this integration easier.

If we reflect on the non-functional requirements in Section 6.4, the 3-tier architecture suggests a good fit, because one of the non-functional requirements was maintainability. Moreover, it adds qualities not envisioned initially, such as testability. Because components belong to specific layers in the architecture, other layers can be mocked or stubbed, making this pattern is relatively easy to test. A developer can mock a presentation component or screen to isolate testing within a business component, as well as mock the business layer to test certain screen functionality. This is complemented with the functional requirements where unique features are identified, each with its own distinctive purpose. Since security is one of the most important non-functional requirements, another vertical layer that is spread over all three tiers is introduced, namely the security slice. The purpose of the security layer is to provide guidelines for the architectural decisions in order to provide high security standards in all tiers from the 3-tier architecture. Figure 7.2 depicts the proposed architecture of the system.

One of the core principles behind the client-server model (the predecessor of the 3-tier model) is separation of concerns. Each layer of the architecture has a specific role and responsibility within the application. For example, a presentation layer would be responsible for handling all user interface and browser communication logic, whereas a business layer would be responsible for executing business rules associated with the request. Each layer forms an abstraction around the work that needs to be done to satisfy a business request. For example, the presentation layer does not need to know or worry about how to get customer data; it only needs to display that information on a screen in a particular format. Similarly, the business layer only needs to get the data from the persistence layer, perform business logic against the data, and pass that information up to the presentation layer. This separation of concerns allows any tier to be changed, replaced, or tested without affecting the rest of the system if the communication interface remains the same. [16]

The deployment of a typical 3-tier web architecture consists of a presentation tier (front-end web server that hosts the web application), business logic tier (application server that hosts the business logic of the system), and data tier (database server that hosts the data itself).

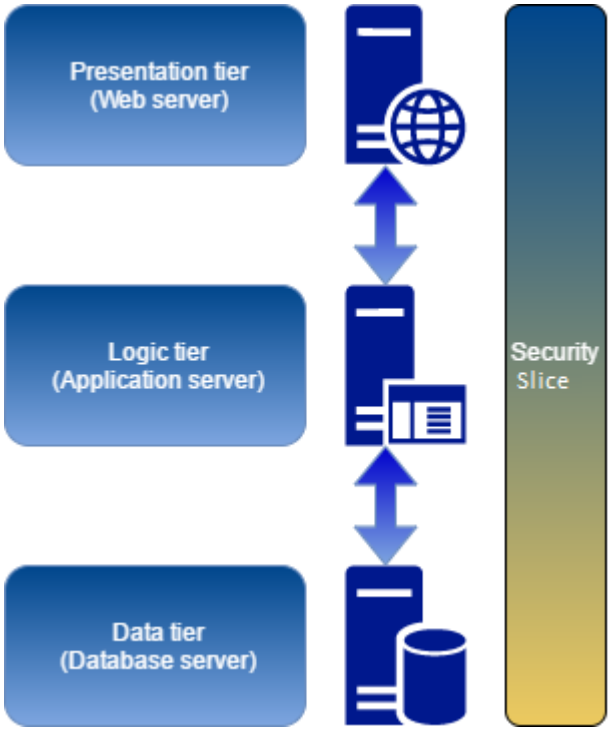

Figure 7.2 3-tier architecture deployment on separate physical servers

The following sections focus on the logical segments of each tier and the reasoning behind the architectural style selected for each of them. The order is bottom-up, starting from the data tier, to the logic and presentation tier.

7.2 Data layer

The data layer is responsible for data storage and manipulation. The manipulation involves serving the upper tier by executing its requests. The requests involve retrieving, storing, modifying or deleting data. Sometimes complex processing is also done on this tier in the form of stored procedures for optimization. The data tier typically consists of a physical computer (usually called server) that runs database management systems, which allow access to the data itself. These servers are high-end processing machines that can support many requests at the same time and heavy processing. Naturally, here a specific workflow details are stored, such as, status of the workflow, step execution details, responsible persons and user role management. [17]

7.3 Business logic layer

The logic layer is responsible for coordinating the application, processing of commands, and enforcing logical decisions related to business rules for the application in question. The responsibility extends to moving and processing data between its two surrounding layers. [18]

Since the system under development is in very initial stages, many modifications are expected in the future. Moreover, possible integration is expected with one existing system from Royal Philips, as we mentioned earlier in this chapter. With this in mind,

a structured and well-defined way must be easily extendable, both vertically and horizontally. Vertical extensibility means that new features can be easily added, and horizontal extensibility means existing ones can be easily extended.

Besides extensibility, ease of modification and the rest of the identified non-functional requirements steer the architecture choice into a component-based style. The main idea behind a component-based architecture is decomposition of the design into individual functional components that expose well-defined communication interfaces. The benefit of using components is that they are reusable, replaceable, easily extendable, and independent. [19]

Having a separate and independent logic-tier, means that there should be appropriate communication protocol established with the front-end part of the application (the presentation tier), as well as with other parts of the system. Typically, in web systems there are two ways of offering functionalities to another system, namely:

1. Through web services
2. Through shared libraries

On the one hand, a service-oriented approach will allow numerous services to be hosted and maintained separately. This will make probable future integration with different systems easier to implement. On the other hand, a complete shared library can serve as an interface between the presentation and the logic-tier. This way, we can be sure that code is executed sequentially and locally inside the main program. With this approach, the performances of the application will be increased since the code runs together with the native application code. Applying this approach will mean that the library will be deployed on the same machine together with the web application. Moreover, an API library is very useful when we want to interact with things with as little overhead as possible. The consequence is that there is a higher coupling between the API and the applications using it.

Since this system is completely developed from scratch, and the services it provides will be only accessible from the web application (presentation layer), a shared library is a good fit. Also, this is preferable because this way we can minimize the need for applying different security measures for the presentation and business layer. Since both presentation and the business logic tiers will be deployed on same machine, the tasks for maintenance and configuration of deploying machines will be simplified. Because of this reasoning, a shared library is chosen as a communication channel between the presentation and business logic tiers. Still, the design of the business logic tier should be such that it is easily maintainable if changes are required in future. We explain in more detail how this is achieved in the next chapter 8 - System Design.

If we reflect again on the non-functional requirements listed in Section 6.3, we can remember that security is the main consideration of this system. The most critical data that will be operated by this system are the production software signing keys and the software files. There will be only a few people who will be authorized to access and operate with the production signing keys. However, since the automation of activities is one of the core ideas of this system, it means that these critical keys will be accessible from the server that runs the business layer. This situation introduces additional risk, because all users who can have access to the business layer (for instance, developers and system administrators), can potentially access the signing keys. This leads to the decision that the business layer should be split into different security zones, that can be potentially deployed in different physical servers with different security measures applied. First server will run the main web application and will interact with the presentation layer. Moreover, this layer will communicate with the database layer and operate with all workflow related data. The data that is stored in the database is mainly metadata for the release process, such as software version, friendly name, start and modification dates and status. These entries do not contain highly critical data. The second server will store the software signing keys and will run the signing applications. Since this is security critical task, this server will be securely isolated from the rest part of

the application. The second server will not be accessible from the presentation layer. Specific actions on this machine can only be initiated through pre-defined messages coming from the first server only. Based on the specific purposes they serve, these servers (and therefore applications they run) are named Release workflow and Software signer system respectively.

The split of the business logic layer into two different servers, implies that appropriate communication mechanism need to be in place in order to connect both servers in the business layer. In the world of distributed network systems, a typical way of offering functionalities usable by multiple systems is through services, specifically web services. A web service is a method of communication between two devices over a network. For simplicity, consider a service as a functionality that can be reused for different purposes. One of its purposes is to serve the needs of the system under design.

The idea of the services to be usable by other systems rather than only the system under design imposes additional requirements. Such requirements are:

- Services should be independent of the clients that use them;
- Services should not have any knowledge of the clients that use them;
- Services should provide a uniform interface.

Due to the variety of technologies nowadays, systems are built using different programming languages and different technologies. For the services to be independent and unaware of their clients, they need to exchange data in such a manner that is indifferent to the programming language used to build the client systems.

Such a way is achieved through exposing an Application Programming Interface (API), or in case of the system under design, a web API. A web API is a programmatic interface to a defined request-response message system, typically expressed with languages such as JavaScript Object Notation (JSON) or Extensible Markup Language (XML). These APIs are exposed via the web by using the Hyper Text Transfer Protocol, or HTTP (see section 7.3.2).

From a functional point of view, the first server will request functionalities offered by the services or APIs (second server), which in turn will process the request and return appropriate response, if needed.

There is an architectural style that encourages development of scalable web services (APIs), uses well-known protocol for communication (HTTP), and promotes independence from clients that use them: the Representational State Transfer architectural style, or REST.

The following sections present the architectural styles and protocols that play a role in the architecture, and finally, section 7.3.3 presents the architecture of the logic tier.

7.3.1. Representational State Transfer (REST)

Representational state transfer (REST) or RESTful web services is a way of providing interoperability between computer systems on the Internet. REST-compliant Web services allow requesting systems to access and manipulate textual representations of Web resources using a uniform and predefined set of operations. Other forms of Web service exist, which expose their own arbitrary sets of operations such as WSDL and SOAP.

"Web resources" were first defined on the World Wide Web as documents or files identified by their URLs, but today they have a much more generic and abstract definition encompassing every thing or entity that can be identified, named, addressed or handled, in any way whatsoever, on the Web. In a RESTful Web service, requests made to a resource's URI will elicit a response that may be in XML, HTML, JSON or some other defined format. The response may confirm that some alteration has been

made to the stored resource, and it may provide hypertext links to other related resources or collections of resources. Using HTTP, as is most common, the kind of operations available include those predefined by the HTTP verbs GET, POST, PUT, DELETE and so on.

By making use of a stateless protocol and standard operations, REST systems aim for fast performance, reliability, and the ability to grow, by re-using components that can be managed and updated without affecting the system, even while it is running.

The architectural properties affected by the constraints of the REST architectural style are:

- Performance — component interactions can be the dominant factor in network efficiency;
- Scalability to support large numbers of components and interactions among components;
- Simplicity of a uniform interface;
- Modifiability of components to meet changing needs; and
- Portability of components.

There are six guiding constraints that define a RESTful system. These constraints restrict the ways that the server may process and respond to client requests so that, by operating within these constraints, the service gains desirable non-functional properties, such as performance, scalability, simplicity, modifiability, visibility, portability, and reliability. If a service violates any of the required constraints, it cannot be considered RESTful.

The formal REST constraints are as follows:

- *Client-server* — Separation of concerns is the principle behind the client-server constraints. By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components. Perhaps most significant to the Web, however, is that the separation allows the components to evolve independently, thus supporting the Internet-scale requirement of multiple organizational domains.
- *Stateless* — The client–server communication is constrained by no client context being stored on the server between requests. Each request from any client contains all the information necessary to service the request, and session state is held in the client. The session state can be transferred by the server to another service such as a database to maintain a persistent state for a period and allow authentication. The client begins sending requests when it is ready to make the transition to a new state. While one or more requests are outstanding, the client is considered to be in transition. The representation of each application state contains links that may be used the next time the client chooses to initiate a new state-transition.
- *Cache* — As on the World Wide Web, clients and intermediaries can cache responses. Responses must therefore, implicitly or explicitly, define themselves as cacheable or not to prevent clients from reusing stale or inappropriate data in response to further requests. Well-managed caching partially or completely eliminates some client–server interactions, further improving scalability and performance.
- *Layered system* — A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way. Intermediary servers may improve system scalability by enabling load balancing and by providing shared caches. They may also enforce security policies.
- *Code on demand* REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This improves system extensibility but reduces visibility.

- *Uniform interface* — a distinguishable central feature of REST is a uniform interface between components. Implementations are decoupled from the services they provide. The trade-off is efficiency, uniform interfaces degrade efficiency, since information is transferred in a standardized form rather than format specific to an application's needs. REST defines four interface constraints: identification of resources, manipulation of resources through representation, self-descriptive messages, and hypermedia as the engine of the application state.

Web service APIs that adhere to the REST architectural constraints are called RESTful APIs. HTTP-based RESTful APIs are defined with the following aspects:

- Base URL, such as http://api.example.com/resources/;
- An internet media type that defines state transition data elements (e.g. XML, and images);
- Standard HTTP methods (e.g., OPTIONS, GET, PUT, POST, and DELETE) [20]

7.3.2. Hypertext Transfer Protocol

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, and hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web. Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text. HTTP is the protocol to exchange or transfer hypertext.

HTTP functions as a request–response protocol in the client–server computing model. A web browser, for example, may be the client and an application running on a computer hosting a website may be the server. The client submits an HTTP request message to the server. The server, which provides resources such as HTML files and other content, or performs other functions on behalf of the client, returns a response message to the client. The response contains completion status information about the request and may also contain requested content in its message body.

A web browser is an example of a user agent (UA). Other types of user agent include the indexing software used by search providers (web crawlers), voice browsers, mobile apps, and other software that accesses, consumes, or displays web content.

HTTP is designed to permit intermediate network elements to improve or enable communications between clients and servers. High-traffic websites often benefit from web cache servers that deliver content on behalf of upstream servers to improve response time. Web browsers cache previously accessed web resources and reuse them when possible to reduce network traffic. HTTP proxy servers at private network boundaries can facilitate communication for clients without a globally routable address, by relaying messages with external servers.

HTTP is an application layer protocol designed within the framework of the Internet protocol suite. Its definition presumes an underlying and reliable transport layer protocol, and Transmission Control Protocol (TCP) is commonly used. However, HTTP can be adapted to use unreliable protocols such as the User Datagram Protocol (UDP), for example in HTTPU and Simple Service Discovery Protocol (SSDP).

HTTP resources are identified and located on the network by Uniform Resource Locators (URLs), using the Uniform Resource Identifiers (URI's) schemes http and https. URIs and hyperlinks in HTML documents form inter-linked hypertext documents.

HTTP/1.1 is a revision of the original HTTP (HTTP/1.0). In HTTP/1.0 a separate connection to the same server is made for every resource request. HTTP/1.1 can reuse a connection multiple times to download resources, after the page has been delivered. HTTP/1.1 communications therefore experience less latency as the establishment of TCP connections presents considerable overhead. [21]

7.3.3. Business-logic layer architecture

The system design is a combination of two architectural styles, namely the component-based style and the REST style. The component-based design allows features to be independent, while REST allows distribution to security risky services to different, isolated servers. In the system-under-design this is the Software signer system, mentioned early.

Figure 7.3 shows the business-logic layer architecture. The functionalities of the "Release workflow system" are exposed through shared library, which uses the components of the system to process the requests. The functionalities of the "Software signer system" are exposed through REST services, which in same way uses the components of the system to process the requests.

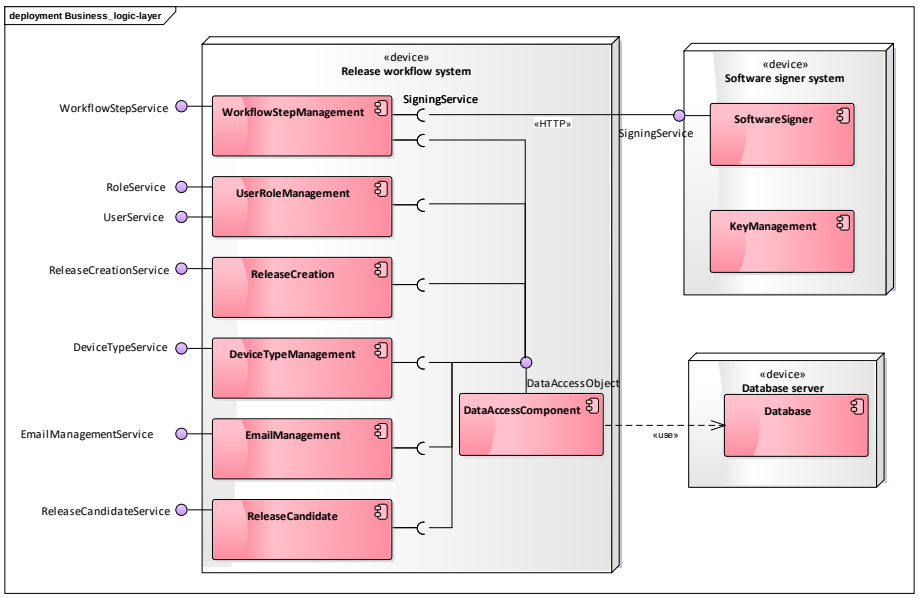

Figure 7.3 Logic tier architecture

The goal of this modular design is to separate the concerns of each component. The "Release workflow system" components handle the specific processing they are intended to perform. The WorkflowStepManagement component is responsible for execution of specific step in the release workflow. This step can be from different type such as approval, software signing or software upload to the cloud. More details for the design of this component are given in the Chapter 8 - System Design. UserRolemanagement component is responsible for managing users, roles and role permissions for specific parts of the system. The role of the ReleaseCreation component is to allow creation and manipulation with release and release candidates entries in the system. The DeviceTypeManagement component is responsible for configuration of the system specific settings and operations, such as workflow definition per different device types. Moreover, this component is used to create a new device type in the system. The role of the EmailManagement component offers services related to E-mail receivers data, such as modifying, adding or deleting factory E-mail recipients. Finally, the role of the ReleaseCandidate component is to provide relevant information about particular workflow, such as: status, device type, software version, build number, last modification date and number of steps.

As one can observe from the diagram, all the components use the Data Access Object (DAO) component. The DAO component is responsible for database access and data manipulation, mainly to retrieve, or issue commands to insert, modify, or delete data. Depending on the database technology used, most of the database management systems support execution of procedures in the database for optimization in processing large datasets.

The role of the SoftwareSigner component of the “Software signer” system offers service related to software file signing processes. These services include initiation of software signing process, retrieving signed files and distributing the signed artifacts to different systems. KeyManagement component is responsible for the features related to generation and replacement of new production signing keys. Please note that this feature / interface is not directly exposed on the front-end application, from security reasons. There will be simple command line interface only exposed from the Signing subsystem that will allow replacement of the production signing keys. Generation of the keys will be one-directional always. No one should be able to read / retrieve the signing keys.

The signing “SoftwareSigner” system does not depend on “Release workflow system”, meaning this RESTful API can be reused in different applications with different technologies.

Chapter 8 dives into more detail for the API and the components.

7.4 Presentation tier

The presentation tier is responsible for presenting data to the user and usually allows data manipulation and entry. The two main types of user interface for this layer are traditional applications and Web-based applications. If we reflect on the functional requirements for this project, the envisioned interface is a web-based application.

Over the years, the World Wide Web has evolved dramatically. Traditional web applications involved transitioning from page to page, where each page is generated dynamically on a remote server and sent to the browser where it is rendered. The beginning of the 21st century spawned the term Web 2.0. The term describes World Wide Web sites that emphasize user-generated content, usability, and interoperability. Parallel to the World Wide Web, a lot of progress has been made in modern web development. A shift has been made towards thin client-side Web-based applications that contain only the user interface (UI) and the data to be displayed. All the data processing is done in the logic tier. Page rendering is transferred on the client-side as the browsers became more and more powerful. [17]

Utilizing the advances made in modern web development, the choice of the architecture for the presentation tier should maximize the usability. The other requirements should not be excluded as well. Maintainability and scalability must be taken into consideration. Additionally, the coupling between the presentation layer and the business-logic layer should be as minimum as possible, so that it will support any changes or replacement of the layers in future if required. [17]

Over the years, user interface development has gone through many changes. One of the biggest change that happened in the recent years, is the use of MVC pattern for developing a web based user interfaces. The Model–View–Controller shortly known as MVC is a software architectural design for implementing web based user interfaces. The MVC pattern is a three-tier architecture that is independent of the programming language used for the system development. In the next Section 7.4.1 we present the core principles of the MCV pattern and the reasoning why it is chosen for the development of the presentation layer in this system.

7.4.1. Model View Controller (MVC)

Model–view–controller (MVC) is a software architectural pattern for implementing user interfaces on computers. It divides a given application into three interconnected parts in order to separate internal representations of information from the ways that information is presented to and accepted from the user. The MVC design pattern decouples these major components allowing for efficient code reuse and parallel development.

Traditionally used for desktop graphical user interfaces (GUIs), this architecture has become popular for designing web applications and even mobile, desktop and other clients. Modern programming languages like Java, C#, Ruby, PHP and others have popular MVC frameworks that are currently being used in web application development straight out of the box.

The core components of the MVC pattern are:

- *Model* means data, which is required to display in the view. It can sometimes be the exact data entities that are retrieved from the business layer or a variation of it. Model encapsulates business tier.
- A *view* can be any output representation of information, such as a chart or a diagram. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants. In MVC pattern view should be simple and free of business logic implementation. View invokes methods on Controller depending on user actions. In MVC pattern View monitors the model for any state change and displays updated model. Model and View interact with each other using the Observer pattern.
- The third part, the *controller*, accepts input and converts it to commands for the model or view. Controller is invoked by view, it interacts with the model and performs actions that updates the model.

The main advantages of applying the MVC pattern are:

- Simultaneous development — Multiple developers can work simultaneously on the model, controller and views.
- High cohesion — MVC enables logical grouping of related actions on a controller together. The views for a specific model are also grouped together.
- Low coupling — The very nature of the MVC framework is such that there is low coupling among models, views or controllers
- Ease of modification — Because of the separation of responsibilities, future development or modification is easier
- Multiple views for a model — Models can have multiple views [22]

The following Figure 7.4 depicts the structure of MVC pattern.

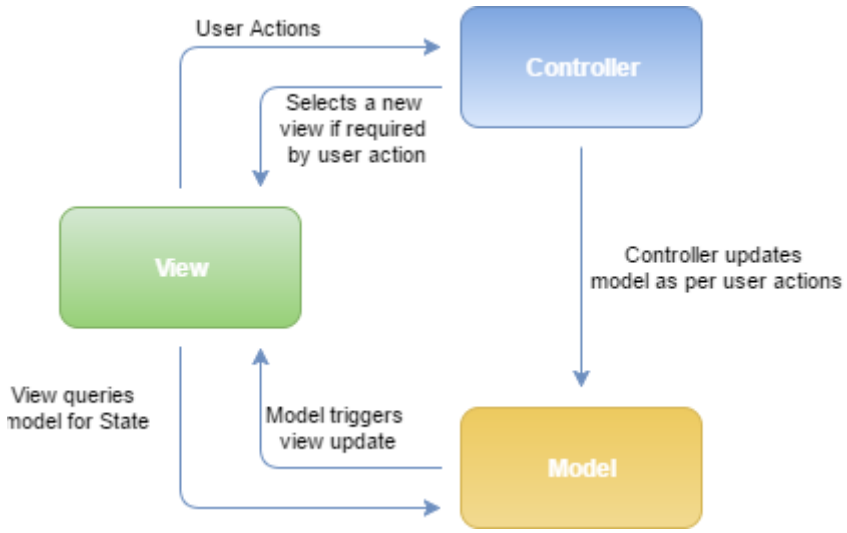

Figure 7.4 MVC design pattern

7.4.2. Architecture choice for presentation tier

The reasoning behind the architecture is based on the user interface point of view. As we mentioned in the previous chapters, the system should replace the existing processes for software signing and deployment. Therefore, the usability of the application is a crucial factor for future usage. Usability by itself is defined with several other non-

functional requirements, such as understandability, learnability, and attractiveness. Maximization of the usability means that the product is easy to learn and use, and is also attractive to the user. Another core reason is that many changes are expected in this system in the future, in both presentation and business-logic layer. By knowing this, we would like to have as low coupling as possible between these two layers.

Based on the facts above, the MVC architecture is chosen for the presentation layer. The main reason is the low coupling between the Controller, the View and the Model concepts. With this decoupling, it should be relatively easy to replace the complete presentation layer and integrate the system with another user interface if it is required in the future. This is in line with one of the potential future integration with one existing system from Royal Philips.

Figure 7.5 shows the high-level architecture of the presentation tier. Again, the component-based or modular style is followed to model the separate features. The architecture consists of five different components that cover the desired functionalities of the system under design and several other components for various purposes, such as visualization libraries to support the visualization or JavaScript and AJAX libraries to implement asynchronous communication. Each of these five components interacts with the interfaces provided from the business-logic layer, more precisely the provided interfaces from the "Release workflow system," explained in the Section 7.3.3.

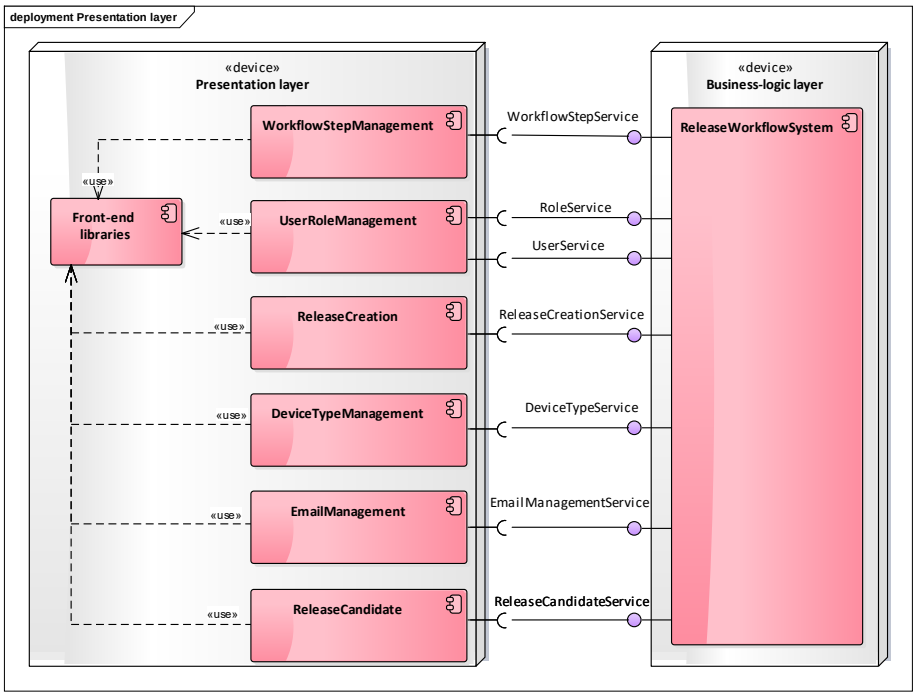

Figure 7.5 Presentation tier architecture

The five application-specific components follow the MVC pattern, with each module having a separate view, model, and controller.

7.5 Trade-offs between non-functional requirements

Previous Sections in this chapter explain the reasoning and set up the basic system architecture. Here, we would like to emphasize the trade-offs that were made between the non-functional requirements(NFR) of the system, listed in the Section 6.4. The most sensitive part of the system architecture is the security. Therefore, several trade-offs were made that increases the satisfaction level for this NFR, but on the other hand decrease the satisfaction level for the other NFR. Here, we list two most interesting examples.

1. Security vs Usability: According to the NFR, Security is the most important aspect of the system. Because of it, several security measures are implemented. One of them is the encryption of the signing keys and therefore having the need for the user to type in the decryption password every time there is a need for production software signing. This, however, limits the usability of the system by introducing another obligatory step in the code signing process. Usability, on the other hand, is another important NFR.
2. Security vs Maintainability: Another important NFR is the maintainability. The system should be easily maintainable and can undergo changes with a degree of ease. However, because of security reasons, we have decided to split the business logic layer and add additional layer that handles code signing processes. This decision is explained in detail in Section 7.3. This split adds more complexity to the system from maintenance point of view, because two different applications and two different servers need to be maintained.

Chapter 8 (“System Design”) elaborates on the specifics of the component design whereas Chapter 9 (“Implementation”) dives even deeper with concrete frameworks used and implementation specific details.

■

8.System Design

The previous chapter, “System architecture,” shows the high-level view on the system and its major components. The business-logic and presentation layers have different responsibilities and therefore distinctive designs are created. This chapter describes the two designs and the transition from high-level architecture to component design.

8.1 Introduction

In the previous chapters, we have identified stakeholders, set principles and guidelines, found the architectural constraints and in general the functional and non-functional requirements of the system. However, before we move into the next designing phase, it is worthwhile to analyze which models we want to create and what kind of view we want to use to communicate them.

IEEE 1471 defines an architectural view as a representation of an entire system from the perspective of a set of concerns, where the concerns are the key interests of the different stakeholders. A view is consisted of parts of one or more models to demonstrate how the concerns are covered. A view is an instantiation of the pattern defined in a viewpoint. [23]

For example, if we are concerned about concurrency and timing issues we are interested in threads and process models. Therefore, we can use views such as process diagrams and timing diagrams.

IEEE 1471 suggests that a viewpoint would hold the following information:

- Viewpoint name
- The stakeholders addressed by the viewpoint
- The stakeholder concerns to be addressed by the viewpoint
- The viewpoint language, modeling techniques, or analytical methods used
- The source, if any, of the viewpoint (e.g., author, literature citation)

A viewpoint may also include:

- Any consistency check associated with the underlying method to be applied to models within the view
- Any evaluation or analysis techniques to be applied to models within the view
- Any heuristics, patterns, or other guidelines that aid in the synthesis of an associated view or its models

In [24] Philippe Kruchten suggests 4 + 1 set of viewpoints, namely: Logical, Process, Development, Physical and Scenarios. Other examples are DODAF [25] (3 main viewpoints with 26 sub-viewpoints), RM-ODP [26] (5 viewpoints), CAFCR [27] (5 viewpoints) and Zachman Framework [28] (with 36 viewpoints).

For the system under development, the following viewpoints are identified:

- **Logical view**. The reasoning is that the system under development is developed by applying the object-oriented paradigm. Therefore, this view elaborates on the object model and the behavior of the design and gives direction for the further implementation phase.
- **Process view**: Although performance, scalability and throughput are not critical non-functional requirements of the system, it is important to understand how specific actions are performed in the system and how objects communicate between them. Therefore, activity or communication diagrams can be useful for modeling these interactions. Moreover, this view gives information

about which components are used / active during a specific process, like software signing. Additionally, the purpose of this view is to give information about which processes can run in parallel and / or sequentially.
- **Development view**: This view focuses on the static organization of the software in its development environment.
- **Deployment view**: With this view, we show how the software components are mapped into hardware devices.

Next sections elaborate the views defined in this introduction.

8.2 Reference use case

Chapter 6 - "System Requirements" discusses the key features and use cases of the product. In that chapter, several main use cases and a few auxiliary ones are described. The use cases represent the user stories in the system, or what the system is supposed to do in the interaction with the user.

As mentioned earlier, several scenarios are identified, such as signing a software image and sending signed software to factory. These scenarios involve participation of the end user. Furthermore, similarities between them exist. For example, most of the scenarios involve selecting a workflow and workflow step. Next, an action is triggered from the user to process a specific step. The request is processed, data is generated and returned to the end user in the form of visual information or feedback. With this, we can identify common steps for all these scenarios. The notation used to describe the common use case is:

- User – represents the end user, such as test architect or security officer
- Web application in browser – represents the presentation tier
- Release workflow system – represents the first level of the business-logic layer
- Software signer system - represents the second level of the business-logic layer

The steps are as follows:

1. The user requests to execute a feature;
2. The web application checks if the user has permission to execute requested feature;
    a. If user has permission, the web application shows the interface;
    b. If user does not have permission, interface is not shown and scenario finishes;
3. The user enters the required information and initiates action;
4. The web application verifies the information
    a. If the entered information is incorrect, the user is notified. No action is taken;
    b. If the entered information is correct, the scenario continues;
5. The web application sends a request to the "Release workflow system" to perform the requested action;
6. The "Release workflow system" accepts and processes the request;
    a. If the step requires communication with the Software signer system, an appropriate request is sent to the "Software signer system" and the returned response is processed.
7. The "Release workflow system" generates the requested information and sends it back to the web application;
8. The web application receives the information and displays it to the user.

8.3 Logical view

Both designs of the logic and presentation tier follow the component-based approach. Each component in the design has distinctive responsibilities, which makes it easier to

be replaced or modified. If we refer to the architecture decisions in Section 7.3, we can remember that we have decided to split the logic tier into two distinct parts that will be deployed on different servers, namely the Release management workflow and the software signer applications.

The architecture of the system includes a combination of two architectural styles, namely REST and component-based. The REST style is used to design the software signer application part and the component-based style to design the business or application logic of the release management workflow application. The following sub-sections describes the logical design with more details.

8.3.1. Release workflow system

The architecture of the release management application consists of several components. To be more understandable and simple, the designs shown in the following figures concern one chosen component, namely WorkflowStepManagement. This component is responsible for processing a specific step from the release workflow. If the name Workflow step is replaced with the name of the other components, such as UserRoleManagement, ReleaseCreation, or DeviceTypeManagement, the design of the other components can be obtained. Figure 8.1 depicts the architecture of the release management workflow application, starting from the web application controllers at the top and moving towards concrete implementations of the components at the bottom.

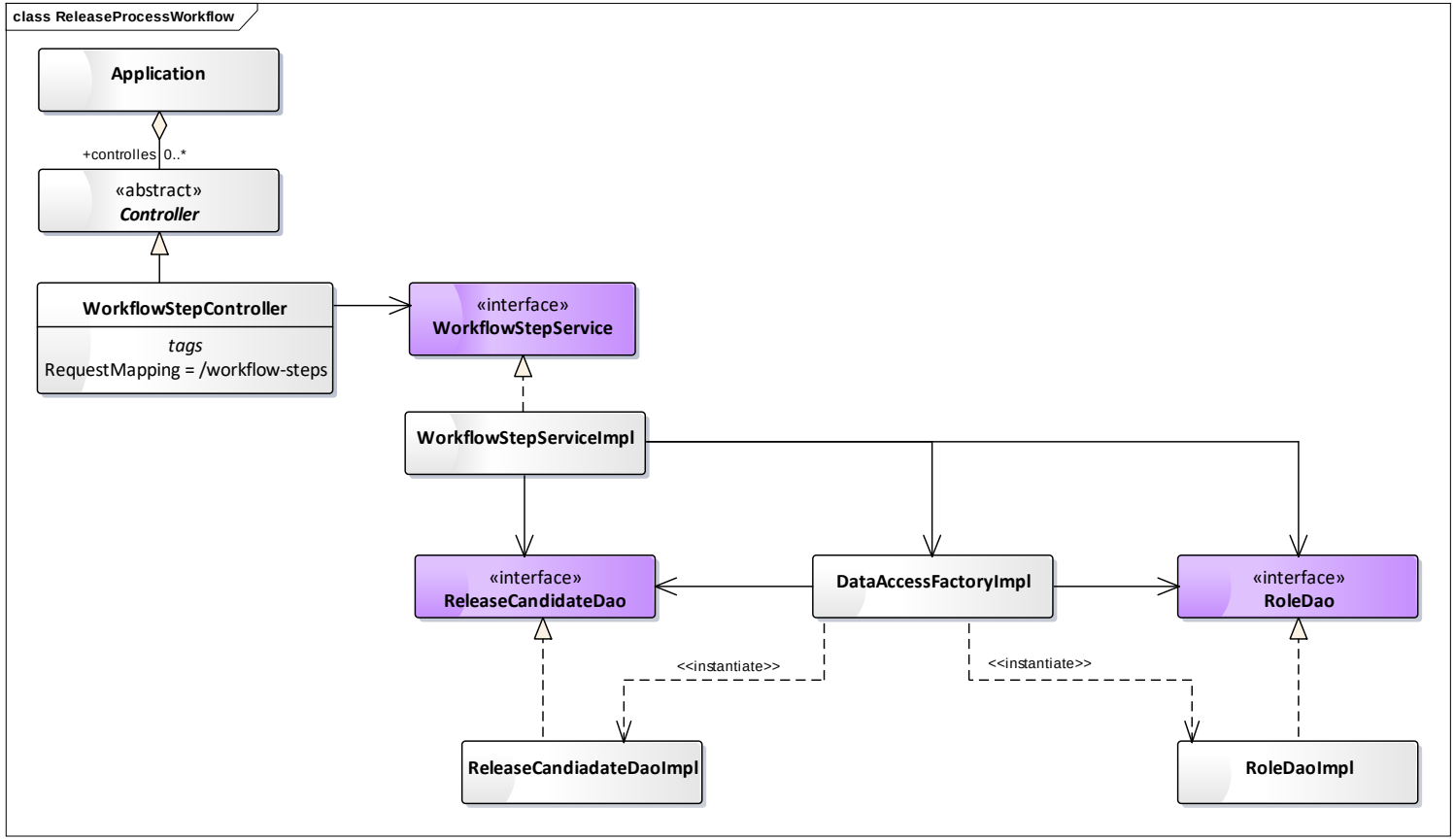

Figure 8.1 Logic tier architecture

The presentation layer is a web based application that consists of several controllers that handle requests coming from the web browsers. Each controller is responsible for handling requests for a specific component. For instance, the WorkflowStep controller is responsible for workflow step related requests, such as executing the step and getting step data from the database.

Each controller is identified by a segment of the URI it can handle. These URIs are uniquely identifiable access points or endpoints for each service offered by the web application. In Figure 8.1, this is shown through the Request Mapping section of the concrete controller WorkflowStepController and its RequestMapping value of "/workflow-steps." Every request with a URI that has a segment equal to "/workflow-steps" is handled by the WorkflowStepController. Furthermore, each controller exposes concrete endpoints that clients can use to request their service.

8.3.2. Software signer system

The design of the Software signer application uses a similar approach to the design of the release workflow application. The core difference is the main point of interaction. On the first one is the REST endpoints, while on the latter is a web browser. Since the functionality that this application does is limited, the design is simpler than the one of the release management application. Here, we have identified only one component, namely the SoftwareSigner component.

The REST-based API consists of only one controller, the SigningController. This controller is identified by a segment of the URI it can handle. Every request with a URI that has a segment equal to "/api" is handled by the SigningController.

Since we are following the same approach as the previous application, this controller also requires exactly one service interface. This level of abstraction helps to further separate the responsibilities of the REST service and the application logic, if one wishes to replace one of the two (with different implementations, for example).

8.3.3. Presentation layer

The architecture of the presentation tier is based upon the MVC design pattern. The pattern is very common in user interface development because of its separation of concerns, and ability to independently develop parts of the system, for example, views and controllers.

Figure 8.2 shows the design of the presentation tier and its connection with the logic tier. Again, the design shown, namely for the Workflow step, represents a design of a single module. The rest of the modules follow the same approach, if the name is replaced with the other modules' names, such as UserRoleManagement, ReleaseCreation, or DeviceTypeManagement. As one can observe, a module consists of a view, a controller, and a model. The view is responsible for the user interface, displaying data to the user, and user interaction. The controller is responsible for handling the interaction between the user and the view, hence the connection between them. Both the view and controller have references to the model. This is, of course, the nature of the MVC pattern itself. In this case, the model represents module-dependent information, specifically workflow step related information, such as, type of step and status. The design of the controller, in this case the Workflow step controller, is already shown in Figure 8.1.

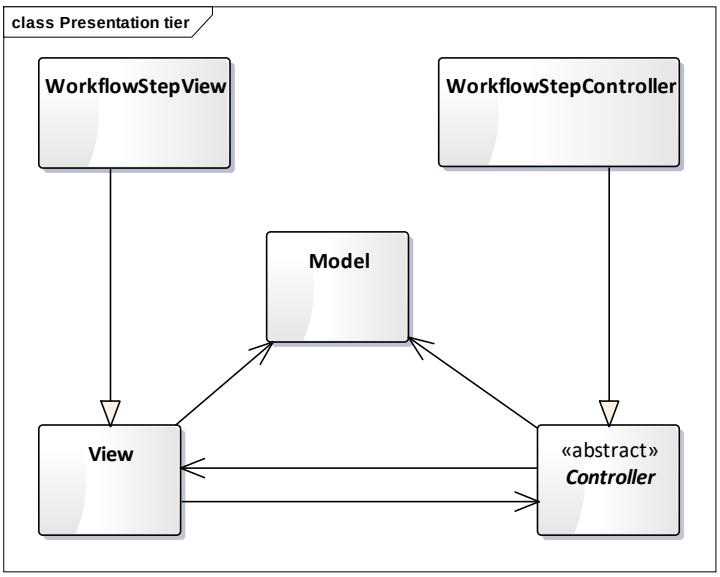

Figure 8.2 Presentation tier design

8.4 Process view

The process view is concerned with aspects regarding non-functional requirements, such as performance and availability. It mainly focuses on the run-time behavior of the system. Typically, the process view is explained using activity diagrams. As we already mentioned, the performance, scalability and throughput are not critical non-functional requirements of the system. Still, it is important to understand how specific actions are performed in the system and how objects communicate between them.

Figure 8.3 depicts the process view of the system, showing a request-response sequence, initiated from a User that wants to execute specific action in the system. Since the software signing process activates both systems of the business-logic layer (Release workflow and software signer), we assume that the User initiated a software signing process. In this scenario, all layers from the architecture are involved in the communication. There are simpler scenarios where the Software signer system is not involved in the communication. For instance, execution of an approval step from the workflow process.

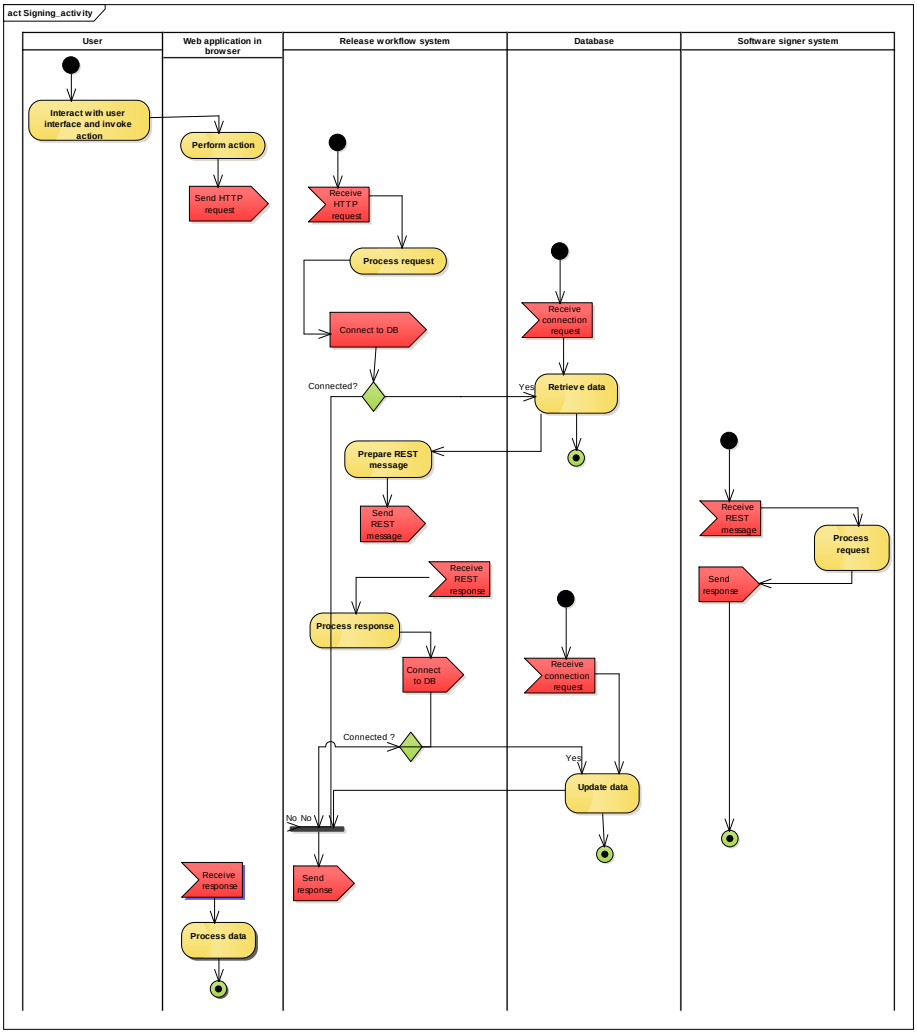

Figure 8.3 Handling signing request process. All layers are involved in this communication.

8.5 Development view

The development view focuses on the actual module organization within the software. This organization is expressed through packages or libraries organized in a way that they provide interfaces between them.

Figure 8.4 depicts the development view of the Release workflow system. The organization of the view is layered, so that each layer communicates with the layers next to it. The Business layer is on the top of the diagram composed of three sub-packages: for the interfaces, implementation and the workflow specific classes. The *interface* package defines interfaces that are the contract between the business-logic layer and the presentation layer. Their concrete implementation is contained within the *implementation* package. Finally, the *workflow* package contains the core implementation of the workflow execution functionalities.

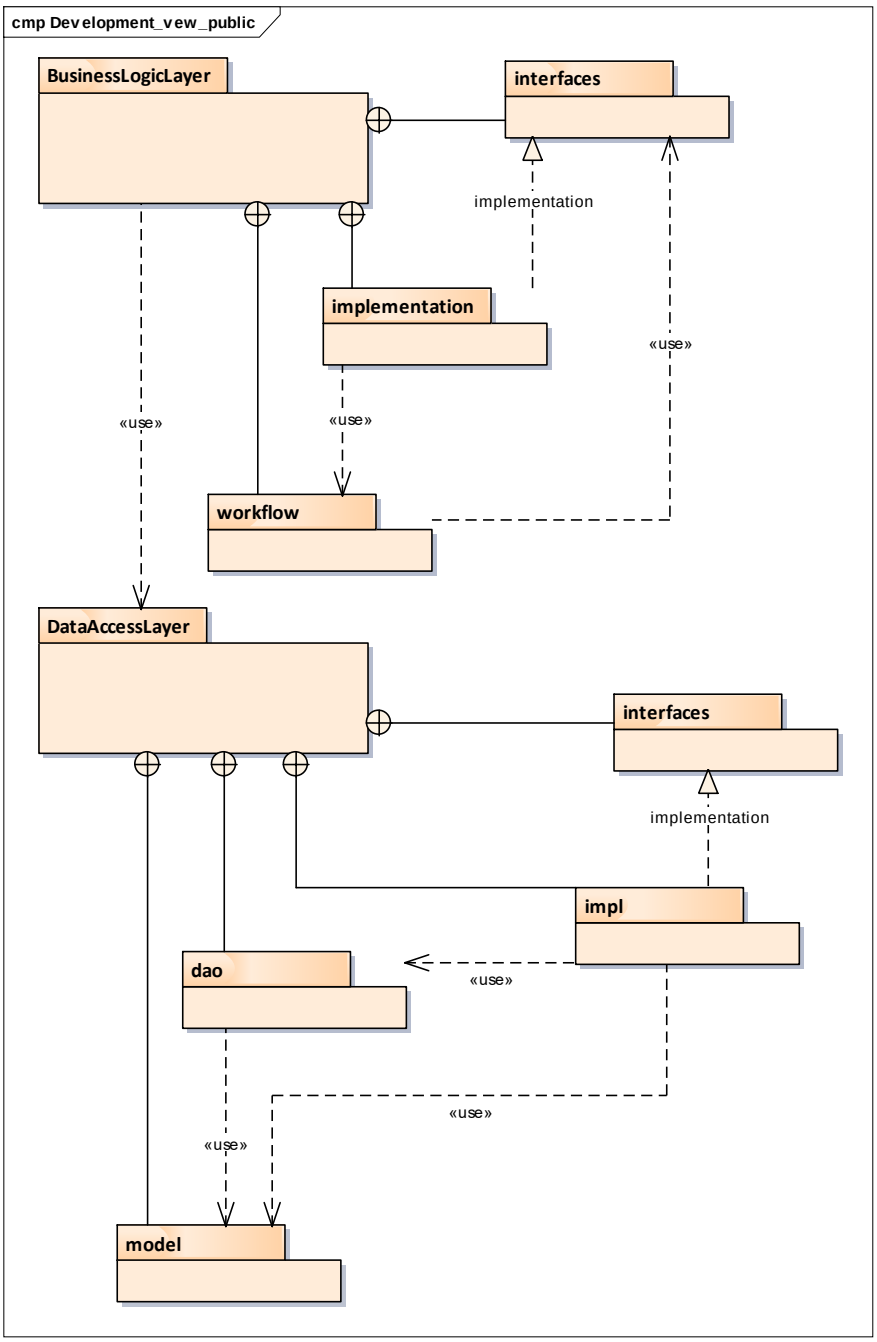

Figure 8.4 Development view of the Release workflow system

The layer below the Business Logic Layer package is represented by the Data Access Layer package. As the name suggests, this package includes the data access related interfaces and their implementation. The package is composed of four packages. First, the *model* package, contains the classes that represent the domain specific concepts such as *ReleaseCandidate*, *Step* and *DeviceType*. These concepts are further used in the second package, called *dao* which contains components whose main goal is to map the database raw entries into these concepts. Next, the *impl* contains the core implementation of the database related operations, such as retrieve and update data. The behavior of these components is defined as interfaces in the fourth package called *interfaces*. This organization allows concrete implementations or even complete packages to be replaced, which makes it easily maintainable.

The development view of the presentation layer is much simpler than the one of the logic layer. Figure 8.5 depicts this view. The tier is split into four distinct packages:

resources, which contains the user interface specifics such as views, HTML files, and JavaScript libraries; *controllers*, which contains user interaction handling; *admin*, which contains user interaction handling in the admin pages of the web application; and *webapp*, which contains configuration and settings components, such as the global exception handling controller.

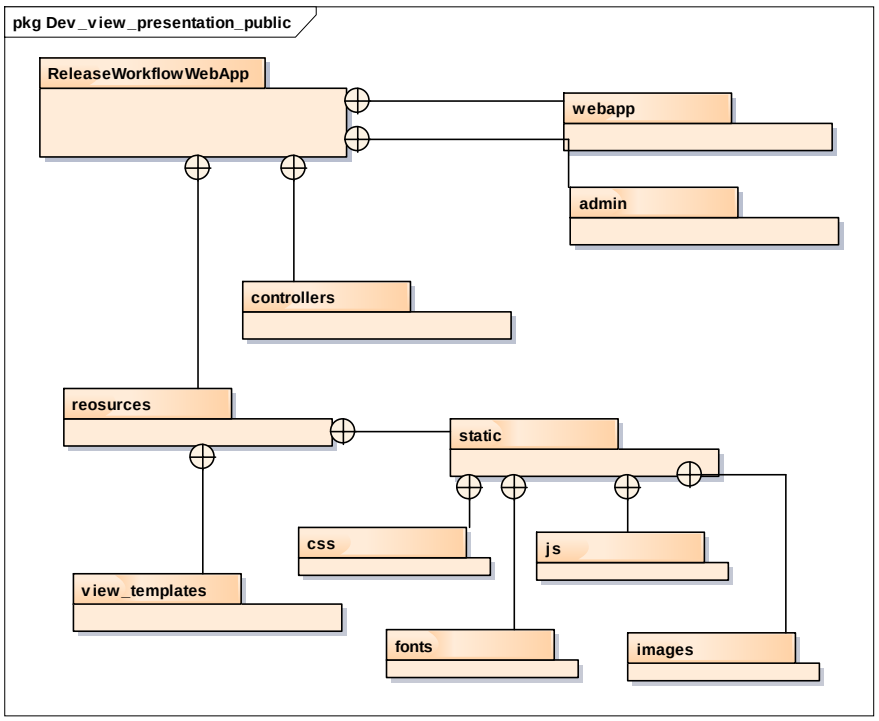

Figure 8.5 Development view of the presentation layer

8.6 Deployment view

The deployment view concerns the structure of the product after implementation regarding software to hardware mappings and distribution aspects. The Deployment view focuses on aspects of the system that are important after the system has been tested and is ready to go into live operation. This view defines the physical environment in which the system is intended to run, including the hardware environment that the system needs (e.g., processing nodes and network interconnections), the technical environment requirements for each node (or node type) in the system, and the mapping of the software elements to the run-time environment that executes them. [29]

Figure 8.6 depicts the designed deployment view of the system. The client, which is the web application executed on the end user's personal computer, makes requests via HTTP(s) to the Release workflow system, which runs on a remote server accessible via the internal Philips Lighting network. This web application communicates with the database server, which hosts the database that contains all data, and with the Software signer server which runs on another remote server accessible also via the Philips Lighting internal network. In order to elaborate the specifics for the deployment view with more details, the deployment view is revisited in Chapter 11.

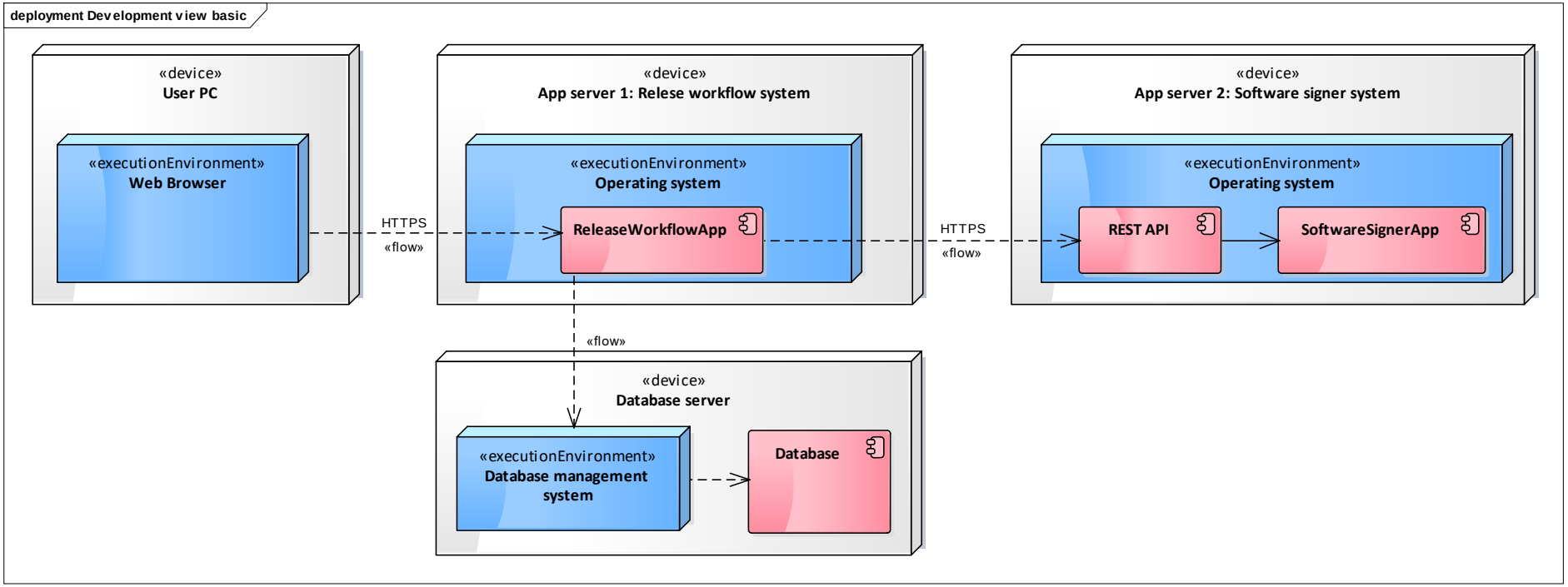

Figure 8.6 Deployment view of the entire system. A more comprehensive view is offered in the Deployment chapter

9.Implementation

The previous two chapters discussed the system architecture and design. This chapter explains the realization of the design, specifically the technologies chosen for its development.

9.1 Introduction

An implementation is provided as part of the project. The implementation is based on the architecture and design, elaborated in the corresponding chapters, and satisfies the functional requirements defined in the "System requirements" chapter. The implementation provided a demonstration of the feasibility of the project. For implementing the design, choices are made on which technologies to use. Since the product is comprised of several layers, several choices are made, one for each layer, in the architecture.

As we mentioned in Section 6.2.3, Design and implementation constraints, there were several technical and operational restrictions that influenced the decisions made in each layer of the architecture. First, the Linux[9] based operating system is preferred option for the application server that hosts the Software signer system, since most of the software signing software applications are specifically developed for Linux operating system. Next, open-source licensees were preferred for complete development, specifically the MIT[10] and BSD[11].

9.2 Presentation and business-logic layer: Release workflow application

The type of user interface for this system was decided when the requirements were gathered for this project. The stakeholders explicitly requested a Web application. This meant that one decision had already been made. Additionally, in the System Architecture we decided to bind the web application with the business-logic layer using a shared library. The next choices were the operating system, the framework, and programming language that was used for the development of these two layers.

As we briefly mentioned in the previous section, a Linux operating system was required for running the Software signer system. Because of this, we decided to use the same operating system for hosting the web application too. The main reasoning was to have unified operating systems for all layers. This way we could apply the same configuration and the same security measures are in place. Also, from the maintenance point of view, having same type of Operating System is a preferred option.

With the evolution of modern software development, many recent technologies and frameworks for both web and business-logic layer development have emerged that support both the architecture and the implementation constraints listed in Section 6.2.3. One requirement that needs to be supported by the technology chosen is the ability to connect to a database and manipulate data. Almost all modern programming languages support the architecture of the back-end part of the application. Based on the candidate's preference, Java[12] was selected as a language for the development of the release workflow application.

There are many frameworks in Java that supports web development. Therefore, choosing the right one is not an easy task. The main aspects that were considered are that this framework needs to support the MVC architecture pattern proposed in the System Architecture chapter. Also, the selection of the framework was based on the popularity,

[9] https://en.wikipedia.org/wiki/Linux
[10] https://en.wikipedia.org/wiki/MIT_License
[11] https://en.wikipedia.org/wiki/BSD_licenses
[12] https://www.java.com

online support, and again the personal preference of the candidate. For these reasons, Spring Web MVC[13] framework was selected as a framework for the development of the web application. The Spring Web MVC framework provides Model-View-Controller (MVC) architecture and ready components that can be used to develop flexible and loosely coupled web applications. Therefore, if we compare with our reasoning in the System Architecture chapter, we can conclude that the Spring MVC framework is a good fit for this architecture. [30]

Spring MVC supports many types of views for different presentation technologies, such as JSP[14], Hyper Text Markup language (HTML)[15], and Thymeleaf[16] templates. Again, based on the candidate preference, the Thymeleaf templates were chosen for the design of the user interface. The Thymeleaf completely relies on the standard Web development technologies such as HTML, JavaScript [17]and Cascading Style Sheets (CSS)[18]. Since usability was one of the non-functional requirements of this project, special attention was put on the user interface design. Users can use devices, such as personal computers, laptops, or tablet, to access it. In order to provide a consistent design, a specific approach is taken, namely responsive web design. One of the most famous and often used frameworks for developing responsive sites is Bootstrap[19]. Section 9.5 elaborates the user interface with more details.

9.3 Software signer system

The Software signer system consists of a REST API and the application logic modules. Again, there are multiple technologies and frameworks that support development of RESTful APIs and can be hosted on a Linux based operating system. Based on the candidate's preference, once more Java was selected as a language for the development of the Software signer system. The Spring Framework was used for the development, since it fully supports building the RESTfull services elaborated in the System design chapter.

9.4 Data layer

The main aspects when choosing the Database Management System (DBMS) were that it is fully supported in Java, can be hosted in Linux based operating system, and has open-source license. Therefore, the decision was made for the MySQL[20] database engine. MySQL is an open-source, free-to-use database management system, and it is widely used, especially in web systems.

■

[13] https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html
[14] http://www.oracle.com/technetwork/java/javaee/jsp/index.html
[15] http://www.w3schools.com/html/html_intro.asp
[16] http://www.thymeleaf.org/
[17] https://www.w3schools.com/js/
[18] https://www.w3schools.com/css/css_intro.asp
[19] http://getbootstrap.com/
[20] https://www.mysql.com/

10. Verification and Validation

The previous chapter described the implementation of the project. To ensure that the system is being correctly created, the process of verification and validation should be put in place. This chapter describes the process together with the techniques used.

10.1 Validation

Validation of a software product is the process of checking whether the product built satisfies the stakeholder's requirements. Two aspects need to be taken into consideration in the validation process. First, does the product built satisfies the functional requirements; does it do what it says in the functional requirements? Second, does the product satisfy the non-functional requirements? For the functional requirement, the validation should address whether the product offers the features defined. For the non-functional requirements, the validation should address how they are achieved and satisfied.

Following an iterative development approach (see chapter 13), the results were continuously validated by the stakeholders. Once per week, a meeting was scheduled with the main stakeholders that included a progress report and a demo of the current version of the product. During the demo, feedback was provided on the completeness and correctness of the product under development.

Additionally, the candidate was in the same department with the main future users of the system. With this setting, the users of the system were always up-to-date with the system implementation, so that they could validate it during the complete development process.

Regarding the non-functional requirements, the security was identified as the most important one. For this reason, a dedicated internal team of security experts was created, which main purpose was to evaluate the system architecture from security perspective and approve it for further implementation. Additionally, the candidate proposed performing a formal penetration test from authorized experts that will validate the security of the developed product. This activity is planned to be executed before the system goes live. At the moment of writing this report this is not performed yet, and therefore this is listed as future work (see Section 12.2).

10.2 Verification

Verification of a software product is the process of evaluating whether the system is engineered properly. This means that verification should check how well the product is developed, tested, and documented. To ensure the quality of the product, verification is performed by software testing and code quality measurements.

Since the product is split into two parts, namely web application - presentation layer and business logic layer, different testing is applied to them. In the web application, the user interface needs to be tested, as well as the code, and in the business logic layer the code needs to be tested.

On the user interface, manual testing was conducted. The user interface was tested with following browsers: Google Chrome[21] (version 60.0.3112.90), Mozilla Firefox[22] (version 54.0.1), and Internet Explorer[23] 11. This testing revealed several bugs as well as proposed several improvements. For instance, it was detected that page refresh was not

[21] https://www.google.com/chrome/
[22] https://www.mozilla.org/en-US/firefox/
[23] https://www.microsoft.com/en-us/download/internet-explorer.aspx

very convenient when user initiated specific action, such as software signing. Because of this, an AJAX[24] based interaction between user interface (the web browser) and the back-end part of the system was implemented, which prevented refreshing of the page when user was interacting with it.

The code in the business-logic layer was mainly tested manually, by running specific scenarios, as well as by performing unit tests on specific functions in the code. The testing environment where the system ran was the consisted of one server running all layers of the system. The Operating System for this server was Linux Ubuntu 16.04 LTS [25]. Unit tests were written in the JUnit[26] testing framework. JUnit is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks which is collectively known as xUnit. [33]

Finally, manual tests were performed on the system outputs. Since one of the key features of the system is the automated software signing, the output of the signing process was manually tested and verified. This included binary comparisons of the signed software files produced with the developed system and the files that are produced with existing manual process for software signing. This comparison proved that the system only automates the current process, without interfering and modifying it. ■

[24] https://www.w3schools.com/xml/ajax_intro.asp
[25] https://www.ubuntu.com/download/desktop
[26] http://junit.org/junit4/

11. Deployment

Based on the architecture and design of the system, a prototype is implemented that satisfies the stakeholders' requirements. This chapter elaborates on the specifics of the deployment considering the implementation choices regarding infrastructure, technologies and programming languages.

11.1 Deployment view

Based on the decisions for the system implementation (elaborated in the chapter 9), the development view defined in the Section 8.6 is revisited to take these choices into consideration and provide a clearer picture of the entire system. Figure 11.1 depicts the revisited deployment view of the system.

The main interface to the system is the web application that is delivered to the users on their PCs web browsers. The "Application server 1" host the web application, called "Release Workflow System". To run this application an environment that hosts a Java Virtual Machine is required.

The "Application server 2" hosts the RESTful API and the "Software signer system". The "Release Workflow System" sends HTTP requests and the "Software signer" server responds with HTTP response. To run the API and the services behind it, an environment that hosts a Java Virtual Machine is required. Moreover, to run the "Software Signer" scripts, a Linux operating system is required, with specific configuration.

Last, a database server that hosts the database is required for the deployment of the system. A management service is needed to handle connections to the database as well as management of the database itself. The JDBC acronym stands for Java Database Connector. MySQL is chosen as the database engine.

Section 11.2 explains the deployment options that were analyzed and steer the decision for the chosen option.

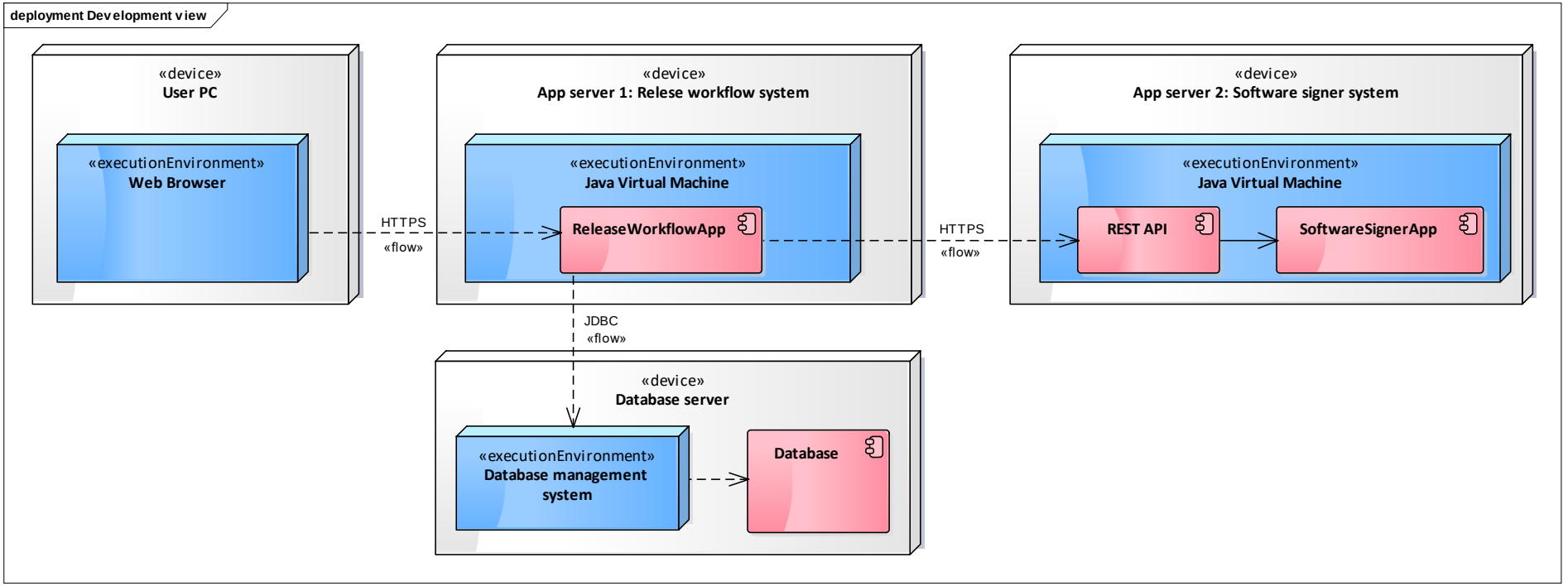

Figure 11.1 Deployment view revisited. Concrete implementation specifics are taken into consideration.

11.2 Deployment options analysis

Based on the deployment view presented in the previous Section 11.1 we have identified the following deployment options for hosting the complete architecture:

1. Deploying in the internal Lighting Data Center
2. Deploying in the existing IT environment in the Home Systems department
3. Deploying in the cloud computing environment. For this, the following options are considered:
    a. Amazon Web Services[27]
    b. Google Cloud[28]

The following sub-sections summarize the advantages and disadvantages of each option and steer the decision for the chosen one. If we reflect on the System requirements described in chapter 6, we can see that Security is one of the most important non-functional requirements of the system. The system shall have access to the production keys used for signing the production Hue software. These keys are sensitive and the system needs to protect them from unauthorized access. Therefore, when we analyze the advantages and disadvantages of each option, we considered the security requirements as the most influencing point.

11.2.1. Deploying in the Lighting Data Center

Philips Lighting has its own IT infrastructure (so called Lighting Data Center) where many internal applications are hosted and running. Therefore, considering deployment here was a natural option.

Lighting Data Center has available servers with Linux Red Hat distribution, which the system-under-development supports. These servers can be easily set up and scaled and adjusted according to the system needs. The Lighting IT department will be responsible for server maintenance and support for all server and hardware related issues. The Home Systems Department will be responsible for all application related issues, as well as the database related operations, such as maintenance, backup, and restore.

Since these servers are owned by the Philips Lighting Data Center (PLDC), the complete physical and logical security is a responsibility of the PLDC.

11.2.2. Deploying in the existing IT environment in the Home Systems department

Another option is to host the complete system in the existing IT infrastructure inside the Home department, where the main users of the system will be located.

However, there are many security risks if this option is considered. First, proper physical security measures must be in place for all PCs where the system components will be installed. Second, custom logical security measures must be implemented for the machines, so that the risk of unauthorized users getting remote ownership is prevented.

Additionally, complete maintenance of all servers will be responsibility of the Home department. This maintenance will include, but not be limited to: server updates, backups, restore and database maintenance.

11.2.3. Deploying in a cloud environment

Cloud computing and storage provide users with capabilities to store and process their data in third-party data centers. This allows organizations to better focus on doing what they are known for, and allow many of the technicalities be taken care of by an infrastructure that already exists and is constantly being upgraded. There are many cloud

[27] https://aws.amazon.com/
[28] https://cloud.google.com/

service providers. Because of the sensitivity of the data that could be stored and operated on these servers, only the following well-known providers were considered: Amazon Web Services and Google Cloud Platform. An additional reason is that there are already several services from Philips Lighting that are hosted on both Google Cloud and Amazon Web Services. When analyzing the cloud options, the following criteria must be satisfied:

- The solution should offer the possibility to define Virtual Private Networks so that all resources can be accessed only from the internal Philips Lighting network.
- The system architecture should not be dependent on the cloud environment. This means that the cloud solution should fully support the system architecture. At minimum, the following services are needed to support and host the system architecture:
  - ✓ Application servers for both the ReleaseWorkflowApp and the SoftwareSignerApp
  - ✓ File system with high level of security
  - ✓ Database server that supports MySQL engine

The following sub-sections analyzes cloud hosting options in more details.

Amazon Web Services (AWS)

The following services that are suitable for the system architecture are available from the AWS:

- **Application server:** Amazon EC2 is a web service that provides flexible compute capacity. New servers (instances) can be added within minutes, which are suited to scaling upwards or downwards at any time. Both application servers can be configured to run on the Amazon EC2 service. [31]
- **Database server:** Amazon RDS is a scalable and highly available relational database in the cloud which fully supports the MySQL database engine. Amazon RDS makes it easy to set up, operate, and scale MySQL deployments in the cloud. Amazon RDS allows the developers to fully focus on application development by managing time-consuming database administration tasks including backups, software patching, monitoring, scaling and replication. Our database server can run in the Amazon RDS. [32]
- **File storage:** Amazon offers two types of file systems, the Amazon Elastic File System (EFS) and the Amazon Elastic Block Store (EBS). EFS provides scalable file storage for use with Amazon EC2 instances in the AWS Cloud. Amazon EFS offers a simple interface that allows quick creation and configuration of the file systems. With Amazon EFS, storage capacity is elastic, growing and shrinking automatically as new files are added or existing ones are removed, so the applications have the storage they need, when they need it. All software artifacts and all generated files during the signing process can be stored on the Amazon EFS. [33]

  Amazon EBS allows creating storage volumes and attaching them to Amazon EC2 instances. Once attached these volumes can be used as standard block storage, for example, file system can be created or database can be run on top of them. Amazon EBS volumes are placed in a specific Availability Zone, where they are automatically replicated to protect users from the failure of a single component. All EBS volume types offer durable snapshot capabilities and are designed for 99.999% availability. Amazon EBS encryption offers seamless encryption of EBS data volumes, boot volumes and snapshots, eliminating the need to build and manage a secure key management infrastructure. EBS encryption enables data at rest security by encrypting data volumes, boot volumes and snapshots using Amazon-managed keys or keys that can be created and managed using the AWS Key Management Service (KMS). In addition, the encryption occurs on the servers that host EC2 instances, providing

encryption of data as it moves between EC2 instances and EBS data and boot volumes. [34]

- **VPN:** Amazon VPC enables organizations to have a private, isolated environment in the cloud. This is important feature, since it will enable private environment that can only be accessed from the internal Philips Lighting network. Additionally, a Hardware Virtual Private Network (VPN) connection can be created between the corporate data center and the VPC and leverage the AWS cloud as an extension of corporate data center. Amazon VPC provides advanced security features such as security groups and network access control lists to enable inbound and outbound filtering at the instance level and subnet level. Optionally, users can choose to launch dedicated instances which run on hardware dedicated to a single customer for additional isolation. [35]

Google Cloud Platform (GCP)

The following services that are suitable for the system architecture are available from the GCP:

- **Application server:** Google Compute Engine delivers virtual machines running in Google's data centers and worldwide fiber network. Compute Engine's tooling and workflow support enable scaling from single instances to global, load-balanced cloud computing. Compute Engine's VMs are designed to boot quickly, come with persistent disk storage, and are available in many configurations including predefined sizes or the option to create Custom Machine Types optimized for specific needs. Both application servers can be configured to run on the Amazon EC2 service. [36]
- **Database server:** Cloud SQL is a fully-managed database service that makes it easy to set up, maintain, manage, and administer relational PostgreSQL and MySQL databases in the cloud. Cloud SQL offers high performance, scalability, and convenience. Hosted on Google Cloud Platform, Cloud SQL provides a database infrastructure for applications running anywhere. Our database server can run in the Amazon RDS. However, Google Cloud SQL needs a specific way and syntax for accessing and manipulating data. This will require additional changes in the application source code which will make the system Google Platform dependent. [37]
- **File storage:** Google Persistent Disk is durable and high-performance block storage for the Google Cloud Platform. Persistent Disk provides SSD and HDD storage which can be attached to instances running in either Google Compute Engine or Google Container Engine. Storage volumes can be transparently resized, quickly backed up, and offer the ability to support simultaneous readers. Persistent Disks are automatically encrypted to protect your data. You can supply your own key or Google will automatically generate one for you. [38]
- **VPN:** With Google Cloud Platform (GCP) VPC, you can provision your GCP resources, connect them to each other, and isolate them from one another in a Virtual Private Cloud (VPC). You can also define fine-grained networking policies within GCP, and between GCP and on-premise or other public clouds. VPC is a comprehensive set of Google-managed networking capabilities, including granular IP address range selection, routes, firewalls, Virtual Private Network (VPN), and Cloud Router. [39]

Cloud options comparison and conclusion

Based on the comparison between Google Cloud Platform and Amazon Web Services cloud providers, we can conclude that both GCP and AWS support the architecture and the main requirements listed above. However, AWS has a slight advantage because of its database engine.

Google Cloud Platform offers two ways of hosting the MySQL database. The first requires stand-alone installation of MySQL server on the virtual servers. This option

will mean that all responsibility for maintenance and administration of the database will be on Philips Lighting, in same way as hosting in the internal premises.

The second option is to completely use Google Cloud SQL. However, Google Cloud SQL needs a specific way and syntax for accessing and manipulating data. This will require additional changes in the application source code which will make the system Google Platform dependent.

On the other hand, hosting the MySQL database in Amazon RDS instance means that the database is fully managed by Amazon and is accessible with the same existing source code.

11.2.4. Deployment decision

Table 11.1 summarizes the analysis made in the Section 11.2. Based on the analysis, an appropriate decision was made for the deployment of the developed system.

Deployment option	Security	Architecture support	Responsibilities
Lighting data center	– Additional custom logical security measures need to be implemented (For instance: Two - factor authentication and disk encryption).	– Fully supports the proposed architecture. – Environment can be scaled according to the system needs.	– Lighting DC is responsible for server and hardware related topics. – Home Systems department is responsible for all application related issues and all database maintenance tasks, such as backups and index rebuild.
IT environment at Home Systems	– Additional physical security measures need to be implemented. – Additional custom logical security measures need to be implemented (ex. Two - factor authentication, disk encryption).	– Freedom to customize the servers according to the system needs, so that they fully support the architecture. – If need for scaling, additional hardware need to be purchased.	– Home Systems department is responsible for complete hardware and software maintenance tasks.
Google Cloud Platform	– Virtual Private Cloud is supported. Connection to the resources is only available through VPN from Philips Lighting network. – File system encryption is possible.	– Fully supports the proposed architecture. – Environment can be scaled according to the system needs. – Application needs some adjustments in the source code for database connection, since Google Database Service has specific Google API for accessing.	– Google service provider is responsible for all hardware/server related issues and all database maintenance tasks, such as backups and index rebuild. – Home department is only responsible for application functionality.
Amazon Web Services	– Virtual Private Cloud is supported. Connection to the resources is only available through VPN from Philips Lighting network. – File system encryption is possible.	– Fully supports the proposed architecture. – Environment can be scaled according to the system needs.	– Amazon service provider is responsible for all hardware/server related issues and all database maintenance tasks, such as backups and index rebuild. – Home department is only responsible for application functionality.

Table 11.1 Deployment options comparison

12. Conclusions

This chapter analyses the results achieved by this project as well as the added value to the stakeholders.

12.1 Results

Developing software applications for the Hue devices poses unique challenges among the engineers at Philips Lighting. These challenges arise at each stage of the Software Development Life-Cycle (SDLC). Improvement of the complete SDLC is of immense importance to the Philips Lighting. This was the focus of this project, to make the process as automated as possible, without compromising security aspects of the systems.

The end result of this project solves a lot of challenges in the SDLC. It is a complete release management tool dedicated to the engineers at the Home department at Philips Lighting. First, it visualizes ongoing software release workflows in a simple and easy to use user interface. With a lot of searching and sorting possibilities, users are able to quickly find out the specific details about a particular software release process. Second, the core activities of the SDLC process are fully automated. The testing architects are now able to digitally sign software with just a few clicks on the web-based user interface. What is more important is that this signing is executed in a highly secure and protected environment. This is of a huge importance for Philips Lighting not only because this automation saves a lot of time, but also because it reduces the risk of a human error that was present before. The same benefits are gained through an automation of other activities in the SDLC, such as approval of steps, distributing the signed software to the Hue production factories, and deploying the signed software to the Hue device cloud. Third, the system provides a traceability about each step executed in the process. For instance, who approved the software for signing or who uploaded the signed files to the Hue device portal. Finally, the system is highly configurable, which makes it easy to be extended and adjusted to support different device types with different release workflows.

From the software architecture point of view, the designed system exhibits properties, such as modularity and maintainability, which makes it easily extendable. One of the core non-functional requirements of this project was security. This is achieved by applying advanced security mechanisms in every aspect of the system.

■

13. Project Management

This chapter elaborates on the management conducted throughout the project's lifetime.

13.1 Introduction

The management of the project was conducted under the agile methodology by using the Kanban approach. Kanban is a popular framework used by software teams practicing agile software development. Kanban gives teams flexible planning options, faster output, clearer focus, and transparency throughout the development cycle. This methodology is applicable in different domains, but it was initially forged for software development as an alternative to the traditional waterfall approach.

The project consisted of two parts. The first part of the project was focused mainly on Hue domain study and analysis, system environment analysis and requirements engineering. The second part of the project focused on design and implementation of the system based on the outcome of the first part.

The following sections dive into the details of the project, such as the work-breakdown structure, project plan and execution.

13.2 Work-Breakdown Structure (WBS)

The work-breakdown structure for the project is presented in Figure 13.1. Two top-level packages are identified to match the two parts discussed earlier. Each package is further decomposed into smaller packages. These packages are later referenced in the project plan and execution section.

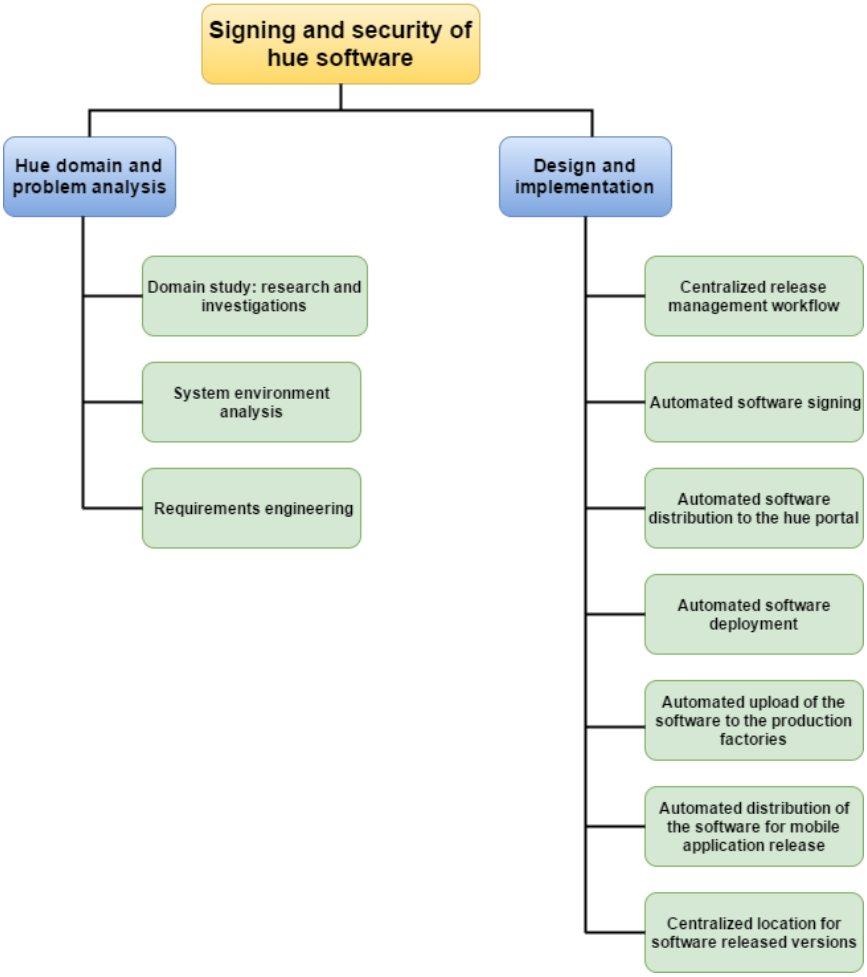

Figure 13.1 Work-breakdown structure of the project

The Hue domain study and analysis part consists of three packages:

- Domain study: Research and investigations — research and study about the Hue system, the current software signing process and the current software distribution procedure;
- System Environment Analysis — research and study about the software and hardware components that are composed into the current Hue system; and
- Requirements Engineering — formalization of the project requirements.

The Design and implementation part consists of seven packages, each with its own set of sub-packages. It includes:

- Centralize release management — design and develop a workflow based system that will be used for the release process of Hue software;
- Automate software signing — automate the current software signing process;
- Automate upload to the device cloud — automate the current process of uploading software images to the device portal;
- Automate software deployment to the end devices in the field;
- Automate upload to the factories — automate the current process of sending software images to the Hue production factories;
- Automate upload to mobile apps — automate the current process of uploading software images to the mobile application development team; and
- Centralized location for software images — design and develop a web based system that will serve as a centralized location of the Hue software images.

13.3 Project Planning and Scheduling

Based on the breakdown structure defined in the previous section, a project plan was formulated. As expected, the initial project planning defined in the beginning did not match the actual project execution due to the incremental approach. Several adjustments were introduced along the way, as the knowledge and understanding deepened. The next two sections show the initial and last version of the project plan in forms of a Gantt charts.

13.3.1. Initial

The initial version of the planning involved four major parts: the preparation and setup, for which the initial two weeks were reserved; the Hue domain study and problem analysis, for which the first two months were reserved; the design and implementation, for which six months were reserved; and project finalization phase, one month. The reporting on the project is considered an ongoing task for the entire lifespan of the project. Figure 13.2 shows a Gantt chart for the initial planning for the project.

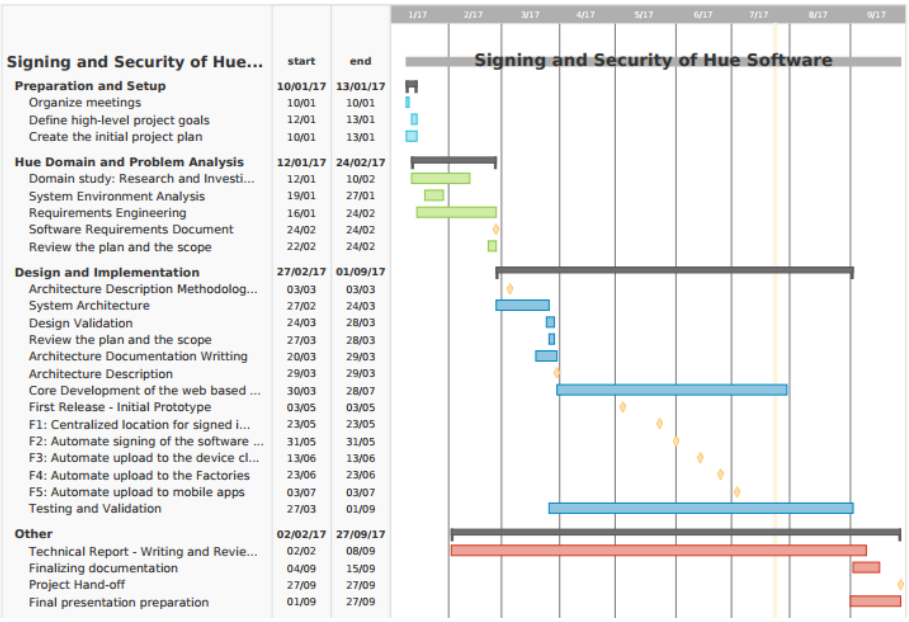

Figure 13.2 Initial project plan

13.3.2. Final

During the project, several adjustments were introduced. New tasks were added and several were modified or deleted. This was a reflection on the understanding of the project and refining of the stakeholders' requirements. Additionally, these adjustments were reflection to the corrections of the initial estimations. This is reasonable, since at the beginning of the project the level of domain knowledge and problem understanding was not sufficient to make highly accurate estimations.

Figure 13.3 Final project planning

13.4 Project execution

The execution of the project followed a structured path based on the project planning. The first three months comprised of meetings with stakeholders and reading domain literature. They also included understanding the high-level requirements of the stakeholders and refining them.

Following the agile approach of iterative software development, a Project Steering Group (PSG) meeting was held on regular basis where the progress of the project was presented by the trainee and the direction of the project was redefined. Additionally, regular weekly meeting was held with the Philips Lighting supervisors, where the project progress was discussed with more deep level of details. During these meetings, the trainee presented the status as well as parts of the final deliverable. The stakeholders gave feedback and to some extent validation of the deliverable. If it was deemed necessary, additional meetings were scheduled by the trainee to obtain extra information regarding specific domain knowledge. For each meeting, notes were taken by the trainee (meeting minutes), which were later put in documents and occasionally sent back to the stakeholders for review and comments.

By following the project plan, stakeholders could transparently observe how the project is progressing, and if the status satisfied their standards and requirements. Since the product includes several features, an ordered approach was followed in their development, one at a time.

As mentioned in section 5.2, several risks were identified throughout the project which caused slight changes in the direction of development. These changes were clearly presented to the stakeholders coupled with a proposed mitigation strategy from the candidate. ■

14. Project Retrospective

This chapter finalizes the document by providing a reflection on the project based on the author's perspective. It also depicts the revision of the design opportunities, defined in the begging of the project.

14.1 Introduction

During these nine months, the project demonstrated several familiar aspects and numerous challenging ones. On the familiar side, there was already sufficient knowledge in software development. Although it was a first time for me to use Java programming language in a professional environment, I had enough experience with similar object-oriented languages so the transition to Java went smooth. Additionally, there was adequate knowledge of software architectures, modeling and designing a software as well some insights on project management. Having a working experience as Information Security Office in a bank, as well as having a Master diploma in the field of information security was of a huge importance for this project, since the security was a core topic for all aspects of the system.

The challenging parts include getting to know the domain and understanding the current software release processes. As with every project, a sufficient level of understanding of the domain is required to be able to translate requirements into a result. Besides the insights from the stakeholders, time of the project was devoted on domain understanding and problem analysis in order to gain sufficient knowledge for the current software development life cycle, especially in the beginning of these nine months. Luckily, the stakeholders from the Philips Lighting were more than helpful and provided needed information and feedback whenever it was requested. This is important because it gives a two-way feedback, for the candidate to understand them, and for them to understand whether the candidate understands the domain.

Additional challenge was to discover all project stakeholders. Many departments contribute to the development of the Hue system. Some of them are even geographically distributed to different countries, such as India and China. Therefore, a sufficient time was spent on the stakeholder's analysis. Again, the project owners and project supervisors team were very helpful in defining the initial list of stakeholders, that were furthermore interviewed during the project.

One of the most challenging technical aspects was the deployment for the final product. As I have already mentioned in the Deployment chapter, there are a lot of Philips internal services that are running on a cloud based environment. Although, I was already introduced to the cloud computing and solutions, I have never had practical experience with them. This was a challenge for me at the beginning, especially in understanding the terminology used and possibilities that these solutions have. Therefore, time of the project was devoted on studying the popular cloud service providers and what they could offer that is relevant to the current project.

Overall, the project was a vast experience and a chance to develop and improve a range of skills, both technical and non-technical. Cooperating with people and managing expectations is crucial and this was repeatedly exercised along the project. Managing and executing the project, while at the same time defining and shaping it, was a great lesson that broadened my skills and knowledge. On top of that, several innovative technologies were used that broadened the technical knowledge and opened new horizons for the future.

14.2 Design opportunities revisited

During the study of the Hue system domain and the problem analysis, the following design opportunities were identified, namely security, usability and reliability. Security is achieved by applying advanced security mechanisms in every aspect of the system. First, system is integrated with existing user membership services, used inside Philips Lighting. Next, the system utilizes advanced encryption methods in several features, such as distributing signed files to different endpoints or storing production software signing keys at the signing machine. Furthermore, the complete system is deployed in a highly secured and isolated environment, utilizing a proven and trusted cloud environment. Finally, a dedicated internal team of security experts was created, which main purpose was to evaluate the system architecture from security perspective and sign it off for further implementation.

Since the system should have replaced complete process, it needed to be made usable and intuitive. This was achieved by designing simple and easy to user interface, which can convey the information and provide feedback to the user as clearly as possible. At the moment of writing this report, the system was still not used in the production environment. Valuable feedback is expected from the users regarding the usability once the system is used more extensively and therefore possible changes of the user interface are expected in the future. Usability test is also good option to evaluate how easy-to-use the system is.

The reliability in the current aspect meant the following. The speed of execution is not critical, but the correctness of the processes is crucial. It was preferred that processes are running sequentially because every action need to be atomic and system should know its state in every moment of time. This was achieved by synchronizing activities in the system and making all process execution activities running sequentially, both at the application and operating system level. ■

Glossary

This section presents the terminologies used throughout this report along with their explanations.

Term	Explanation
PDEng	Professional Doctorate in Engineering
TU/e	Eindhoven University of Technology
SAI	Stan Ackermans Institute
Cloud	General term for anything that involves delivering hosted services over the Internet
API	Application Programming Interface
Server	A computing platform whose purpose is to serve other computing platforms
Web server	Same as server, available over the Internet
Database server	Same as server, used to host a database
XML	eXtensible Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure (HTTPS) is the secure version of HTTP. The 'S' at the end of HTTPS stands for 'Secure'.
REST	Representational State Transfer (architectural style)
RESTful APIs	Interfaces that adhere to the REST style
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
HTML	Hyper Text Markup Language is a markup language for describ-ing web documents (web pages)
MVC	Model View Controller (architectural style)
CSS	Cascading Style Sheets
JDBC	Java database connectivity technology that defines how a client may access a database
SDLC	Software Development Life Cycle
IoT	Internet of Things
IoT-GSI	Global Standards Initiative on Internet of Things
HSDP	HealthSuite Digital Platform
SSL	SSL (Secure Sockets Layer) is the standard security technology for establishing an encrypted link between a web server and a browser.
ASPX	ASPX files are generated by a web server and contain scripts and source codes that help communicate to a browser how a web page should be opened and displayed
JSP	Java Server Pages (JSP) is a technology that helps software developers create dynamically generated web pages based on HTML, XML, or other document types
PHP	PHP (recursive acronym for PHP: Hypertext Preprocessor) is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML
PGP	Pretty Good Privacy (PGP) encryption program provides cryptographic privacy and authentication for data communication.
PKI	Public key infrastructure
HSM	Hardware Security Module
LDAP	Lightweight Directory Access Protocol

Bibliography

[1] The Washington Post, [Online]. Available: https://www.washingtonpost.com/news/innovations/wp/2017/07/21/how-a-fish-tank-helped-hack-a-casino/?utm_term=.5264a938d065. [Accessed 15 September 2017].

[2] Stan Ackermans Institute, "4TU Federation," [Online]. Available: https://www.4tu.nl/sai/en/. [Accessed 12 September 2017].

[3] Eindhoven University of Technology, "PDEng Software Technology," [Online]. Available: https://www.tue.nl/en/university/departments/mathematics-and-computer-science/education/graduate-programs/pdeng-programs/software-technology/. [Accessed 12 September 2017].

[4] Philips Lighting, "Bringing light to the internet of things," [Online]. Available: http://www.lighting.philips.com/main/education/lighting-university/lighting-university-browser/webinar/webinar-semantic-lighting.html. [Accessed 21 April 2017].

[5] Wikipedia, "Internet of Things," [Online]. Available: https://en.wikipedia.org/wiki/Internet_of_things. [Accessed 1 June 2017].

[6] Philips Lighting, "Philips Lighting," [Online]. Available: http://www.lighting.philips.nl/bedrijf/over.html. [Accessed 1 April 2017].

[7] Philips Lighting, "Philips Hue System," [Online]. Available: http://www2.meethue.com/en-us/. [Accessed 1 April 2017].

[8] Wikipedia, "Software signing," [Online]. Available: https://en.wikipedia.org/wiki/Code_signing. [Accessed 15 May 2017].

[9] C. Schwaderer, "The distribution of things: IoT, M2M, and software distribution," [Online]. Available: http://embedded-computing.com/articles/the-iot-m2m-software-distribution/. [Accessed 10 March 2017].

[10] Wind River Systems, Inc, "Managing the IoT Lifecycle from design through End-of-Life".

[11] SSL Shopper, "What is code signing," [Online]. Available: https://www.sslshopper.com/what-is-code-signing.html. [Accessed 12 September 2017].

[12] S. Brander, "Key words for use in RFCs to Indicate Requirement Levels," Harvard University, Network Working Group, 1997.

[13] I. Sommerville, Software Engineering (9th edition), Addison Wesley, 2011.

[14] L. Bass, P. Clements and R. Kazman, Software Architecture in Practice, Addison Wesley, 2003.

[15] Sheffield Hallam University, "Architecture of Web-based systems," [Online]. Available: http://teaching.shu.ac.uk/aces/rh1/de/web_based_systems_architectures_1_tutorial.htm. [Accessed 12 September 2017].

[16] M. Richards, "Software Architecture Patterns," [Online]. Available: https://www.safaribooksonline.com/library/view/software-architecture-patterns/9781491971437/ch01.html. [Accessed 12 September 2017].

[17] T.Masinov, "Web-based visualization of guidelines and drug use in epilepsy : a project contributing to smart drug," Technische Universiteit Eindhoven (TU/e). Stan Ackermans Instituut, Eindhoven, 2015.

[18] M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee and R. Stafford, Patterns of Enterprise Application Architecture, Addison Wesley, 2002.

[19] Microsoft, Microsoft Application Architecture Guide, 2nd Edition, Microsoft Press, 2009.

[20] Wikipedia, "Representational State Transfer (REST)," [Online]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer. [Accessed 12 September 2017].

[21] Wikipedia, "Hypertext Transfer Protocol," [Online]. Available: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol. [Accessed 12 September 2017].

[22] Wikipedia, "Model View Controller (MVC)," [Online]. Available: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller. [Accessed 12 September 2017].

[23] IEEE, "IEEE Recommended Practice for Architectural Description for Software-Intensive Systems," 2000. [Online]. Available: https://standards.ieee.org/findstds/standard/1471-2000.html. [Accessed 12 September 2017].

[24] P. Kruchten, "Architectural Blueprints—The “4+1” View Model of Software Architecture," *IEEE Software 12*, pp. 42-55, November 1995.

[25] U.S Department of Defence, "The DoDAF Architecture Framework," 2010.

[26] K. Raymond, Reference Model of Open Distributed Processing (RM-ODP), Boston: Springer, 1995.

[27] G. Muller, "CAFCR: A Multi-view Method for Embedded Systems Architecting. Balancing Genericity and Specificity," in *Systems Architecting: A Business Perspective*, CRC Press, 2011.

[28] J. A. Zachman, "The Zachman Framework," 2008.

[29] N. Rozanski and E. Woods, "The deployment viewpoint," [Online]. Available: https://www.viewpoints-and-perspectives.info/home/viewpoints/deployment/. [Accessed 12 September 2017].

[30] Tutorials Point, "Spring - MVC Framework," [Online]. Available: https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm. [Accessed 12 September 2017].

[31] Amazon, "Amazon EC2," [Online]. Available: https://aws.amazon.com/ec2/. [Accessed 12 September 2017].

[32] Amazon, "Amazon Relational Database Service," [Online]. Available: https://aws.amazon.com/rds. [Accessed 12 September 2017].

[33] Amazon, "Amazon Elastic File System," [Online]. Available: https://aws.amazon.com/efs/. [Accessed 12 September 2017].

[34] Amazon, "Amazon Elastic Block Store," [Online]. Available: https://aws.amazon.com/ebs/. [Accessed 12 September 2017].

[35] Amazon, "Amazon Virtual Private Cloud (VPC)," [Online]. Available: https://aws.amazon.com/vpc/. [Accessed 12 September 2017].

[36] Google, "Google Cloud Compute Engine," [Online]. Available: https://cloud.google.com/compute/. [Accessed 12 September 2017].

[37] Google, "Google Cloud SQL," [Online]. Available: https://cloud.google.com/sql/. [Accessed 12 September 2017].

[38] Google, "Google Persistent Disk," [Online]. Available: https://cloud.google.com/persistent-disk/. [Accessed 12 September 2017].

[39] Google, "Google Virtual Private Cloud (VPN)," [Online]. Available: https://cloud.google.com/vpc/. [Accessed 12 September 2017].

About the Authors

Igor Anastasov received his Bachelor of Electrical Engineering and Information Technologies from the Faculty of Electrical Engineering and Information Technology (Skopje, Macedonia), in 2011, specializing in Informatics and Computer Engineering. He received his MSc degree in Software Engineering of the Faculty of Computer Science and Engineering (Skopje, Macedonia) in 2014.

In his Master Thesis called "SIEM implementation for global and distributed environments" he proposes a new model and architecture for SIEM system that is using multiple hierarchically connected SIEM systems. Part of the investigations in this thesis resulted in a research paper published at the International Conference on Computer Information Systems (ICCIS) WCCAIS 2014 in Tunisia.

After his graduation in 2011 he worked in several IT companies in Macedonia, developing software applications mainly in .NET and SQL Server RDBMS. Between 2012 and 2014 he worked in ProCredit bank, Macedonia as Information Security Officer. His responsibilities include development and execution of a clear and effective strategy concerning Information Security throughout the organization. From September 2015 until September 2017, he worked at the Eindhoven University of Technology, as PDEng trainee in the Software Technology program from the 4TU.Stan Ackermans Institute.