UNIVERSITY OF TARTU
Institute of Computer Science
Software engineering Curriculum

Lasha Tsintsabadze

A Prototype to Analyze Role- and Attribute-Based Access Control Models

Master's Thesis (30 ECTS)

Supervisor(s):
Raimundas Matulevicius

Tartu 2017

A Prototype to Analyze Role- and Attribute-Based Access Control Models

Abstract:

The goal of this thesis is to create an access control comparison prototype, where people will do experiments with security models and analyse reports based on their actions. The thesis is split into two parts: theoretical and practical. In the theoretical part, we studied how security models like, Role-Based Access and Attribute-Based Access work, defined the meta models and explained the security flows. After that, we did the theoretical comparison between these models and defined the comparison criteria, which later was used in the prototype. Meanwhile, in practical part, we put main points of the theoretical part and defined requirements and use cases in order to give maximum experience to the users about what is going underneath the application during the interaction through graphical user interface.

Keywords:

RBAC, ABAC, Security model comparison, Security model comparison prototype, Access control model

CERCS: T120 Systems engineering, computer technology.

Prototüüp analüüsimaks rolli- ja vahendipõhiseid juurdepääsumudeleid

Lühikokkuvõte:

Käesoleva lõputöö eesmärgiks on luua juurdepääsu kontrolli võrdlemise platvorm või tööriist, mille abil kasutajad saavad eksperimenteerida ning luua turvaanalüüse ja -mudeleid. Lõputöö jaguneb kahte ossa: teoreetiline ja praktiline. Teoreetilises osas uuritakse, kuidas turvalisusmudelid, nagu näiteks kasutajapõhine juurdepääs ja atribuudipõhine juurdepääs töötavad, defineeritakse metamudeleid ja selgitatakse turvalisuse voogu. Seejärel võrreldakse kahte mudelit, fikseerides võrdluskriteeriumid, mida hiljem kasutatakse platvormil. Praktilises osas kasutatakse teoreetilise osa põhipunkte ning defineeritakse vajadused ja kasutuslahendid, et anda kasutajatele maksimaalne arusaam rakenduse sees toimuvast kasutajaliidesega suheldes.

Võtmesõnad:

RBAS, ABAS, turvamudelite võrdlus, turvamudelite võrdluse prototüüp, juurdepääsu kontrolli mudel

CERCS: T120 Süsteemitehnoloogia, arvutitehnoloogia tehnikateadused.

Acknowledgment

First of all, I would like to express my deep sense of gratitude to my supervisor Raimundas Matulevicius, for his patience and guidance during the research. I must appreciate University of Tartu, the Faculty of Software engineering for giving me a chance to get the valuable education for my successful career path. In the end, I would like to express my profound gratitude to my family and friends for their tremendous support and continuous encouragement throughout my study. This accomplishment would not have been done without them. Thank you

Table of Contents

1 Introduction 8
2 Background Study 10
2.1 Role Based Access Control 10
2.1.1 Major description and principals 10
2.1.2 Meta-Model 11
2.2 Attribute-based Access Control Model 12
2.2.1 Major description and principals 12
2.2.2 Meta-Model 14
2.3 Summary 17
3 Comparison of Access Models 18
3.1 Comparison criteria definition 18
3.2 Comparison and result presentation 20
3.3 Summary 23
4 Prototype 24
4.1 Requirements specification 24
4.1.1 Product perspective 24
4.1.2 Scenario description 24
4.1.3 Scenario functions 25
4.1.4 Expectations 36
4.1.5 System requirements 37
4.1.6 Authentication 37
4.1.7 Scenario requirements 38
4.1.8 Prototype requirements 40
4.2 Implementation 41
4.2.1 Scenario and Role-Based Access Control (RBAC) implementation 41
4.2.2 Attribute-Based Access Control(ABAC) implementation 48
4.3 User manual 52
4.4 Summary 58
5 Conclusion 59
5.1 Future work 59
References 60
Appendix 62
I. List of Acronyms 62
II. License 63

List of Figures

Figure 1 SecureUML Metamodel (adapted from [4])...12
Figure 2 XACML meta-model (Adopted from [5] [6])..15
Figure 3 XACML architecture (Adopted from [5] [6])..16
Figure 4 Main types of Extended RBAC model (Adapted from [22])..............................20
Figure 5 Access control comparison system structure..24
Figure 6 Scenario class diagram...25
Figure 7 Introduction of the prototype use case...26
Figure 8 Analytical comparison of security models use case..26
Figure 9 Apply security access model use case..26
Figure 10 List companies in the system use case...27
Figure 11 Add new company in the system use case...27
Figure 12 Delete company from the system use case...28
Figure 13 View company detail from company list page use case.....................................28
Figure 14 Update specific company use case..29
Figure 15 List all candidates in the system use case..29
Figure 16 Add new candidate in the system use case..30
Figure 17 Delete candidate from the system use case..30
Figure 18 View candidate detail from candidate list page use case.....................................31
Figure 19 Update specific candidate use case...31
Figure 20 List jobs in the system use case..32
Figure 21 Delete job from the system use case...32
Figure 22 View job detail from job list page use case...33
Figure 23 Change job status use case..33
Figure 24 Add new job in the system use case..34
Figure 25 Change authority role use case..34
Figure 26 Authorize into the system use case..35
Figure 27 Registration of new user use case..35
Figure 28 Take a quiz use case...36
Figure 29 Quiz result management use case..36
Figure 30 Prototype layout..42
Figure 31 Spring security configuration file...43
Figure 32 Company resource controller with RBAC access model.......................................44
Figure 33 Security actions types in HR management system...44
Figure 34 An example of SecureUML model of flat RBAC...46
Figure 35 An example of SecureUML model of hierarchical RBAC......................................47
Figure 36 An example of SecureUML for constrained RBAC..48
Figure 37 Implementation of PermissionEvaluator..49
Figure 38 Implementation of ContextAwarePolicyEnforcement...49
Figure 39 Implementation of PolicyEnforcement component..50
Figure 40 Log in and registration forms...52
Figure 41 Dropdown menu for user name and logout..52
Figure 42 Sidebar menu and introduction page..53
Figure 43 Access control comparison result page..53
Figure 44 Change user role dropdown menu..54
Figure 45 Apply access model dropdown menu...54
Figure 46 Company detail view, list view and create view...55
Figure 47 Job detail view; list view; create view;...56
Figure 48 Candidate detail view, list view and create view..57

Figure 49 Quiz result management system. 58

List of Tables

Table 1 A comparison of Role-centric access model and Attribute-centric access model (adapted from [22]) 21
Table 2 System expectations 36
Table 3 Software expectations. 37
Table 4 System requirements for prototype. 37
Table 5 Authentication function requirements for prototype. 37
Table 6 Scenario functional requirements. 38
Table 7 Prototype functional requirements. 40
Table 8 Policy repository 51

1 Introduction

A conspicuous part of security related topic in computer science is access control. Main idea of access control is protecting sensitive data. It determines whether the user has permission to access information or not. For a long time, AC world was dominated by discretionary and mandatory access control models. In late 1990s research community realized that MAC and DAC cannot cope with a fast-growing IT industry requirements. So, in 1992 RBAC model was introduced, which dominated AC word for almost a decade. But now it faces a same problem again, it cannot cope with new challenges. The base problem of MAC, DAC and RBAC is that, they are created to control access in static environment, where users, resources and permission must be predefined and nothing changes for a set of periods, but nowadays modern technologies showed that traditional access control should give flexible, complex and anonymous access control in dynamical environment.

New challenges raised demand for more flexible access models and we have now models like, Attribute-based, Usage-based, Risk-based, etc. The Motivation of this thesis is to help people elucidate which model fits their needs. Unfortunately, most of the new models are conceptual models, which means that there it is not production ready. So, we limited the scope of this project to only two models: Role-Based and Attribute-Based. The reason behind choosing Attribute-Based model is National Institute of Standards and Technology (NIST), which already published official paper about the definition and consideration of ABAC.

Nowadays people can find a lot of papers and materials about analytical comparison of RBAC and ABAC, which will give a good theoretical idea about which one is better. But there is nothing that will give practical experience to them before using it in their projects. The goal of our project is to create a prototype where people will see RBAC and ABAC models in practice. It is intended to be an educative program for people who doesn't know much about security models. The aim is to implement a prototype in that way that users will be able to see what is in the background of the application during their interaction with graphical user interface. Users should see step by step flow of how RBAC or ABAC is securing the resource when server receives a request.

In order to implement Access Control Comparison Platform, we need to make several contributions to the thesis:

- Understand how access models work specifically Role-Based access and Attribute-based access
- Produce empirical comparison and define comparison criteria.
- Based on comparison criteria, create requirements and use cases of platform.
- Platform implementation based on requirements.
- Creation of user manual.

Overall, the structure of this thesis is aligned like this: Chapter 2 presents an overview of access models such as RBAC and ABAC. We provide a general description and explain meta-model for each access model. Furthermore, we introduce modelling languages like secureUML and XACML, that can be used to express each access model. Chapter 3 presents an analytical comparison of ABAC and RBAC, where we concentrate on defining comparison criteria and also present results using table revealing the conceptual similarities and differences between these models. Chapter 4 defined the requirements for access model comparison platform. This chapter is split in three parts. Firstly, we define scenario which

our prototype will be based on, then we define requirement specifications and use cases. Secondly, we describe the implementation of the prototype. We present the technology stack of the application, the architecture and step by step illustration of how each access model is implemented. In the end, we will present the user manual of porotype, which explains usage of each features of platform. We provided step by step instructions and visualized it on platform GUI. Finally, Chapter 5 concludes this thesis, which includes the limitations of the prototype and the new recommendations for future work.

2 Background Study

In this chapter, we will provide an overview of Role-Based and Attribute-based Access Control models and explain how they work. In Section 2.1 we concentrate on *Role-Based Access Control model*, where we describe the main concepts and meta-model of it. Also, we will present *secureUML*, the language for modelling RBAC rules in unified modelling language(UML). We will concentrate on *secureUML's* major principal description and on the structure of the language. We will explain the notation and the met model in the next section. On the other hand, in section 2.1 we will give an overview of *Attribute-based Access Control* model (*ABAC*). Firstly, we will provide the main concepts of the *ABAC,* then we will present *XACML* modelling language, which is used to express ABAC Policies.

2.1 Role Based Access Control

2.1.1 Major description and principals

Role-based access model is an access control method to ensure that only authorized user can have access to the data in the system. Unlike other access models, in RBAC users are assigned to Roles, where roles already have granted permissions. Users can be assigned to any number of roles based on their job requirements. For example, let's take a user who should have analyst and developer roles, each role will have permissions that are needed to access specific objects in the system [1].

The concept of the role-based access model is clear and straightforward. One of the core advantages of RBAC is significantly less responsibility of system administrator. In RBAC is no static template for creating security policies, because all organizations have different requirements. Let's take an example to have a clearer understanding how RBAC works. Imagine the situation when a user changes the job inside the organization in the non-role-based environment, the system administrator should update user permissions manually for different object levels. However, in role-based environment administrator just should change the role of the user which already have granted set of permissions.

RBAC policy is embodied in various components such as role-permission relationships, user-role relationships, and role-role relationships. These components determine whether the user has access to a resource or not in the system. RBAC can modify the policy to meet the requirements of an organization which is the most significant benefit. RBAC implements three most important security principals: least privilege, separation of duties and data abstraction. Least Privilege is supported because RBAC can be configured, so only those permissions required for tasks conducted by members of the role are assigned to the role. Separation of duties is achieved by ensuring that mutually exclusive roles must be invoked to complete a sensitive task. Data abstraction is supported using abstract permissions such as credit and debit for an account.

The family of RBAC consists of four models: flat, hierarchical, Constrained and Symmetric RBAC. Core RBAC [1][2] is the base model, minimal requirement for any system which is supporting RBAC. Core RBAC's elements are users, roles, objects, operations, and permissions. The main process of RBAC is that Permissions are assigned to Roles and Roles are assigned to users. Roles may have one or many Permissions and Roles. It also includes Sessions, which is the mapping between authorized users and roles assigned to them. Each session is linked to the specific user, and each session is related to roles. *Session_roles* and

session_users functions can be triggered to return linked roles and user to the session. When the user gets sessions, he also gets access to his permissions.

Hierarchical RBAC, constrained RBAC and symmetric RBAC is extensions of Core RBAC. Apart from Core model, the Hierarchical model has role hierarchies' definition. Role hierarchies are used to mirror the hierarchical line of authorities and responsibilities in an organization. It is defined regarding permissions, where senior roles include permissions of junior roles also. For example, the role r1 is inherited by r2 if r1 permissions are included in r2 permissions. There are two types of role hierarchies: general and limited role hierarchies. The main difference between them is that general role hierarchies have a multiple inheritances support, which makes possible to inherit user membership from more than two role sources. On the other hand, limited role hierarchy is restricted to a single immediate inheritance [1][2].

The constrained RBAC [1][2] Apart from role hierarchy model places a restrictive rule on the potential inheritance of permission from opposing roles. Hence, it can be used to accomplish for appropriate separation of duties by limiting the power of individual user or session. For example, login account creation and account creation authorization should be allowed for the same user, it should be separated. Constrained RBAC allows static and dynamic separation of duty.

The Symmetric RBAC [1][2] includes the requirements of Constrained RBAC. It implements a permission-role review requirement the same user-role requirement we have in core RBAC. It gives identification of the permissions to existing roles and vice versa. For example, the administrator removes all the user's permissions by identifying permission of leaving users and then reassigns to other users with a same or different set of permissions.

2.1.2 Meta-Model

SecureUml overview

Integrating security engineering is very important in software development process. It allows developers to integrate security policies into the system at a high level of abstraction and decrease chances of violating those policies and prevents errors in the future development of access control models.

SecureUML is an extension of the standard UML language. It is used to describe the vocabulary to annotate access models in UML environment. *SecureUML* is oriented on RBAC model. It defines all components of RBAC such as role, role permission, and user-role assignment. Moreover, it also provides support for authorization constraints definition. Because of its extensibility, *secureUML* is very easy to use language for business analysis as well as a designing security model [3].

The main purpose for us to use this language is to demonstrate RBAC capabilities, based on our scenario, described in Figure 1. *SecureUML* gives us the opportunity to define different models with different levels of abstraction, using the same syntax and compatible semantics. Usual workflow of *secureUML* model creation is:

- User identification
- Role identification
- Role hierarchy identification
- User and Role mapping

- Resource identification
- Action identification
- Authorization constraint identification

SecureUml meta-model

The meta-model defines the abstract syntax of the language, i.e. the structure of a model representation that is independent of particular notation. As shown in Figure 2 *SecureUML* meta-model introduces the new types like user, permission, role as well as relationships between them. Instead of making separate type for protected resources, secureUML allows every *ModelElement* to use the role of it. *SecureUML* also introduces *ResourceSet*, which is set of modelElement defining permissions and authorization constraints. Permission connects role to *ModelElement* or a *ResourceSet*, which is defined by *ActionType*. Every *ActionType* contains operations on a specific resource in the system. On another hand, *ActionTypes* available for a particular meta-model type is defined by a *ResourceType*. An authorization constraint represents access control policy in the model. It checks every precondition before calling some resource in the system. For example, let's assume that we want to have access condition for operation *editBlog* () on class Blog to make sure that only user with right role will have access to it. To achieve this goal and authorization constraint will check if the user is the author/owner of the blog. Authorization constraint is attached to the protected resource(*ModelElement*) via permission [3].

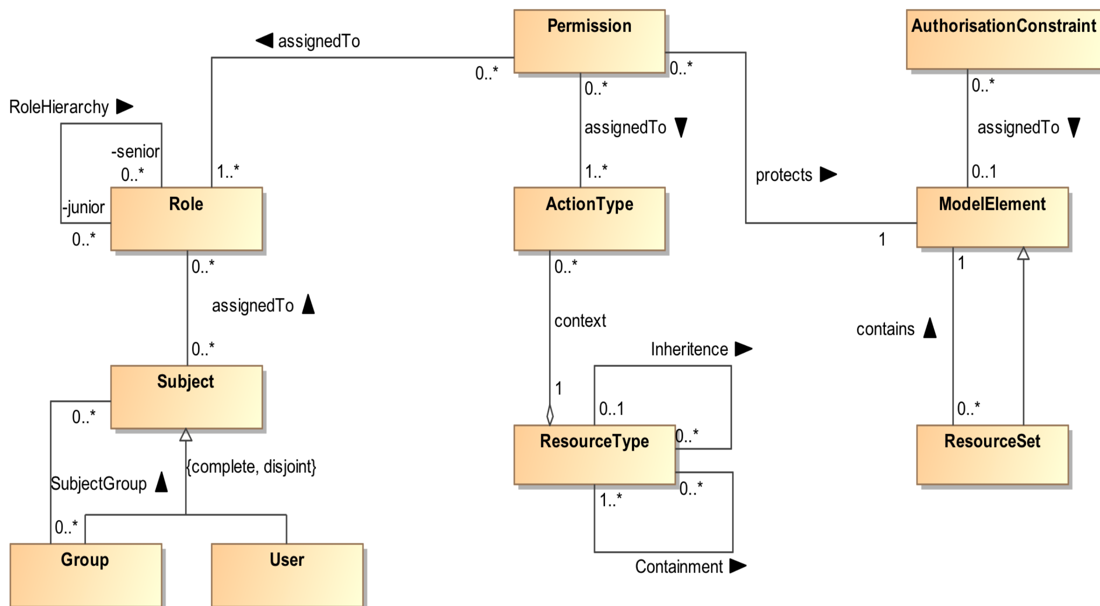

Figure 1 SecureUML Metamodel (adapted from [4]).

2.2 Attribute-based Access Control Model

2.2.1 Major description and principals

Unlike its rivals, ABAC is a distinct access model, because it authorizes access to the resources by evaluating policies against the attributes of entities and the environment conditions suitable to the request. It separates authorization and authentication by asking what are you and not who are you. In other words, ABAC can define permissions not only based on

the role but any relevant characteristics(attributes) of the entity. The main elements of ABAC model are Subject, Object, Operation, Policy, Environment, Rule and Attributes. Attributes itself are split into three types: Subject Attributes, Object Attributes and Environment Attributes [7].

The *Subject* is an individual (e.g. human or NPE[1]) who sends the request for acting on a specific resource. Each subject has an identity, which is defined by attributes. For example, attributes hold subject's name, age, job title, role and so on. In ABAC environment *Subject* is a definition of *user.* The *Object* is a resource secured by the ABAC, such as files, tables, programs, devices and so on. Basically, everything that can be managed by the subject performing some actions. Like *Subjects, Objects* also have attributes. For example, a publication in newspaper portal can have attributes like owner name, creation date, access permission and so on. The *Environment* is a description of the context in which access occurs. *Environment attributes* describe operation and technical characteristics like location of the access request, current date and time, network security level and so on. The Environment is not related to the subject or object; it is applied to the whole policy. An *Operation* is the action *subject* want to perform on the *resource(object)* like read, write, delete, execute, update and so on. A *Policy* is a representation of Rules that decide to permit or reject an incoming request for a resource based on values of the *subject, object and environment attribute* values. A Rule is a Boolean function which decides if subject can access object environment [7] [8].

In ABAC control, the *object* is protected using *Access Control Mechanism(ACM).* When the request comes, ACM will collect *attributes,* evaluate the logic of *policy* and enforce the decision to reject or permit access to the object. ACM must be able to manage process of decision enforcement, also must be able to determine which policy is applicable for the request, which attributes to get and from where to get it and so on. For this ACM uses several functional points, like *the Policy Enforcement Point (PEP), the Policy Decision Point (PDP), the Policy Information Point (PIP), and the Policy Administration Point (PAP)* [14].

The Policy Enforcement Point (PEP) functional point has two main duties: to request authorization decision and to enforce the decision. In other words, it is a point which stands between resource and request. PEP cannot be bypassed to get access to the resource. *The Policy Decision Point(PDP)* function point tasks applicable policy, evaluates it and calculates authorization decision, which is either *Permit* or *Deny.* In other words, PDP is a ABAC control engine. PEP enforces the decision from PDP. PDP and PEP aren't necessary to be centralized, they can be distributed throughout the network. PDP component calculates decision using *the Policy Information Point (PIP),* which provides PDP with necessary data from attributes to calculate decision. Before enforcing policies, it should be tested to make sure that they satisfy the requirements. This is handled by the *Policy Administration Point (PAP).* PAP manages policy creation, testing, debugging and storing it to policy repository [2].

[1] An entity with a digital identity that acts in cyberspace, but is not a human actor. This can include organizations, hardware devices and software applications.

2.2.2 Meta-Model

XACML overview

In this section, we will discuss various elements found in the eXtensible Access Control Mark-up Language (XACML), which is presented in Figure 2. XACML is OASIS[2] standard and uses XML[3] mark-up language as a syntax. It is used to express ABAC access model concepts.

XACML model consists from three main elements *PolicySet, Policy* and *Rule. Rule* defines the desired effect returned to Requester, either *"Deny", "Permit" and "Not Applicable" "Deny"* means that request was evaluated by all applicable Policies and request is not authorized to provide some actions on the resource. *"Permit"* means that request was evaluated by applicable policies and request is authorized to perform some actions on the resource. And *"Not Applicable"* means that no applicable policy was found for given request and it cannot be evaluated [5] [6].

Target element is not only a part of all the core components of XACML. It is a mapping between *Subject*, *Object* and *Action* to the *Policy*, *PolicySet* or *Rule.* It holds the index of the Policies, so when XACML engine receives request, it will pull all the policies from "repository" as an input and use *Target* element to find which *Policy*, *PolicySet* or *Rule* applies to the request. Then XACML will compare request attributes and *Target attributes* and in case of match applicable *Policy, PolicySet or Rule* will be evaluated, else XACML engine will return "*NotApplicable*" decision to the request. *Target* element itself contains: *Subjects, Action, Environment and Resources* elements. *Subjects* is a set of *Subject* elements, which represents the identification of the entity, who is willing to perform actions on the resource. *Resource* element represents the actual resource which *subject(user)* is trying to access. *Action* element defines the action set, like read, write, execute etc, subject can perform on the resource after getting permission. *Environment* element define system attributes, which lets us define system property check for requester. For example, assume that we will apply policies only based on domain. We will specify domain name in *Environment* element and use it to match with requester domain name [5] [6].

The root elements of XACML language is *Policy* and *PolicySet. PolicySet* is an element which may include other *Policy* or *PolicySet* elements, as well as links to other remote Policy containers. *Policy* element itself represents a single access control policy which is expressed using *Rule, Target and Obligation* elements. *Rule* element is used for implementing the authorization logic. The structure of the Rule is split in three main parts: *Condition, Target and Effect.* As we already mention *Target* is used for indexing *Rule. Condition* element is the place where the actual authorization logic is defined and always returns Boolean result. Based on the outcome of the *Condition Effect* is evaluated. *Effect* is an attribute of the *Rule,* which specifies the outcome of it. Usually *Rule* has two types of *Effect, "Deny"* and *"Permit".* If the *Condition* evaluates to *true,* the *Effect* of the *Rule* will be *"Permit",* else *"Deny"* [5] [6].

---

[2] A non-profit, international consortium that creates interoperable industry specifications based on public standards such as XML and SGML

[3] A markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

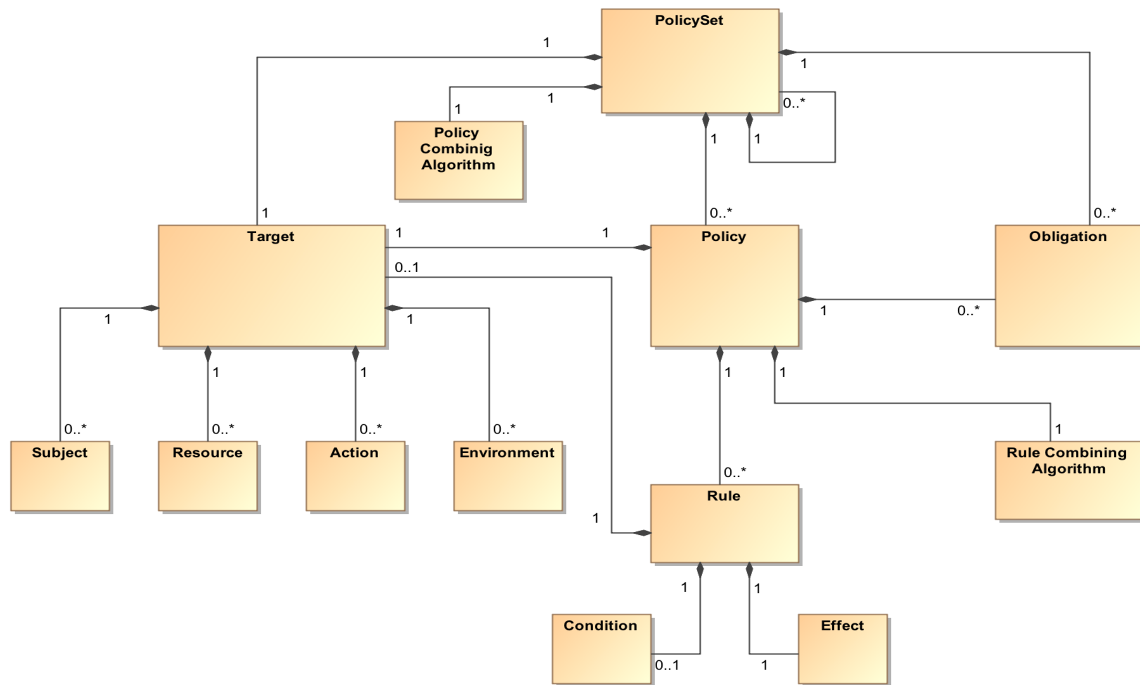

Figure 2 XACML meta-model (Adopted from [5] [6])

PolicySet may contain multiple *Policies* and *Policy may include multiple Rule* elements. Those *Rules* and *Policies* can have different access control decision evaluation. In this case, XACML needs some way to monitor what decision each *Rule* and *Policy* makes. This is done by using *Combining algorithms*. Those algorithms help XACML to combine multiple decision in the single decision. XACML implements two types of combining algorithm: *Policy combining algorithm* used by *PolicySet* and the *Rule-combining algorithm* used by Policy component. For example, let us take *Deny override* algorithm, which is one of seven built in algorithms in XACML. It basically says that if any evaluation will return *"Deny",* then the final decision also will be *"Deny"* [5] [6].

XACML also introduces *Obligation* concept which is a part of *PolicySet* and *Policy* elements and defines the certain actions that must be carries out before access it permitted. It is an optional element and may not be implemented in *Policy*. *Attributes* in XACML are named values, which characterize *Subject*, *Resource*, *Action* or *Environment* in which request is sent from. For instance, attribute values may include a user's name, user's security consent, the requested file and so on. The request sent from PEP to PDP is formed using attributes, where they will be compared to the policy attributes to make access decisions. For retrieving attribute values out of request XACML implements two mechanisms: *AttributeDesignator* and *AttributeSelector*. *AttributeDesignators* allows Policy to look for the attribute using the name, type or issuer. And *AttributeSelector* allows Policy to get attributes using XPath[4] query [5] [6].

[4] A query language for selecting nodes from an XML document.

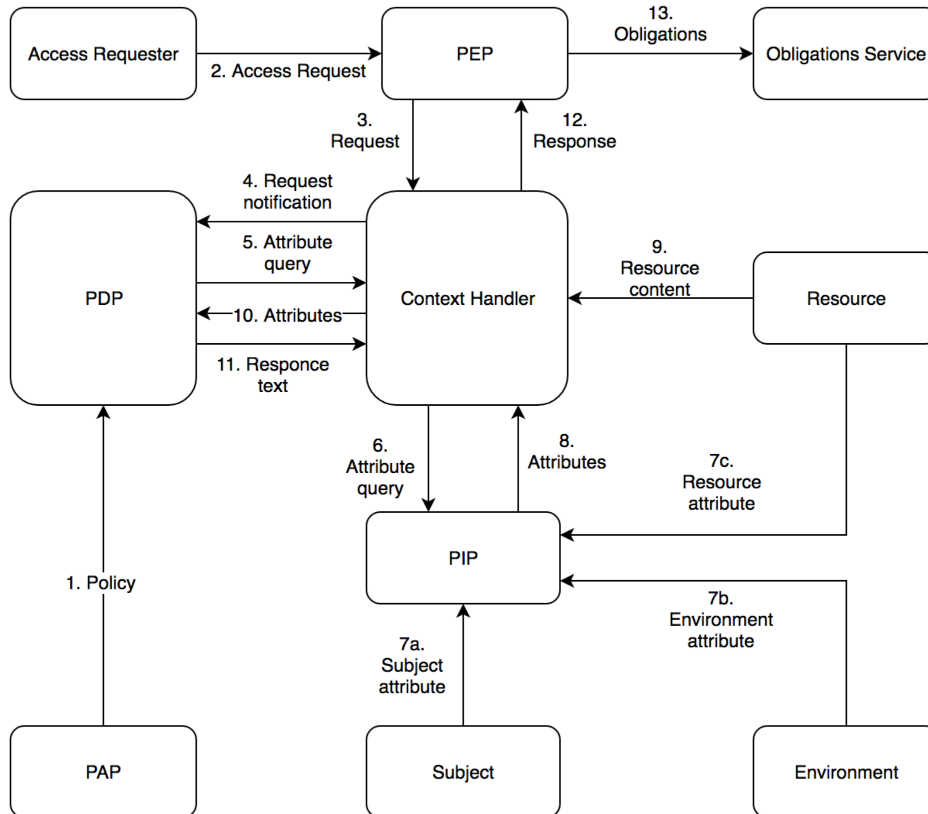

Figure 3 XACML architecture (Adopted from [5] [6])

Figure 3 provides the overview and the flow of the XACML language. As we see XACML architecture consists from several key components:

- *PAP (Policy Administration Point)* - component is responsible for *Policy* and *PolicySet* creation.
- *PDP (Policy Decision Point)* - component is responsible for execution on applicable policy and rendering policy decision.
- *PEP (Policy Enforcement Point)* - component is responsible to block access request, forward it to PDP and act based on the decision received from PDP.
- *PIP (Policy Information Point)* - component is responsible for retrieving information from attribute values, like *subject, resource* and *environment.*
- *CH (Context Handler)* - component is responsible for converting decision request to XACML data format.
- *OS (Obligation Service)* - component is responsible for handling obligations

Overall the data flow in XACML looks like this - First *PAP* loads all available *policies* and *policySets* in *PDP*. When system receives access request, *PEP* will intercept it and forward authorization request to *CH*. *CH* will change authorization request format to XACML supported format and will send request notification to *PDP*. After receiving request notification *PDP* will ask *CH* to send back all available attribute data. *CH* will request *PIP* to collect all data from attribute values. After receiving request *PIP* will retrieve data from *subject attributes*, *resource attributes and environment attributes* (7c) and send it to *CH* (8). *CH* may also get *resource content* (9), but this is optional. After collecting attribute values and resource content, *CH* will send it back to *PDP* (10). After receiving necessary data *PDP* will evaluate policies and send back to *CH* (11). *CH* will decode response from XACML format and forward it to *PEP*. Next *PEP* will process response data and grant access if the decision will

be *Permit* and deny access if decision will be *Deny.* In case policy has some *obligations,* before making deciding access decision *PEP* will send response to OS (13) to check if obligations are fulfilled. This step is also optional [6].

2.3 Summary

In this chapter, we introduced ABAC and RBAC architectures. We detailed the major concepts and how they interact with each other, which showed us how these models make access decisions. We also introduced modelling languages such as secureUML and XACML. We discussed how these languages are compatible with these models. In the next chapter, we will talk about how these models differ from each other. We will define the requirements and challenges modern access models face and then examine how these models are dealing with them.

3 Comparison of Access Models

This chapter gives a comprehensive analysis of two approaches in access control world: role-centric approach represented by RBAC and attribute-centric approach represented by ABAC. Our aim is to define the difference between RBAC and ABAC models by identifying their limitations.

3.1 Comparison criteria definition

In order to correctly identify limitations of models we need to dig into the history a little bit when the AC world was dominated by DAC and MAC models. It will help us to define inherent characteristics of RBAC, which during time become the limitations against future needs of AC. Next, we need to determine what kind of requirements the modern AC must meet. To sum up, this chapter is organized as follows. In section 3 we will write an overview of old AC-s. In section 3.1 we will identify the criteria of modern AC and finally, in section 3.2 we will illustrate how RBAC and ABAC models meet the requirements defined in 3.1 section.

Around three decades AC world was dominated by discretionary and mandatory access control models. Discretionary access control, known as DAC is a security model where object's owner defines an Access Control List (ACL) for specific objects like a database table, file, etc. It contains entries(ACE) which include user identities and privileges who has access to the resource. In other words, the owner decides objects privileges. A common example of DAC is windows file system. On the other hand, Mandatory Access Control (MAC) is stricter, where only administrators can manage access to programs and files. No other user can override the policies. This model was used in military systems.

In late 1990s research community realized that MAC and DAC cannot cope with a fast-growing IT industry requirements. So, in 1992 RBAC model was introduced, described in section 2.1, which is the most dominated and used access control model nowadays. However, during these years, the practical use of RBAC model showed that it has some problem. RBAC is a part of traditional access controls, which was created to control access in static environment, where users, resources and permission must be predefined and nothing changes for a set of periods, but nowadays modern technologies showed that traditional access control should give flexible, complex and anonymous access control in dynamical environment.

For access model comparison, first we should define the comparison criteria. Nowadays modern access controls should satisfy several requirements. They should control: *static access, fine-grained access, context insensitive access, content independent access, on-going access, user prior identification access, multi-factored access and inflexible access*. Let's break down these requirements and examine how RBAC and ABAC can secure them [22].

Fine-grained access refers to a state where details and precision have a great attention. Unfortunately, RBAC is more coarse-grained access than fine-grained, where a state has a lack of attention to details and provides rough estimation only, which sometimes is leading to the accidental situations where user gets unauthorized access. For example, in policy user may have access to one cell in the table, but his permissions can permit him to view the whole table. Fine-grained access control is crucial for AC flexibility in sense of assigning different rights to the users [22].

The appearance of cloud computing created the need for *context insensitive access control.* Context insensitive access can be described as a state where an event or statement is composed of a set of conditions to give a better understanding. The context information may be different for making access decision, everything is based on usage scenario. Context information consist multiple characteristics like location, network type, device type, time, temperature and so on. RBAC doesn't support context insensitive access, because it's access decisions are based only on user role. So, it is cannot cope the situation where the context of identity may be based on more than static roles [22] [6].

The need for *ongoing access control* emerged from e-commerce application usage. Ongoing access control defines the state where access is controlled continually during the active user session. RBAC, which is the part of traditional AC-s, only support one-time access control where access control decision is made once on request time and the granted access will last until the user session ends. The permission will only re-evaluate after the session termination. During session, no ongoing permission check is applied [22] [6].

Content independent access is user-centric approach and can be described as a set of characteristics that are a part of something like a user. Centric means that something has a central position, so user centric access control will be a control where a user has the greatest importance. Being a role-centric approach, RBAC is not very good at controlling access to the objects whose authorization is defined by the content. The good example of content independent access is health care application, where the doctor can only access the data of his patients who were treated in last week. To create this kind of policy administrator will need patient record contents, specifically the date of treatment to the current doctor [22] [6].

Sometimes it is crucial to ensure access control without registering or user provisioning processes. This type of access called *user prior identification access.* The example of *user prior identification access* is a hotel where users are offered with free internet. RBAC is identity-based access, meaning that it cannot provide access control without identifying the user. Next requirement for AC is *multi-factored access control.* Multi-factored access is more accurate and reliable then single factored access. Unfortunately, traditional access controls only support single factored access control. RBAC decision factor relies only on user roles [22] [6].

Inflexible access control means that AC should be able to handle dynamically changed circumstances. For a big company with a good management, proper hierarchy structure and user roles along with permissions, RBAC is very useful model. However, for small companies where people work in agile process, rotating job responsibilities dynamically RBAC is not the best model to choose, because it wasn't created to handle flexible, dynamically changes environments. For example, let’s extend hotel scenario where clients are provided with free Wi-Fi. On the other hand, the employees of the restaurant have a role and corresponding permission to do things. To provide AC for this scenario, the administration should deploy to kinds of access control model, one for hotel employers and another for clients. This approach is not ideal, modern access control should be flexible enough to control identity-based access and identity-less access [22] [6].

3.2 Comparison and result presentation

To solve the limitations of RBAC, researchers have tried to extend existing RBAC model. In this section, we will see how they overcome the limitations of RBAC and how ABAC model handles the comparison criteria, defined in above section.

All the proposed solutions for extending RBAC model is aiming to make possible apply RBAC in the environment where role identification is not enough data to provide authorization. Researches try to extend RBAC with different factors like time, environment, location etc. Figure 4 presents main types of Extended RBAC models like context-based, location-based, location and time based, temporal and environment based, and miscellaneous. These types will be presented in Table 1, where we will evaluate them against the comparison criteria [22].

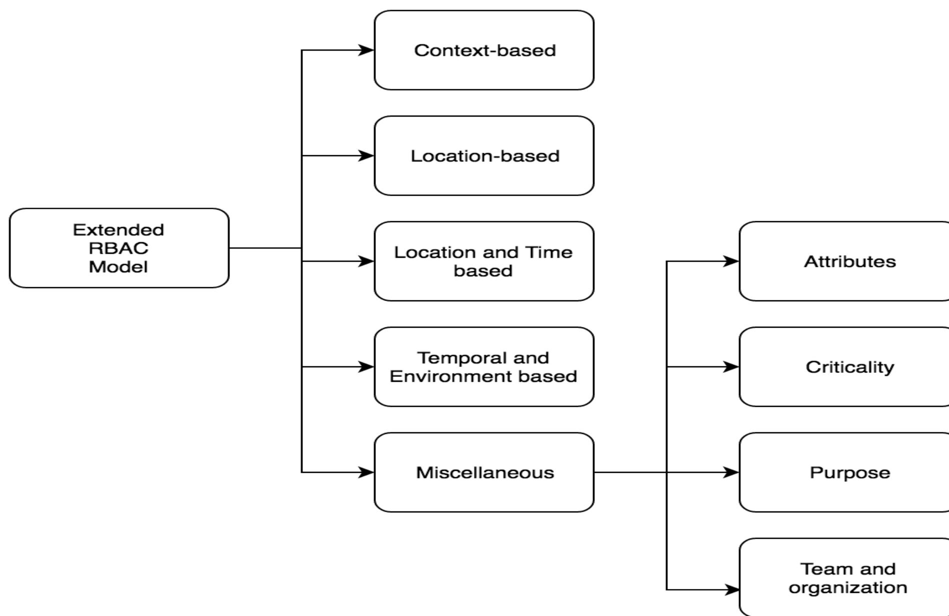

Figure 4 Main types of Extended RBAC model (Adapted from [22])

These types try to make RBAC context-sensitive and dynamic model, but if we look at the big picture we will see there are several issues that extended RBAC models have. First, they are unable to address other requirements like on-going access, flexibility, anonymous access, multi-factor access and other access controls described in above section. Moreover, if we look at Table 1, we will see that extended RBAC models are using different RBAC versions. This may lead to some difficulties if we want to combine them into a single solution because RBAC concept was changing within each version. The difference between extended models doesn't end here, they also support different level of RBAC family. Most of them support flat RBAC, while others can also support hierarchical and constrained RBAC. For example, while proposed solutions in [9] and [10] support flat and Hierarchical RBAC, [11] solution only supports flat RBAC. Moreover, all the proposed solutions are related to some specific target domain. For example, [21] solution proposes Team-based access control (TMAC), which is the solution only for collaborative environments, may not work in other domain, like location based services.

Table 1 A comparison of Role-centric access model and Attribute-centric access model (adapted from [22])

Proposed solution	C1	C2	C3	C4	C5	C6	C7	C8	Targeted Domain	Version
Role-centric Access model										
Context Aware models										
Haibo et al., [9]	✓	✗	✓	✗	✗	✗	2	✗	Web services	RBAC 1996
Covington et al., [10]	✓	✗	✓	✗	✗	✗	3	✗	Pervasive Computing	GRBAC[5] 2001
Zhang et al., [11]	✓	✗	✓	✗	✗	✗	2	✗	Pervasive Grid	RBAC 1996
Location-Based RBAC Models										
Hansen et al., [[12]	✓	✗	✓	✗	✗	✗	2	✗	Mobile Sys-tems	RBAC 1998
Damiani et al., [13]	✓	✗	✓	✗	✗	✗	2	✗	Location-based services	RBAC 2000
Location and Time-Based RBAC Models										
Toahchoodee et al., [14]	✓	✗	✓	✗	✗	✗	3	✗	Dengue sup-port system	STRBAC [6] 2007
Ray et al., [15]	✓	✗	✓	✗	✗	✗	3	✗	Pervasive Computing	STRBAC 2007
Kim et al., [[16]	✓	✗	✓	✗	✗	✗	3	✗	Ubiquitous Applications	RBAC 1996
Temporal and Environment-Based RBAC Models										
Bertino et al., [17]	✓	✗	✓	✗	✗	✗	2	✗	Database Man-agement Sys-tems (DBMS)	RBAC 1996, RBAC 1998
Shafiq et al., [18]	✓	✗	✓	✗	✗	✗	2	✗	Collaborative Environments	GTRBAC [7]2005
Miscellaneous Extended RBAC Models										
Jin et al., [19]	✓	✓	✓	✗	✗	✗	2	✗	Health Care Applications	RBAC 2001
Wang et al., [20]	✓	✓	✗	✗	✗	✗	3	✗	Cooperative Hypermedia Environments	RBAC 1997
Thomas et al., [21]	✗	✓	✓	✗	✗	✗	2	✗	Collaborative Environments	RBAC 1995
Attribute-Centric Access Control										
ABAC model	✓	✓	✓	✗	✓	✓	n	✓	-	-

[5] Generalized role-based access control.
[6] A spatio-temporal role-based access control model
[7] A generalized temporal role-based access control model

Table 1 represents a comparison between the abilities of extended RBAC models and attributes based models. Each row represents the access model, proposed to cover the limitations of RBAC model. The header of the table consists of four parts: proposed solution, criteria, targeted domain and version. Proposed solution is the actual access model. Criteria is the requirements which we defined in section 3.1. We marked each criterion with C_i symbol, so C1 represents *static access*, C2- *fine-grained access,* C3- *context insensitive access,* C4 - *content independent access,* C5 - *on-going access,* C6 - *user prior identification access,* C7 - *multi-factored access* and C8 - *inflexible access.* The ✓ symbol represents that the requirement is fully covered by the proposed access model and ✗ symbol means that requirement is not supported in the proposed access model. For measuring multi-factored access control, we use numbers, where 1 means single factored if access model supports only user role factor. 2 means bi factored if access model supports role factor and more. The integers may go up to any number of factors. Targeted domain represents the environment for which the access model was proposed, in other words, it means that access model satisfies criteria only in the specific domain, like health care system. Under version column there is information about with version of RBAC was used to develop proposed solution. For instance, let's take [20] from the Miscellaneous extended RBAC model type we will see that it supports static access control, fine-grained access control and access decisions are based on three factors. The model was developed for Cooperative Hypermedia Environments and is based on RBAC 1997 version.

From table 1 it is visible that attribute-centric models are more effectively handling the modern access model requirements then extended versions of RBAC. Sure a few extended models can handle some of the requirements, but no one can perform better than ABAC. As we already know that attribute-centric models are better than role-centric models, let's examine in more detail how ABAC is dealing with modern requirements of AC. Attribute-based access model can be easily deployed in dynamic and context sensitive environment, because in ABAC relationship between object and subject is not predefined and context information such as location, time, name and so on can be considered as attributes that describe subject or object. And if context is irrelevant for either user or resource then it can be considered as environment attributes, something like domain name [22].

The rising popularity of XML and JSON formats for using to exchange data created the need for content based access control. Modern access controls should have a possibility to make the decision based on the content of XML or JSON. Fortunately, ABAC model can be used to fulfil this requirement, because it supports XACML policy language with is based on XML and of course it supports XPath [22] [6].

Using attributes ABAC model can support identity-based and anonymous access control. Let's take a scenario where we want to apply identity-based access control. In this case for making access decision, we may need individual or unified identification of the user as an attribute, something like a unique Id of the user or distinct name of the user and so on. On the other hand, to provide anonymous access control, we don't need any user provisioning process, we can use environment attributes for this purpose. Environment variables were developed to handle situations where access decision is not based on user specific information. For example, we can provide clients with free internet if their request will come from specific location and for this we don't need any specific information (ex: role) about the client [22] [6].

As we see from table 1, all the extended versions of RBAC merely provide support for decision factors which is maximum three which should be predefined. Thus, in ABAC as its attributes are categorized in Object, Subject and Environment can provide a large range of contextual information with the ability to modify any decision factor without changing the whole access control model. So ABAC is considered as an "n" time decision factor access control [22] [6].

The concept of attributes makes ABAC inherently flexible access control. As we already discussed ABAC can also support identity-based and identity-less access controls. ABAC has the ability to apply not only RBAC but also DAC and MAC access controls. So how does ABAC can be policy neutral? To answer this question, we should focus more on the essence of traditional access models, how they are achieving desired access control. For instance, the basic aspects of RBAC Policies are to apply the use of role in the system. So, without digging into many details, we can say that ABAC can apply RBAC model by considering user role and only must have the attribute in the model. Similarly, ABAC can cover MAC and DAC models by allowing or restricting data flow using attributes [22].

3.3 Summary

In this chapter, we did a comprehensive analysis of two different models, RBAC and ABAC. We defined the list of security requirements that modern access models should satisfy. After that, we broke down requirements and examined how each model can secure them. In the end, we took several variations of RBAC and analysed compared them against ABAC using the requirement list. In the next chapter, we concentrate on defining prototype parts, creating use cases and requirements. Also, will be illustrated the structure of the prototype and user manual.

4 Prototype

The purpose of this chapter is to present a description, implementation and user manual of the prototype. Specifically, in section 4.1, we will focus on defining application scenario, user characteristics and requirements. The goal is to provide an overview of the whole system and explain its scope and functionality. Next, we will outline all the functionality of the application in use cases, also define the system and software expectations, in other words, what needs to be true for the requirement to be executed. After that, in section 4.2, we will present the implementation of prototype. First, we will describe the implementation of scenario and RBAC via Spring Security Framework. Following that, we will describe the implementation of ABAC and at last, we will illustrate how each model is applied to the scenario. In the end, in section 4.3, we will provide step-by-step guidance of prototype usage.

4.1 Requirements specification

4.1.1 Product perspective

The product is a web based prototype, whose main purpose is to give theoretical and practical knowledge to the users about security models. Using the prototype users can choose different security models, like RBAC and ABAC, and apply them to the application and experience how they work in real life. They will also have a chance to compare their results for better analysis and also write quiz after the experiment to check their understanding of how access models work. Figure 5 illustrates overall structure of this product. Apart from the identified users (Recruiter, Admin, Super Admin), we will have anonymous users who only can log in to the system or register as a user. In order to manage identified users, test results, we need super administrator role. The software will be maintained in the host server, where it will be handled by hosting application. The software is keeping data in the database, which will be handled by database management system. ACCP is web based application so it will be used through internet browsers.

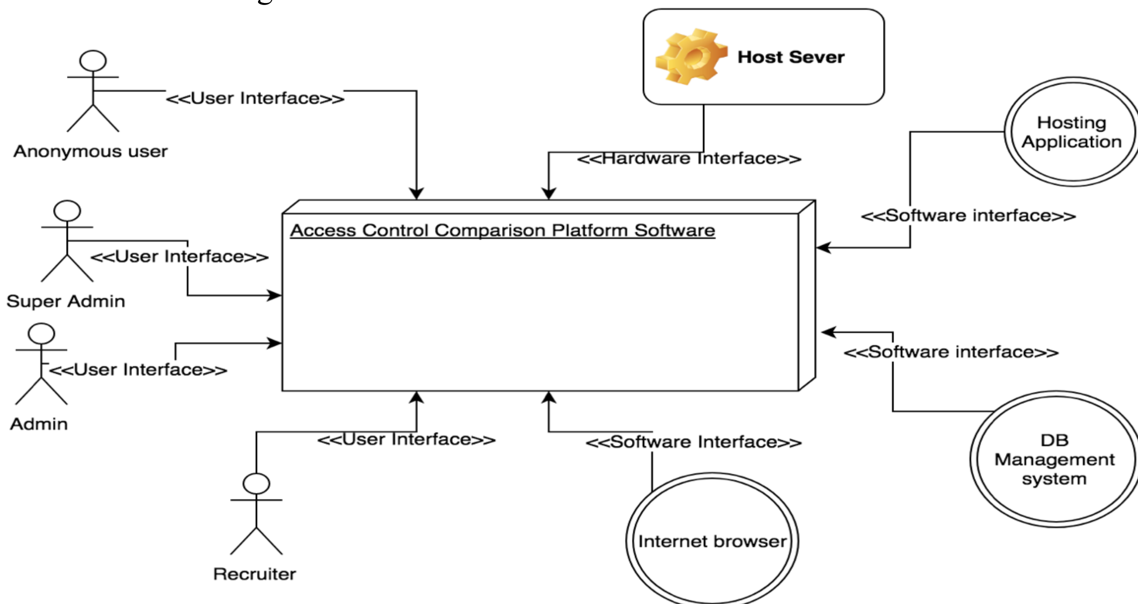

Figure 5 Access control comparison system structure

4.1.2 Scenario description

Access models can be widely used in distributed environments to manage controlled access for applications like database management systems, health care systems, resource sharing

systems and so on. We are considering taking the recruitment management system as our scenario to demonstrate pros and cons of each chosen access models. Managing candidates and positions is crucial for recruiters, where they must deal with many requests. In this kind of application where users interact with the database, it is required to provide security regarding data privacy, data loss and data access.
Recruitment management system is the web application which is broadly categorized into three entities: Companies, Jobs and Candidates. All the entities have attributes providing information about them. Companies represent an employer, which has announced position. Jobs are represented as announced positions and Candidates are assigned to them using position attribute. Overall the application flow goes like this, the user creates the company who is looking for a candidate, under a company user creates jobs and assigns candidates to each job. The entity relations are presented in Figure 6.

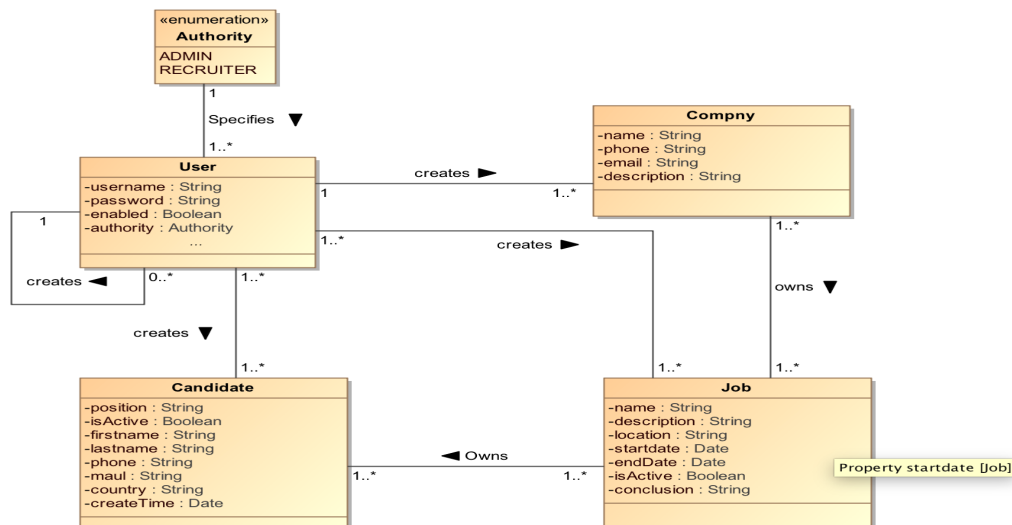

Figure 6 Scenario class diagram.

System has two roles: *administrator* and *recruiter. Administrator* is a super user, with unlimited permissions on all available resources in the system. On the other hand, recruiter role has limited permissions (the role permissions will be explained more thoroughly in below sections). Each user holds unique username and password which is used to log into the system to use offered functionality.

4.1.3 Scenario functions

1. Use case - Prototype introduction

Brief description: Authorized user reads through the introduction of the prototype.
Step-by-step description: To initiate current use case, authorized user should have Admin or Recruiter role.

- Authorized user opens application and browses introduction page
- Authorized user reads through scenario description
- Authorized user reads about secured resources in the system
- Authorized user reads about user roles and permissions assigned to the role
- Authorized user reads about which models he/she can apply to the scenario

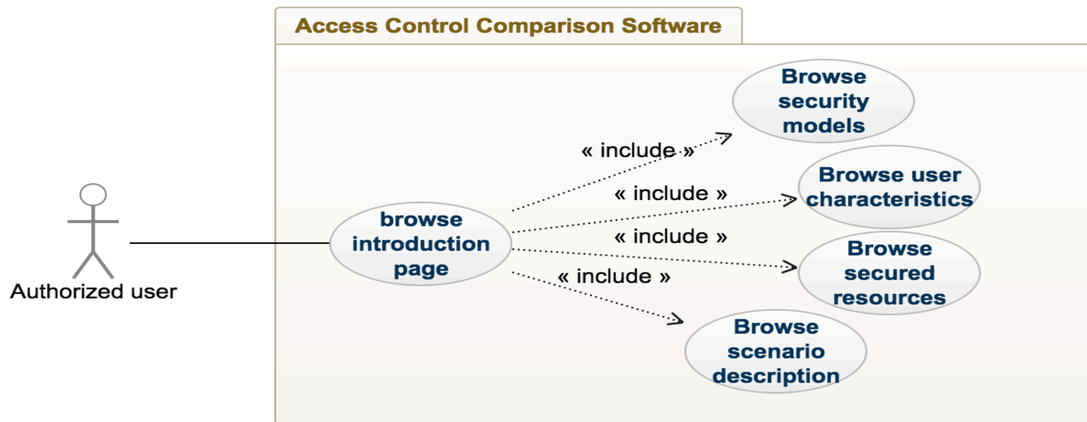

Figure 7 Introduction of the prototype use case

2. Use case - compare security models

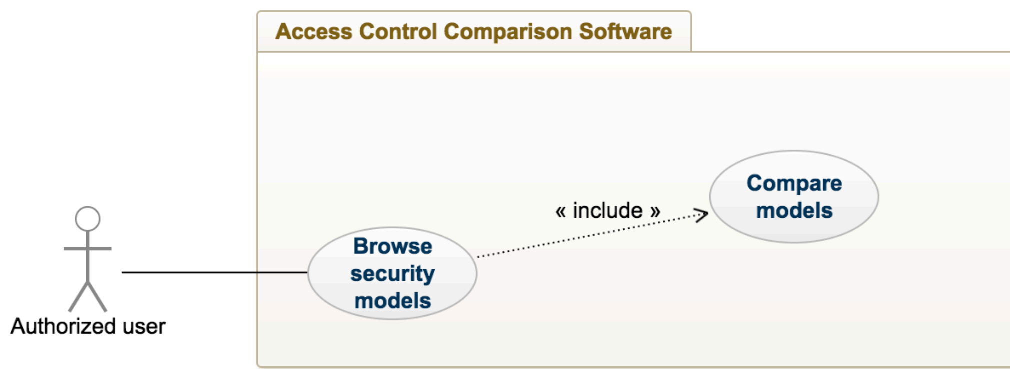

Figure 8 Analytical comparison of security models use case.

Brief description: Authorized user opens available security models' description page.
Step-by-step description: To initiate current use case, authorized user should have Admin or Recruiter role.

- Authorized user opens security models' description page
- Authorized user chooses button "Compare models"
- Authorized user is redirected to security models' comparison page
- Authorized user reads through the analytical comparison of ABAC and RBAC

3. Use case - Apply access model to the scenario.

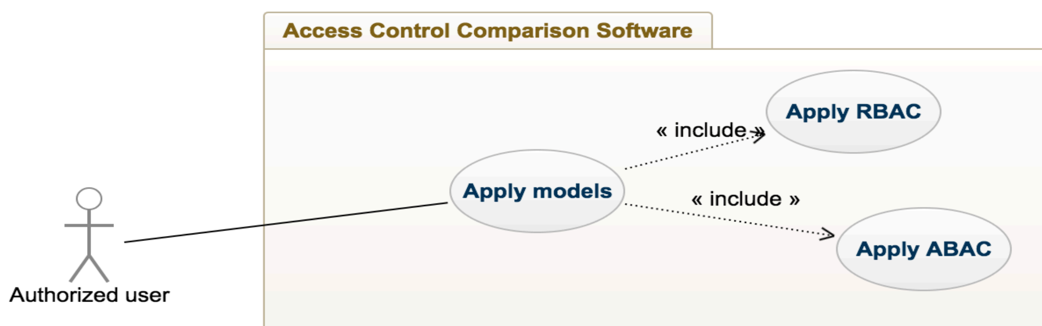

Figure 9 Apply security access model use case.

Brief description: Authorized user applies chosen security access model to the scenario application.

Step-by-step description: To initiate current use case, authorized user should have Admin or Recruiter role.

- Authorized user clicks on "Apply Access model" dropdown menu
- Authorized user picks desired security model from list.
- Authorized user presses the button "Apply model"

4. Use case - List companies in the system

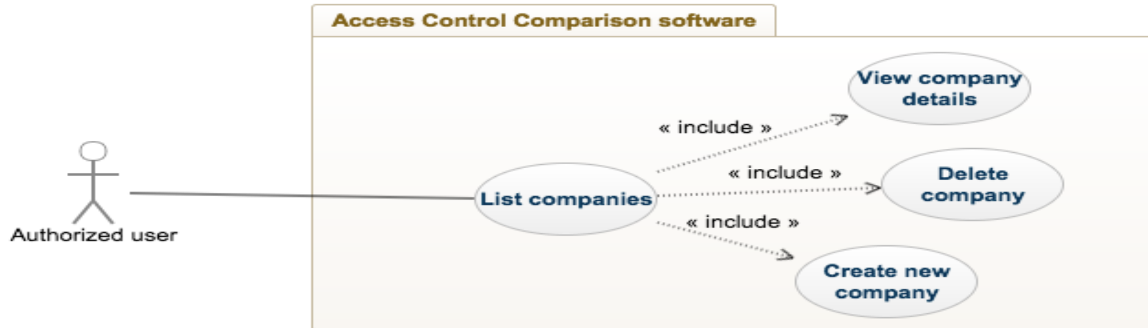

Figure 10 List companies in the system use case.

Brief description: Authorized user gets the list of companies.
Step-by-step description: To initiate current use case, authorized user should have Admin or Recruiter role.

- Authorized user browses company list page.
- System queries all the companies in the system.
- Companies page is opened and all found items are displayed.
- Additional functionality is shown for the authorized user via buttons "Create new company", "Delete company", "View company detail"

5. Use case - Create new company

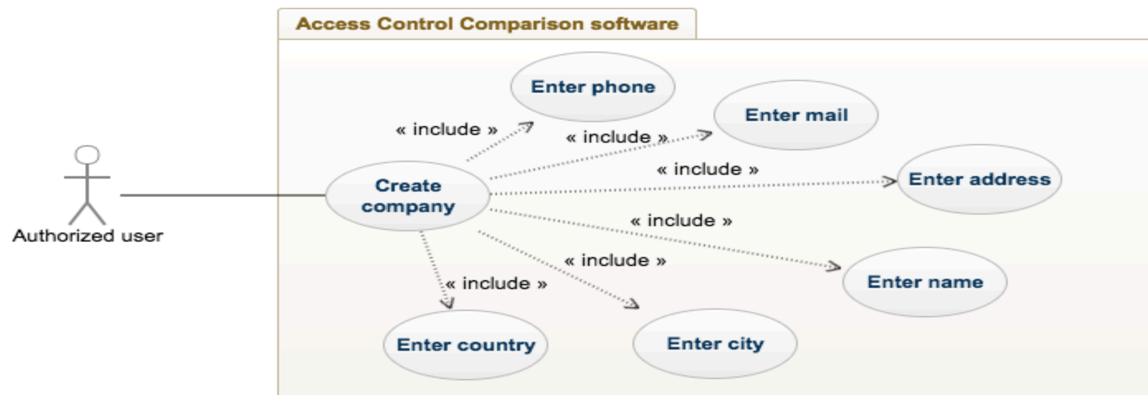

Figure 11 Add new company in the system use case

Brief description: Authorized user creates a new company.
Step-by-step description: To initiate current use case, authorized user should have an Admin role and companies page should be opened.

- Authorized user chooses button "Create new company"
- New form is opened with setting for new company like "Company name", "Address", etc.

- Authorized user fills form fields.
- Authorized user presses button "Submit".
- The system saves new company in database.
- Authorized user is redirected to company list page.

6. Use case - delete company

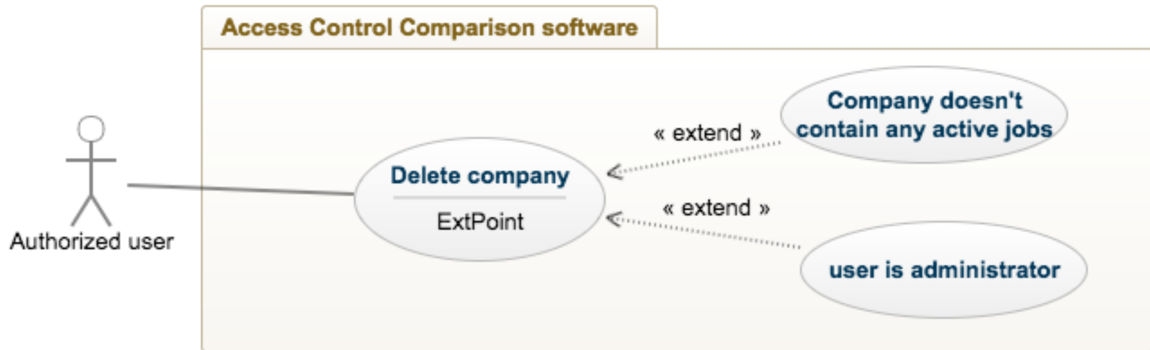

Figure 12 Delete company from the system use case.

Brief description: Authorized user removes a company from list.
Step-by-step description: Before executing this use case authorized user should have an Admin role and a company has to exist in the database.

- Authorized user selects specific company from the company's collection.
- Authorized user presses the button "Remove company".
- Authorized user is asked for confirmation "Are you sure to delete {company name}?"
- Authorized user confirms his/her choice.
- System checks if current company has any active job
- System removes the company from the database and updates company list page with new results.

7. Use case - View company details.

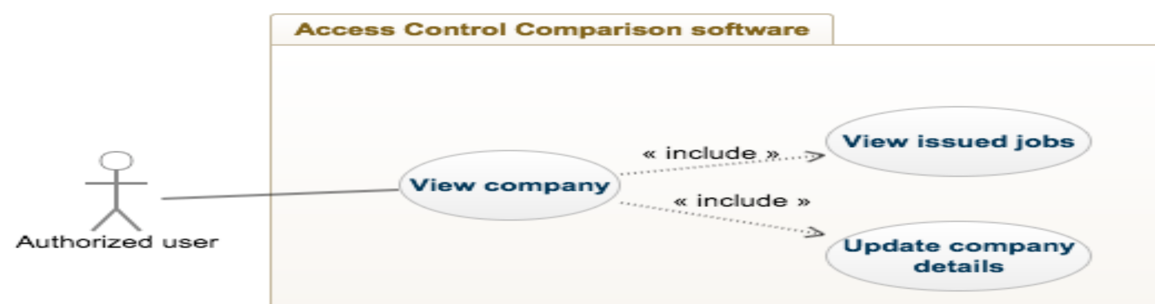

Figure 13 View company detail from company list page use case.

Brief description: Authorized user views a company detail page from the list
Step-by-step description: Before initialization of this use case, Authorized user should have as an Admin or Recruiter role and a company should exist in the database.

- Authorized user clicks on a specific company from the collection
- Authorized user is redirected to the company details page
- Authorized user views company details, like "Company name", "address" etc.
- Authorized user browses the list of the jobs issued by this company.

- Additional functionality via button "Edit company" is show to the authorized user in case he/she wants to edit company information.

8. Use case - Update company

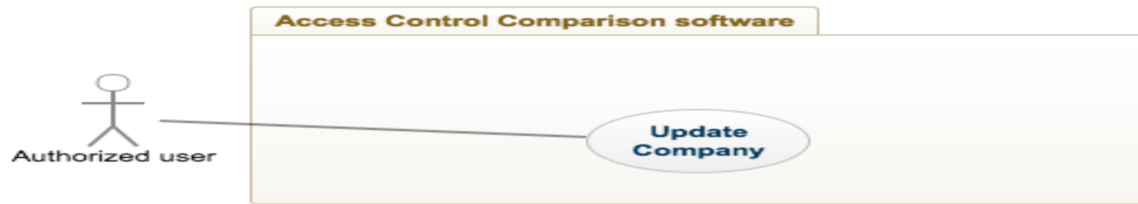

Figure 14 Update specific company use case.

Brief description: Authorized user updates a company from company detail view.
Step-by-step description: Before executing this use case authorized user should have an Admin role and a company should exist in the database.

- Authorized user selects specific company from the company's collection.
- Authorized user is redirected to the company details page
- Authorized user press button "Update company"
- Authorized user alters the company fields.
- Authorized user clicks button "Save changes"
- System updates current company information in the database

9. Use case - List candidates in the system

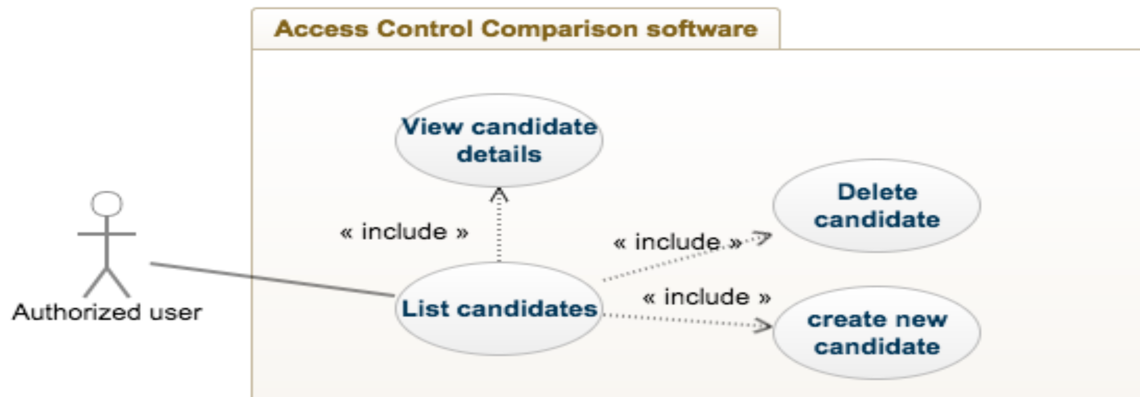

Figure 15 List all candidates in the system use case.

Brief description: Authorized user gets the list of candidates.
Step-by-step description: To initiate current use case, authorized user should have an Admin or Recruiter role.

- Authorized user browses candidate list page.
- System queries all the candidates in the system.
- Candidates page is opened and items are displayed.
- Additional functionality is shown for authorized user via buttons "Create new Candidate", "Delete candidate", "View candidate detail"

10. Use case - create new candidate

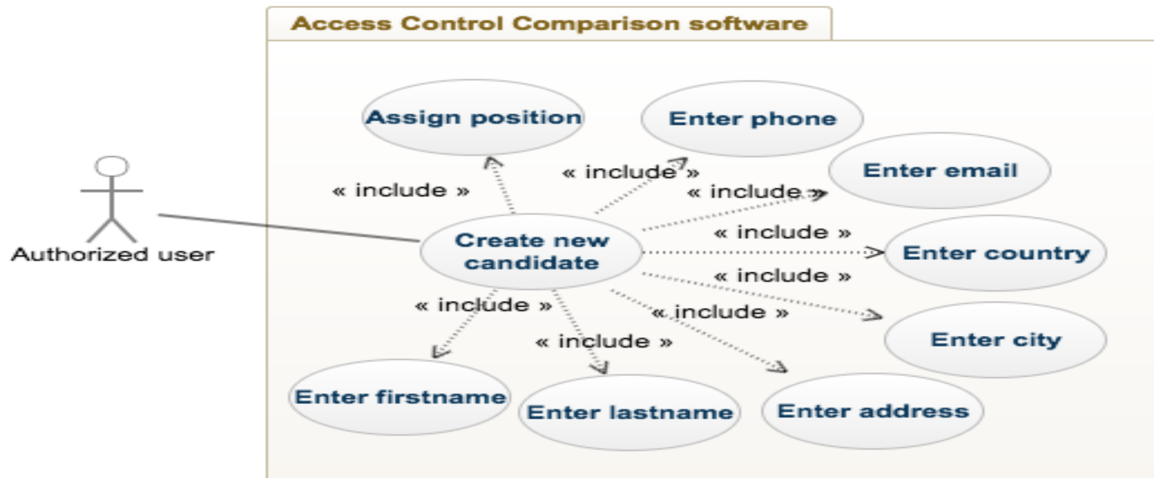

Figure 16 Add new candidate in the system use case.

Brief description: Authorized user creates a new company.
Step-by-step description: To initiate current use case, authorized user should have an Admin or Recruiter role and jobs page should be opened.

- Authorized user chooses button "Create new candidate"
- New form is opened with setting for new candidate like "Name", "Address", etc.
- Authorized user fills form fields.
- Authorized user assigns job to the candidate
- Authorized user presses button "Submit".
- The system saves new candidate in database.
- Authorized user is redirected to candidates list page.

11. Use case - delete candidate

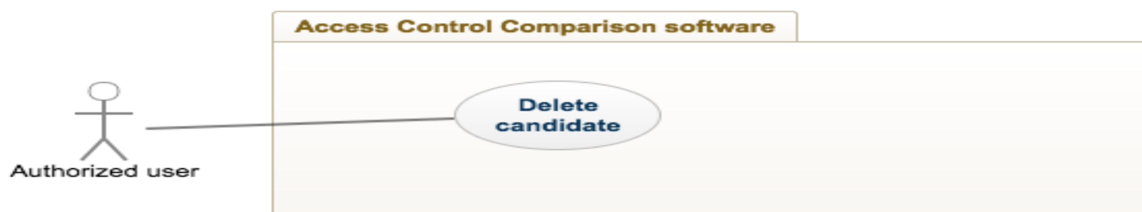

Figure 17 Delete candidate from the system use case

Brief description: Authorized user removes a candidate from list.
Step-by-step description: Before executing this use case authorized user have an Admin or Recruiter role and a candidate has to exist in the database.

- Authorized user selects specific candidate from the candidate's collection.
- Authorized user presses the button "Remove candidate".
- Authorized user is asked for confirmation "Are you sure to delete {candidate name}?"
- Authorized user confirms his/her choice.
- System removes the candidate from the database and updates candidate list page with new results.

12. User case - view candidate details.

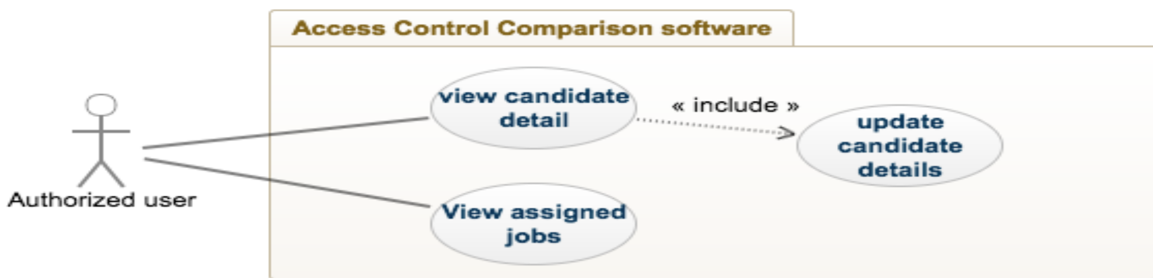

Figure 18 View candidate detail from candidate list page use case.

Brief description: Authorized user views a candidate detail page from the collection
Step-by-step description: Before initialization of this use case, authorized user should have an Admin or Recruiter role and a candidate should exist in the database.

- Authorized user clicks on a specific candidate from the collection
- Authorized user is redirected to the candidate details page
- Authorized user views candidate details, like "Name", "address" etc.
- Authorized user browses the list of the jobs assigned to the candidate.
- Additional functionality via button "Edit candidate" is show to the person in case he/she wants to edit candidate information.

13. Use case - Update candidate

Figure 19 Update specific candidate use case.

Brief description: Authorized user updates a candidate from candidate detail view.
Step-by-step description: Before executing this use case authorized user should have an Admin or Recruiter role and a candidate should exist in the database.

- Authorized user selects specific candidate from the candidate's collection.
- Authorized user is redirected to the candidate details page
- Authorized user presses button "Update candidate"
- Authorized user alters the candidate fields.
- Authorized user clicks button "Save changes"
- System updates current candidate information in the database

14. Use case - List Jobs in the system

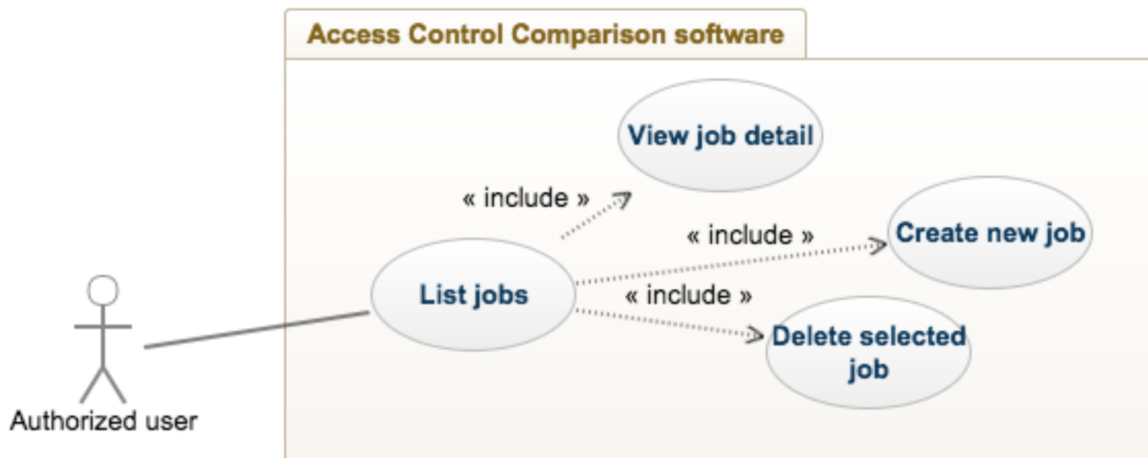

Figure 20 List jobs in the system use case.

Brief description: Authorized user gets the list of jobs.
Step-by-step description: To initiate current use case, Authorized user should be signed in the application as Admin or Recruiter role.

- Authorized user browses candidate list page.
- System queries all the jobs in the system.
- Jobs page is opened and items are displayed.
- Additional functionality is shown for the authorized user via buttons "Create new job", "Delete job", "View job details"

15. Use case - Delete job

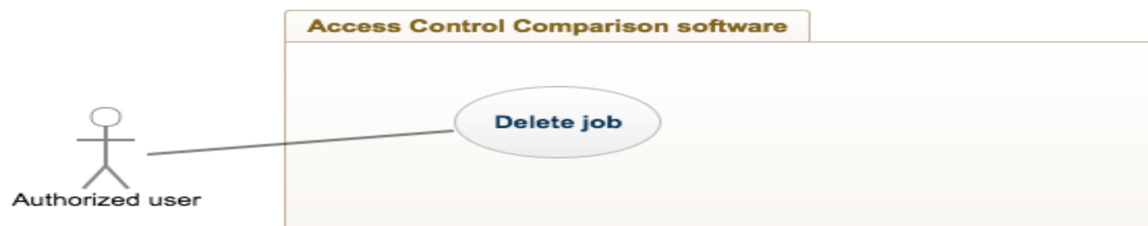

Figure 21 Delete job from the system use case.

Brief description: Authorized user removes a job from list.
Step-by-step description: Before executing this use case authorized user should have an Admin role and a job should exist in the database.

- Authorized user selects specific job from the job's collection.
- Authorized user press the button "Remove job".
- Authorized user is asked for confirmation "Are you sure to delete {job name}?"
- Authorized user confirms his/her choice.
- System removes the job from the database and refreshes job list page with updated results.

16. Use case - View Job details

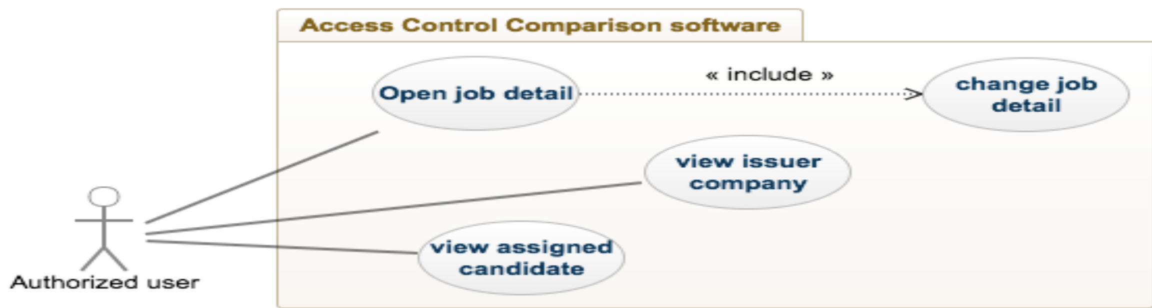

Figure 22 View job detail from job list page use case.

Brief description: Authorized user views a job detail page from the collection
Step-by-step description: Before initialization of this use case, authorized user should have an Admin or Recruiter role and a job should exist in the database.

- Authorized user clicks on a specific job from the collection
- Authorized user is redirected to the job details page
- Authorized user views job details, like "Title", etc.
- Authorized user browses the list of the candidates assigned to the current job.
- Authorized user browses issuer company
- Additional functionality via button "Edit job status" is shown to authorized user in case he/she wants to change job status.

17. Use case - Update job status

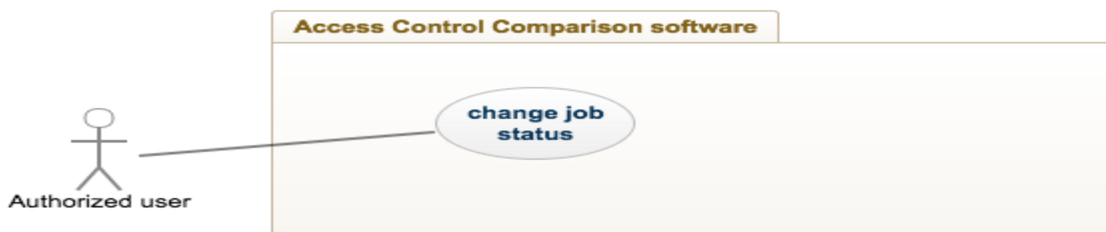

Figure 23 Change job status use case.

Brief description: Authorized user updates a job status from job detail view.
Step-by-step description: Before executing this use case Authorized user should have an Admin or Recruiter role and a job should exist in the database.

- Authorized user selects specific job from the job's collection.
- Authorized user is redirected to the job details page
- Authorized user presses button "Update job status"
- Authorized user alters job status field.
- Authorized user clicks button "Save changes"
- System updates current job's status in the database

18. Use case - Create Job

Figure 24 Add new job in the system use case.

Brief description: Authorized user creates a new job.
Step-by-step description: To initiate current use case, authorized user should have an Admin or Recruiter role and job page should be opened.

- Authorized user chooses button "Create new job"
- New form is opened with setting for new candidate like "Title", "Description", etc.
- Authorized user fills form fields.
- Authorized user assigns job to the company
- Authorized user presses button "Submit".
- The system saves new job in database.
- Authorized user is redirected to jobs list page.

19. Use case - change authority role

Figure 25 Change authority role use case

Brief description: Authorized user changes the role of logged in user.
Step-by-step description: To initiate current use case, Authorized user should have an Admin or Recruiter role.

- Authorized user presses the link "Update user "
- Authorized user is redirected to the user details page.
- Authorized user press button "Update user role"
- Authorized user alters user role field.
- Authorized user clicks button "Save changes"
- System updates current user's authority in the database

20. Use case – Authorization

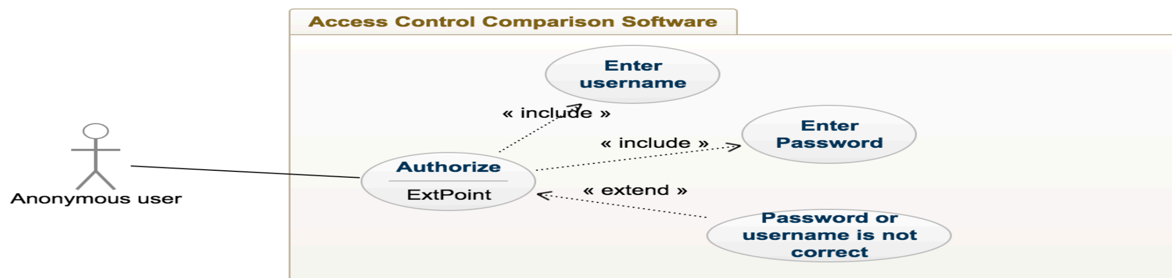

Figure 26 Authorize into the system use case

Brief description: Anonymous user authorizes into the ACCP system.
Step-by-step description:

- Anonymous user opens application
- Anonymous user enters username and password.
- Anonymous user clicks on "Log in"
- Anonymous user logs into the system.
- Anonymous user sees error message if username or password is wrong

21. Use case – Registration

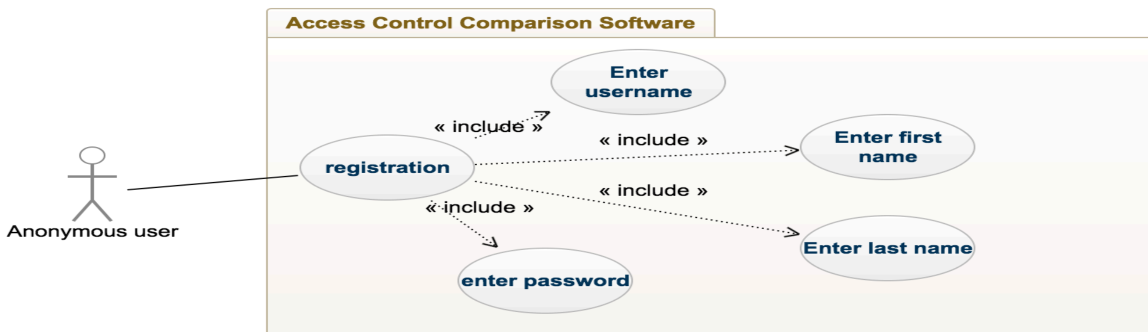

Figure 27 Registration of new user use case.

Brief description: Anonymous user registers new user into the ACCP system.
Step-by-step description:

- Anonymous user opens application
- Anonymous user clicks on registration tab
- Anonymous user enters username, first name, last name and password
- Anonymous user clicks on the button "Register now"
- Anonymous user is redirected to Log in form.

22. Use case – Quiz

Brief description: Authorized user takes a quiz.
Step-by-step description: To initiate current use case, Authorized user should have an Admin or Recruiter role

- Authorized user opens quiz page
- Authorized user clicks on the button "Start quiz" to start the quiz
- Authorized user answers to the questions
- Authorized user submits the quiz
- Authorized user sees the score.

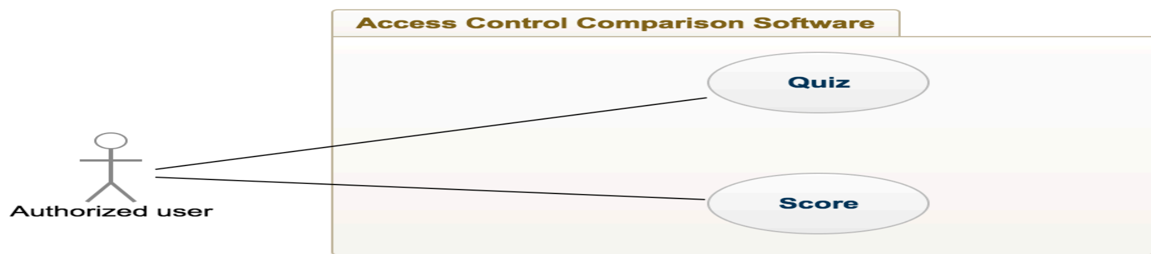

Figure 28 Take a quiz use case

23. Use case – Quiz result management

Brief description: Authorized user manages quiz results.
Step-by-step description: To initiate current use case, Authorized user should have an Super Admin role.

- Authorized user loads the results of the users who have taken quiz.
- Authorized user filters user list using search
- Authorized user resets result list.

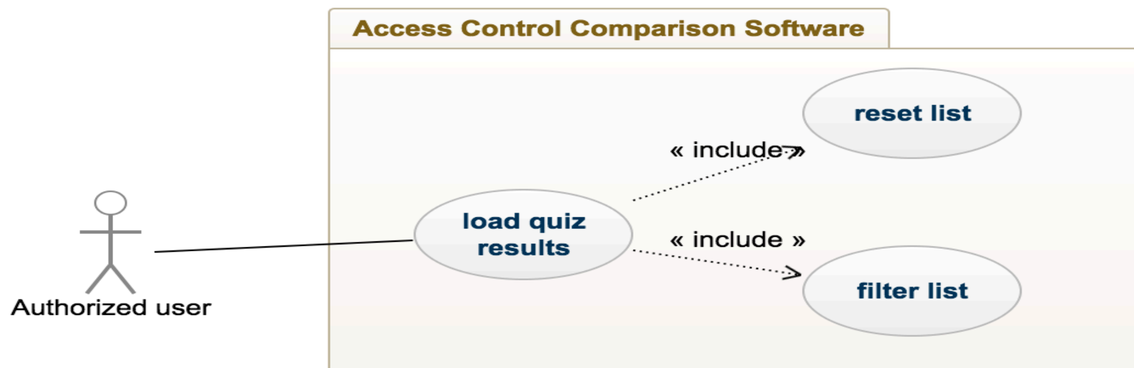

Figure 29 Quiz result management use case.

4.1.4 Expectations

Table 2 defines system expectations and Table 3 defines software expectations for the prototype

Table 2 System expectations

ID	Statement	Description	Source
SE_1	User enters the webpage	For using analyse prototype user mast open web application using browser.	
SE_2	Job shouldn't contain any assigned candidates before removal	Assumption that job cannot be removed if it contains active candidates. otherwise the link between candidate and assigned job will be disappeared.	#15 Use case- Delete job
SE_3	Only administrator can remove Job from system	Assumption that system should check user authority before deleting Job resource.	#15 Use case- Delete job
SE_4	Company mustn't contain any active jobs	Assumption that company cannot be removed if it contains active jobs. otherwise the link between company and announced job will be disappeared.	#6 Use case-Delete Company

SE_5	Only administrator can remove Company from system	Assumption that system should check user authority before deleting Company resource.	#6 Use case-Delete Company
SE_6	Only administrator can create Company from system	Assumption that system should check user authority before creating Job resource.	#5 Use case- Create Company
SE_7	Only administrator can update Company data	Assumption that Recruiter users can only issue a new job or finish current one. Administrator users can do all.	#8 Use case - update company

Table 3 Software expectations.

ID	Statement	Description	Source
SWE_1	User should be registered	As ACCP have quiz functionality we need to register each user to store their performance.	
SWE_2	User must be logged in	For all actions in the prototype the user must be logged in.	#3 Use case - Apply access model to the scenario

4.1.5 System requirements

Table 4 System requirements for prototype.

ID	Statement	Description
SR_1	Server internet connection must be active.	As the prototype is hosted through remote server, user should have an access over internet to use the prototype.
SR_2	Prototype should be held in cloud based service.	for security and easy accessibility prototype should be hosted by cloud services such as Heroku

4.1.6 Authentication

Table 5 Authentication function requirements for prototype.

ID	Statement	Description	Traceability
AR_1	Identified "Recruiter" type users can access the system, using login page.	All visitors can authorize into the system after choosing desired security model. Users who are identified as "Recruiter" should have access to issues jobs from the company and candidates assigned to them.	SWE_1
AR_2	Identified "Administrator" type users can access the system, using login page.	All visitors can authorize into the system after choosing desired security model. Users who are identified as " Administrator " should have an unlimited access to issued jobs from the company and candidates assigned to them. No other user type can delete resources from system.	SWE_1

AR_3	Users can log out.	User can log out from system to end session.	
AR_4	Anonymous users can register	All visitors can register new user if they don't' have one.	

4.1.7 Scenario requirements

Table 6 Scenario functional requirements.

ID	Statement	Description	Source	Traceability
SFR_1	Application Should enable to create new company	• User can add new company in the system. • When adding new company user should enter following required fields: name, city, country, email, phone, address.	#5 Use case-Create Company	SE_6
SFR_2	Application should enable to remove company from list.	• User can delete company if all conditions are satisfied	#6 Use case-Delete Company	SE_5, SE_4
SFR_3	Application should enable access to companies list	• Users can see companies list in the system. • In the list view each company item has following fields: company name and delete button. • Each company item should be linked to its details page.	#4 Use case - List available companies in the system	
SFR_4	Application should enable access to company details.	• Users can see the details of the company. • Administrator and recruiter users should see which jobs where announced by current company.	#7 Use case - View company details	
SFR_5	Application should enable updating company details.	• User can change company information • user can issue new job or finish current one.	#8 Use case - Update company	SE_7
SFR_6	Application Should enable to create new candidate	• User can add new candidate in the system.	#10 Use case - create new candidate	

		• when adding new candidate should enter following fields: first name, last name, country, city, address, email. • User should assign job position to the current candidate.		
SFR_7	Application should enable to remove candidate from list.	• User can delete candidate from the system.	#11 Use case - delete candidate	
SER_8	Application should enable access to candidates list	• Administrator and recruiter users can see available candidates list in the system. • In the list view each candidate should have following fields: candidate first name and last name, delete button. • Each candidate should be linked to its details view.	#9 Use case- List all available candidates in the system.	
SER_9	Application should enable access to candidate details.	• User can view the details of candidate. • User should see which jobs are assigned to the candidate	#12 use case - view candidate details	
SER_10	Application should enable updating candidate details.	• User can update candidate information. • User should have possibility to change candidate status.	#13 Use case - update candidate	
SER_11	Application Should enable to create new job	• user can add new job in the system. • when adding new candidate should enter following fields: title and description. • User should assign issuer company to the current job.	#18 use case - create job	
SER_12	Application should enable to remove job from list.	• User can delete job from the system.	#15 Use case - delete job	SE_3, SE_2

SER_13	Application should enable access to jobs list	• User can see jobs list in the system. • In the list view each candidate should have following fields: job title, delete button. • Each candidate should be linked to its details view.	#14 Use case - List available Jobs in the system	
SER_14	Application should enable updating job details.	• User can update job information. • User should have possibility to change job status.	#17 Use case - update job status	

4.1.8 Prototype requirements

Table 7 Prototype functional requirements.

ID	Statement	Description	Source
ACR_1	Prototype introduction page should be accessible	Introduction page is meant to provide information about: • what types of security model visitors can compare using this prototype, • what type of user roles have the system, • what kind of permissions have each type of role in the system, • what kind of secured resources are in the system, • description of the system, what is does this system do, what is the purpose of this system and how to use it.	#1 Use case - Platform introduction
ACR_2	user can apply access models to the system.	• Users can choose desired security model and apply to the system.	#3 Use case - Apply access model to the scenario.
ACR_3	Page visitor can select access models to compare.	• Anonymous and authorized users can choose access models they want and see the analytical comparison of the chosen access models via comparison page.	#3 Use case - compare security models
ACR_3	User can change own authority to "Administrator"	• All actions for candidate resource should be enabled • Delete operation should be enabled for Job resource, which means that delete button should be enabled.	#19 Use case - change authority role

ACR_4	User can change own authority to "Recruiter"	• All actions for candidate resource should be enabled • Delete operation should be disabled for Job resource, which means that delete button should be disabled.	#19 Use case - change authority role
ACR_5	Prototype should log every step of security actions in the browser	• Prototype should log step-by-step actions about how it secures resource and authorizes user.	
ACR_6	Authorized user can take a quiz for measuring his/her knowledge.	• Prototype should have a page where users can take a quiz and see the score of their performance.	22. Use case - Quiz
ACR_7	Super user should have a separate page where he will see the results of quiz.	• Prototype should have a admin panel for managing quiz scores. • User should be able to see the name and score • User should be able to reset scores.	23. Use case – Quiz result management

4.2 Implementation

4.2.1 Scenario and Role-Based Access Control (RBAC) implementation

For implementing scenario, we have used spring MVC framework. Which means that our scenario application consists of three major layers: Web layer, Service layer and Repository access layer. The overall layout of our scenario is demonstrated in Figure 29. Web layer is the top layer of our web application. Its main responsibility is to process user's request and return a correct response.

Since web layer is the entry point of our application, it should provide exception handling thrown by other layers and take care of unauthorized requests. The service layer is located under Web layer. It plays the role of the transactional barrier and contains application and infrastructure services. It is responsible for authorization and communication with external resources (in our case it may be remote policy repository). Repository layer which provides necessary methods to provide action on the Domain layer. Domain layer is the lowest layer of the application, contains the list of the entities which is operated by other layers.

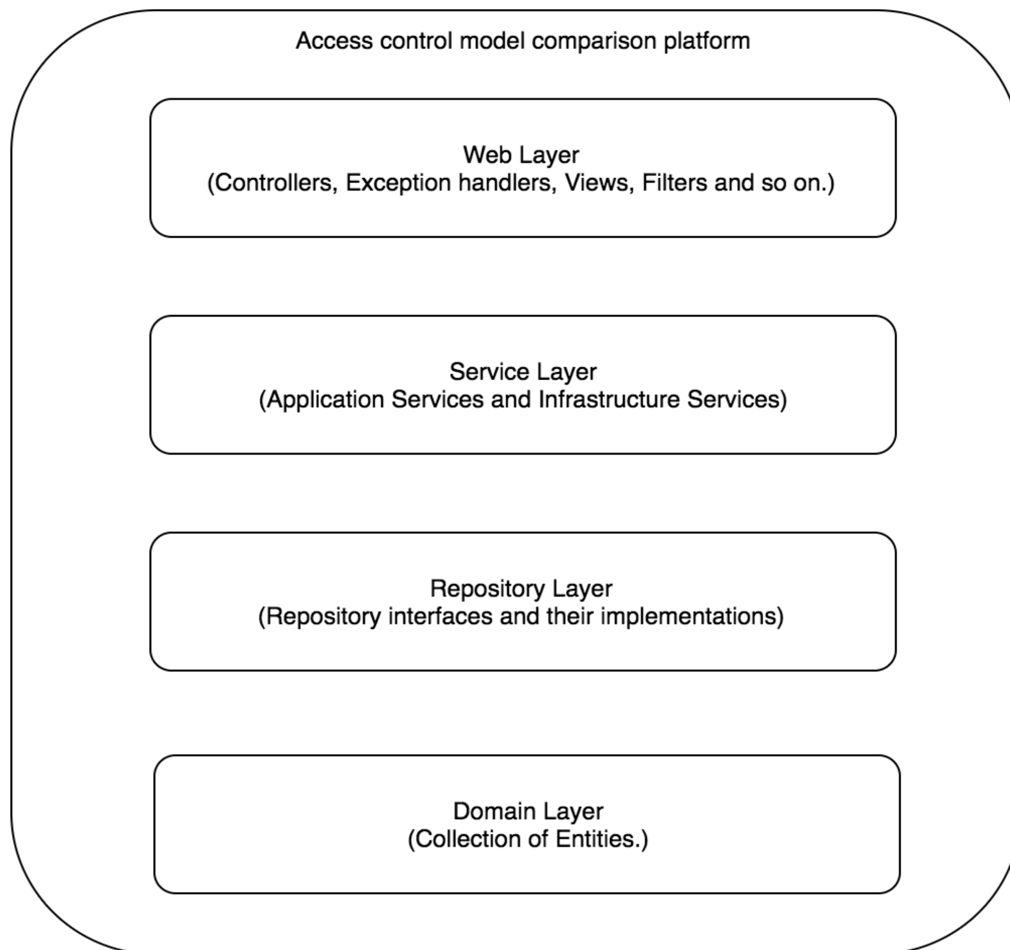

Figure 30 Prototype layout.

Spring MVC framework provides us with annotation to define each layer in the system. Web layer is annotated as *@RestController*, Service layer is annotated as *@Service*, Repository layer is annotated as *@Repository* and Domain layer is annotated as *@Entity* annotation.

Since we have explained each layer of our application, now it is time to implement role based access model and apply it to our controllers. Figure 30 presents the configuration class where security access settings are defined. With this configuration, we are saying that all http requests received by the server should be authorized, specifically it provides two types of configuration. First configuration controls that only admin users can send http endpoint that matches with "*/admin/***" pattern to a server. Second configuration permits anonymous users to log in page and so on. Moreover, it checks that any other requests to the server should be from authorized users and they should have either "Admin" or "Recruiter" role.

```
@Configuration
@ComponentScan("com.recruiter.abac")
@PropertySource(value="classpath:application.properties")
@EnableWebSecurity
@EnableGlobalMethodSecurity(securedEnabled = true)
public class WebSecurityConfig {

  @Configuration
  @Order(1)
  public static class ApiWebSecurityConfigurationAdapter extends WebSecurityConfigurerAdapter {
    protected void configure(HttpSecurity http) throws Exception {
      http
          .csrf().disable()
          .antMatcher("/admin/**")
          .authorizeRequests().anyRequest().hasRole( role: "ADMIN")
          .and()
          .httpBasic();
    }
  }

  @Configuration
  public static class FormLoginWebSecurityConfigurerAdapter extends WebSecurityConfigurerAdapter {
    @Override
    public void configure(WebSecurity web) throws Exception {
      web.ignoring().antMatchers( ...antPatterns: "/comparison","/models","/resources","/system_users","/scenario","/");
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
      http
          .authorizeRequests()
          .antMatchers( ...antPatterns: "/public/**").hasAnyRole( ...roles: "ADMIN","RECRUITER")
          .anyRequest().authenticated()
          .and()
          .formLogin().permitAll()
          .and()
          .logout().permitAll();

      http.csrf().disable().headers().frameOptions().disable();
    }
  }
```

Figure 31 Spring security configuration file

Now that we defined security configuration file for access models let's illustrate how RBAC is applied to our application controllers, specifically company resource controller, presented in Figure 31.

We marked *CompanyControllerRBAC* with *@RestController* (line 18) annotation, which means that Spring framework will perceive this class as a controller. Next, we injected company service class (line 24), which provides our controller with methods for interfering with company repository. Each method in this controller represents rest endpoint, which means that when server receives a http request these methods will be executed and response will be returned to the user. From the scenario requirements section (4.1.5) we know that only Admin users can create and delete company resource, so these rest endpoints are marked with *@Secured* annotation (line 29 and 49), which demonstrates the method level security feature of Spring. It is applied for role '*ROLE_ADMIN*'. These methods will be only mapped to the request matching *"/admin/**"* pattern.

```
17  @Slf4j
18  @RestController
19  public class CompanyControllerRBAC {
20
21      private CompanyService companyService;
22
23      @Autowired
24      public CompanyControllerRBAC(CompanyService companyService) { this.companyService = companyService; }
27
28      @RequestMapping(value = "admin/companies", method = POST)
29      @Secured({"ROLE_ADMIN"})
30      public void createCompany(@RequestBody CompanyResource resource) { companyService.create(convert(resource)); }
33
34      @RequestMapping(value="public/companies", method = GET)
35      public Collection<CompanyResource> getAllCompanies() {
36          Collection<CompanyResource> companyResources = new ArrayList<>();
37          companyService.getAll().forEach(item -> {
38              companyResources.add(convert(item));
39          });
40          return companyResources;
41      }
42
43      @RequestMapping(value = "public/companies/{id}", method = GET)
44      public CompanyResource getCompany(@PathVariable Long id) { return convert(companyService.get(id)); }
47
48      @RequestMapping(value = "admin/companies/{id}", method = DELETE)
49      @Secured({"ROLE_ADMIN"})
50      public void deleteCompany(@PathVariable Long id) { companyService.delete(id); }
53
54  }
```

Figure 32 Company resource controller with RBAC access model.

RBAC policies

For sake of illustrating secureUML with different levels of RBAC model, we will use an example of web application of Recruitment management system described in 4.1.2 section (Figure 6), which has different access constraints, roles, and activities. With our example, we will define the model and visualize the implementation of the project using RBAC mechanism. To demonstrate different types of RBAC requirements, the conditions for access control will escalate from simple to complex model. So first let's identify possible ActionTypes over resources in the application.

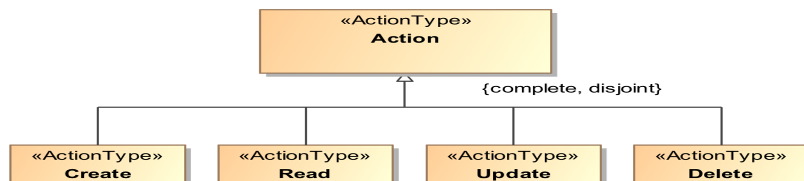

Figure 33 Security actions types in HR management system

As shown in Figure 32 we have four possible actions on each abstract resource: CREATE, READ, UPDATE, and DELETE. Now let's define the hierarchy of application's users. We will have two roles, administrator, and recruiter. Administrators will oversee managing recruiters, candidates, companies and jobs. On the other hand, recruiters will oversee managing their own data, candidates, companies and jobs.

Flat RBAC

In Figure 33 we present relations and access dependencies between roles and resources in flat RBAC. First, we should define multiple users of the system: Saba, Natia, and Giorgi. Then we should define roles, we will have two roles: Administrator and Recruiter. Both roles represent the role inside the organization with required authorization. This means that Saba, Natia, and Giorgi will be assigned to their roles after successful authorization procedure. As resources, we have candidates, companies, recruiters, and jobs with their attributes and operations. Recruiter and Administrator have similar access (CRUD operations) on candidates and jobs, but for recruiter resource recruiter role has only read and update access only for own account data. And for company resource only create and update access. On the other hand, the administrator has full access to company and recruiter resources.

Hierarchical RBAC

Hierarchical RBAC includes all principals of flat RBAC and gives us advantages of using role hierarchy. AS it is illustrated in Figure 34 Administrator role inherits recruiter role. Thus, the administrator role doesn't require having a separate definition of CRUD actions for candidates and jobs. And read, create, update actions for companies. In our case Recruiter resource is an exception, even though both roles have update permission, administrator role still doesn't inherit it from recruiter role, because recruiter can only update own data.

Constrained RBAC

The constrained RBAC not only extends the principals of hierarchical RBAC but introduces the concept of duty separation. From Figure 35 we can see that both roles have precondition check to manipulate recruiter resource. Recruiter role needs recruiter ownership constraint check, which checks if the user is the owner of the account before updating recruiter resource and administrator role needs Recruiter status constraint check, which checks if selected recruiter is not active before deleting the resource. Other constraints are on company resource. To delete a resource, administrator role should fulfil Company status constraint check, which checks if company contains any active jobs. Finally, there is one more constraint on job resource for both roles. The functionality requirement behind Job status constant is that recruiter or cannot delete the resource if job contains any number candidates.

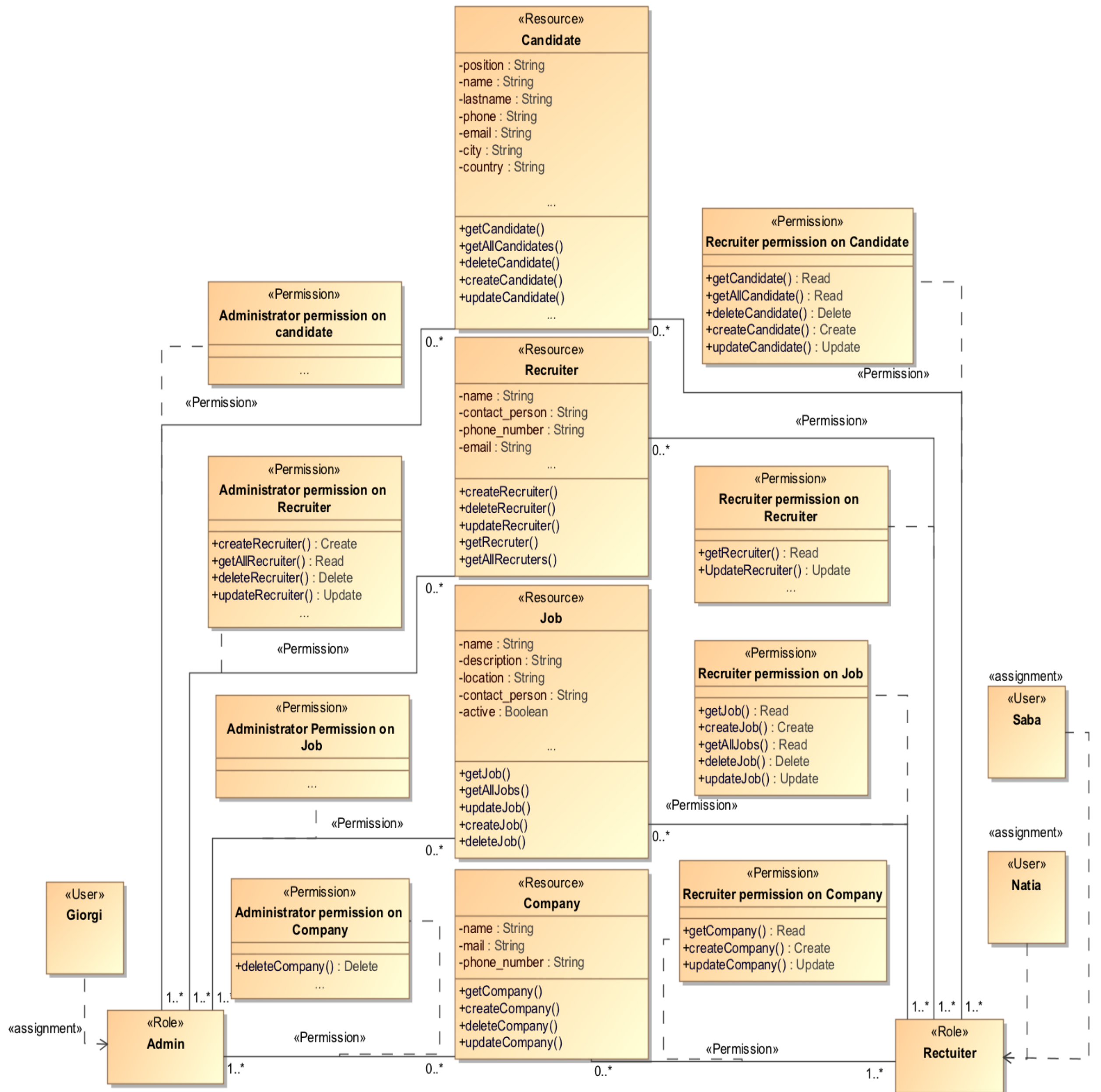

Figure 34 An example of SecureUML model of flat RBAC

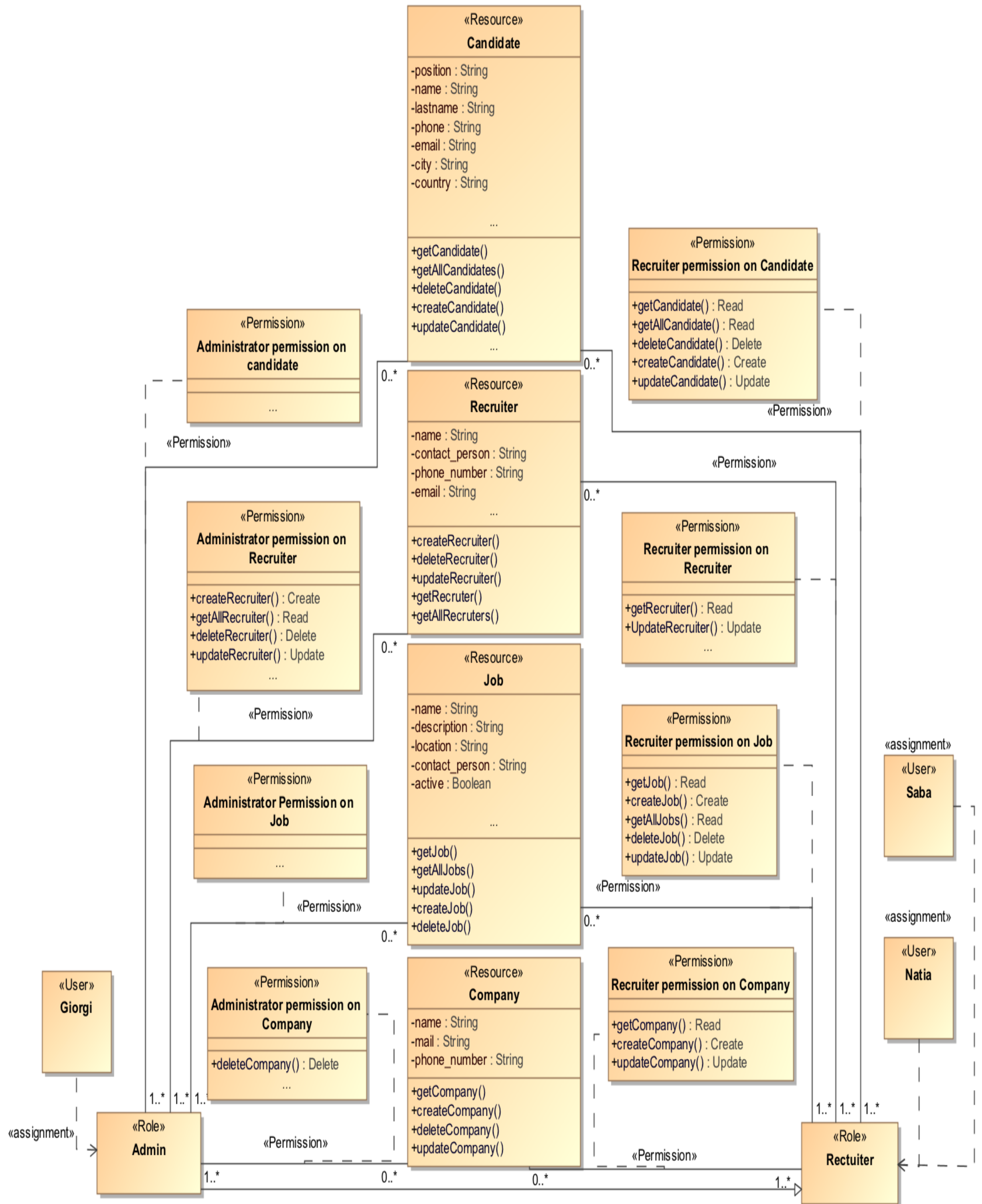

Figure 35 An example of SecureUML model of hierarchical RBAC

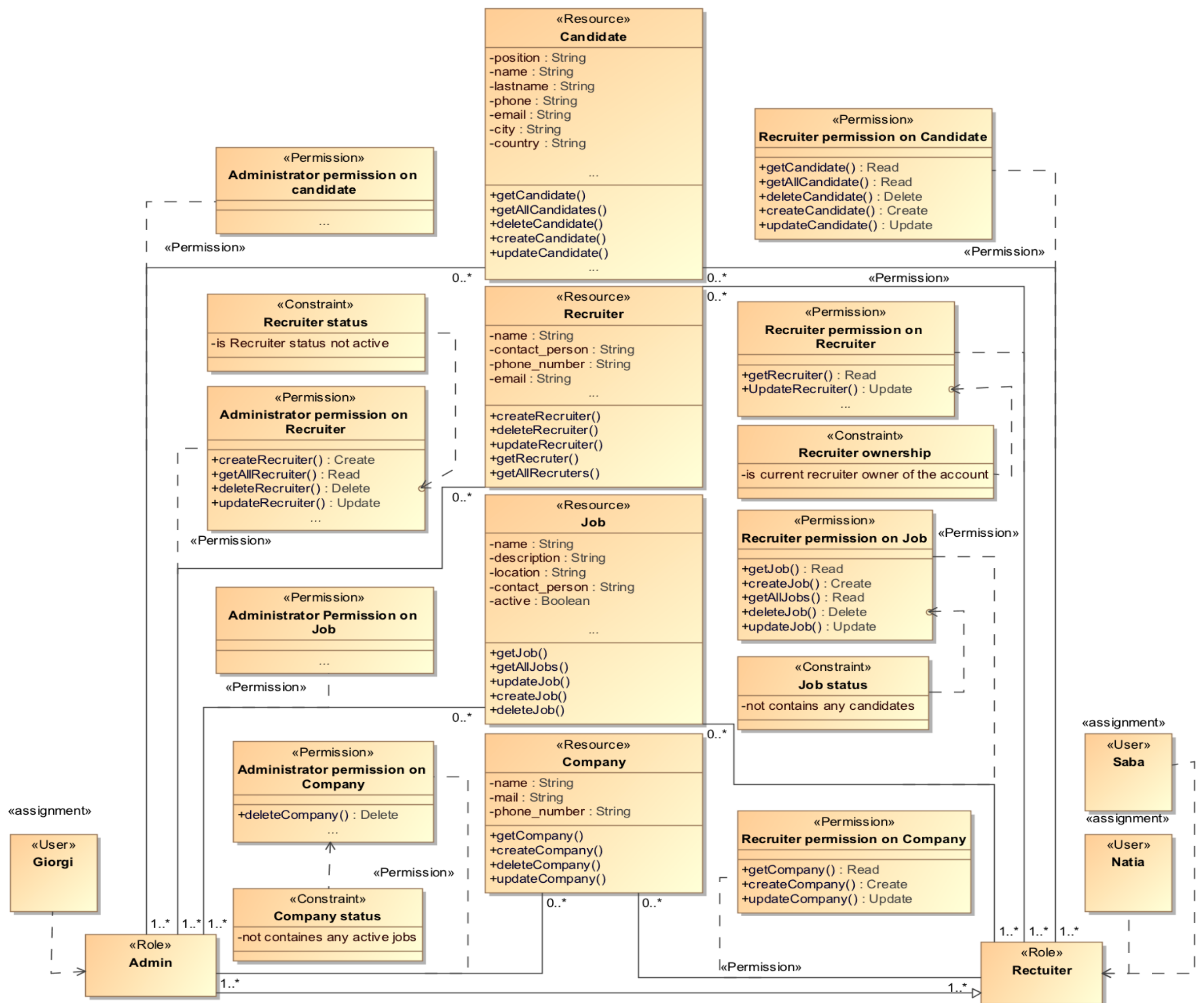

Figure 36 An example of SecureUML for constrained RBAC.

4.2.2 Attribute-Based Access Control(ABAC) implementation

Spring Expression Language(SpEL) and Spring Security

For implementing ABAC we used spring security framework and its expression language SpEL. Spring security framework is very useful when developers want to inject their access control logic in centralized component and enforce it in various places of the application, like before or after REST API call and methods which provide all necessary data for access control logic to work like parameters or objects. On the other hand, SpEL is an expression language which is used by spring framework. It supports querying and manipulating an object on runtime. It is like Java EL[8], which is used in JSP[9] and JSF[10].

Key Components

Our approach is based on the following ideas: we have defined a central repository, a JSON file, which contains the access rules. Each access rule uses Boolean SpEL expression to

---

[8] The EL allows page authors to use simple expressions to dynamically access data from JavaBeans components.

[9] A technology that helps software developers create dynamically generated web pages based on HTML, XML, or other document types.

[10] Java specification for building component-based user interfaces for web applications and was formalized as a standard through the Java Community Process being part of the Java Platform, Enterprise Edition.

defile rules (e.g. subject.id == object.owner.id). Created centralised component which loads the rules from repository, wraps the access context and evaluates rule expressions to grand the access. For enforcing access rules, we have used Spring annotations like *@PostAuthorize* and *@PreAuthorize.* These annotations are used to apply method security and support SpEL expression evaluation. *@PreAuthorize* is suitable for applying security measures before entering method and *@PostAuthorize* check authorization after executing method. Now let's define the key components of the ABAC implementation. The entry point for ABAC logic is *PermissionEvaluator* component. This component delegates all decisions made by spring security annotations like *@PostAuthorize* and *@PreAuthorize*. Figure 37 represents a custom implementation of *PermissionEvaluator*. Main functionality of this code is to the access decision to *PolicyEnforcement* component.

```
16  @Slf4j
17  @Component
18  public class PermissionEvaluatorHandler implements PermissionEvaluator {
19
20    private final PolicyEnforcementFunctionality policy;
21
22    @Autowired
23    public PermissionEvaluatorHandler(PolicyEnforcementFunctionality policy) { this.policy = policy; }
26
27    @Override
28    public boolean hasPermission(Authentication authentication, Object targetDomainObject, Object permission) {
29      Object user = authentication.getPrincipal();
30      Map<String, Object> environment = new HashMap<>();
31      environment.put("time", new Date());
32      return policy.enforce(user, targetDomainObject, permission, environment);
33    }
34
35    @Override
36    public boolean hasPermission(Authentication authentication,
37                                 Serializable targetId,
38                                 String targetType,
39                                 Object permission) {
40      return false;
41    }
42  }
```

Figure 37 Implementation of PermissionEvaluator

Next key component is *ContextAwarePolicyEnforcement*, which is related to *PolicyEvaluator* component, with one major difference, it can be called at any point in the code and it will be filled with authenticated user information. Main use of this component is when the data is needed for making decision when *@PostAuthorize* and *@PreAuthorize* is not available. Figure 38 presents the implementation of *ContextAwarePolicyEnforcement* component. At line 12 we are getting authenticated user object, which means that whenever you use *ContextAwarePolicyEnforcement* component you will always have user data at hand.

```
13
14  @Component
15  public class ContextAwarePolicyEnforcement  {
16
17    protected final PolicyEnforcementFunctionality policy;
18
19    @Autowired
20    public ContextAwarePolicyEnforcement(PolicyEnforcementFunctionality policy) { this.policy = policy; }
23
24    public void checkPermission(Object resource, String permission) {
25      Authentication auth = SecurityContextHolder.getContext().getAuthentication();
26      Map<String, Object> environment = new HashMap<>();
27      environment.put("time", new Date());
28      if(!policy.enforce(auth.getPrincipal(), resource, permission, environment))
29        throw new AccessDeniedException("Access is denied");
30    }
31
32  }
33
```

Figure 38 Implementation of ContextAwarePolicyEnforcement..

Next key component is *PolicyEnforcement.* This is the place where access decision computation is taking place. The enforcement is done by loading all policis using *PolicyDefinition*

component, then filtering applicable policies where target expression is true and finally evaluating existing policies' conditions. If any of them evaluates to true access is granted.

```
11
12  @Slf4j
13  @Component
14  public class PolicyEnforcement implements PolicyEnforcementFunctionality {
15
16    private final PolicyDefinitionFuntionality policyDefinition;
17
18    @Autowired
19    public PolicyEnforcement(@Qualifier("jsonPolicy") PolicyDefinitionFuntionality policyDefinition) {
20      this.policyDefinition = policyDefinition;
21    }
22
23
24    @Override
25    public boolean enforce(Object subject, Object resource, Object action, Object environment) {
26      Collection<Rule> allRules = policyDefinition.getPolicyRules();
27      AccessContextHandler cxt = new AccessContextHandler(subject, resource, action, environment);
28      Collection<Rule> matchedRules = filterRules(allRules, cxt);
29      return checkRules(matchedRules, cxt);
30    }
31
32    private Collection<Rule> filterRules(Collection<Rule> allRules, AccessContextHandler cxt) {
33      Collection<Rule> matchedRules = new ArrayList<>();
34      for (Rule rule : allRules) {
35        try {
36          if (rule.getTarget().getValue(cxt, Boolean.class)) {
37            matchedRules.add(rule);
38          }
39        } catch (EvaluationException ex) {
40        }
41      }
42      return matchedRules;
43    }
44
45    private boolean checkRules(Collection<Rule> matchedRules, AccessContextHandler cxt) {
46      for (Rule rule : matchedRules) {
47        try {
48          if (rule.getCondition().getValue(cxt, Boolean.class)) {
49            return true;
50          }
51        } catch (EvaluationException ex) {
52        }
53      }
54      return false;
55    }
56  }
57
```

Figure 39 Implementation of PolicyEnforcement component.

Figure 39 presents implementation of *PolicyEnforcement* component. In *enforce* method we get all policy rules (line 26), wrap the security context (line 27), filter rules by checking target of the rule (line 28) and finally check if context satisfy any of the filtered rules (line 29). *PolicyDefinition* is an interface representing the repository of the policy. It has method *getPolicyRules,* which retrieves all policies from the repository. The purpose behind implementing *PolicyDefinition* interface was to hide the type of the repository from user, in our case it is JSON file. *Policy* class is the element which needs to be evaluated. It contains two main properties target and condition which both represent SpEL expression. If target is true then rule is applicable and if condition is true then access is granted. Both properties have access to the ABAC elements subject, object, resource, action and environment. *AccessContext* is a wrapper class for ABAC elements. When access decision needs to be taken *PolicyEnforcement* creates the instance of this class and fills it with corresponding data.

Policy repository

For storing policies, we have used in-memory and static JSON file. Below is the list of access rules represented in JSON format. Each rule consists of name, description, target and condition. The first rule says that, unless the user is enabled and the user has "ADMIN"

authority he can do everything in the system. The second policy defines the access of recruiters for company resource. It says that unless the user is enabled and the user only has "RECRUITER" authority user will have access to two actions: get the list of companies and get the detail of single company. The third policy defines permissions of the recruiter for job resource. It says that unless the user is enabled and if the user only has "RECRUITER" authority user will have access to four actions: get the list of jobs and get the detail of single job, update the details of the job and create a new job. Last policy defines permissions of the recruiter for candidate resource. It says that unless the user is enabled and if the user only has "RECRUITER" authority user will have access to all available actions for candidate resource.

Table 8 Policy repository

```
1.  [
2.    {
3.      "name": "Admin",
4.      "description": "Admin can do all.",
5.      "target": "subject.authorities.contains(SimpleGrantedAuthor-
    ity('ROLE_ADMIN'))",
6.      "condition": "subject.enabled == true"
7.    },
8.
9.    {
10.     "name": "Recruiter company permission",
11.     "description": "Recruiter can only list availabke compa-
    nies in the system or view single company detail page",
12.     "target": "subject.authorities.contains(SimpleGrantedAuthor-
    ity('ROLE_RECRUITER')) && {'COMPANIES_GET','COMPANY_GET'}.con-
    tains(action)",
13.     "condition": "subject.enabled == true"
14.   }, {
15.     "name": "Recruiter job permission",
16.     "description": "Recruiter can only get jobs or job, cre-
    ate job, assign candidates and update job status",
17.     "target": "subject.authorities.contains(SimpleGrantedAuthor-
    ity('ROLE_RECRUITER')) && {'JOBS_GET','JOB_GET', 'JOB_UP-
    DATE', JOB_CREATE}.contains(action)",
18.     "condition": "subject.enabled == true"
19.   },
20.   {
21.     "name": "Recruiter candidate permission",
22.     "description": "Recruiter can do all operations against candi-
    date resource",
23.     "target": "subject.authorities.contains(SimpleGrantedAuthor-
    ity('ROLE_RECRUITER')) && {'CANDIDATES_GET','CANDIDATE_GET', 'CAN-
    DIDATE_UPDATE', CANDIDATE_CREATE, CANDIDATE_DELETE}.contains(ac-
    tion)",
24.     "condition": "subject.enabled == true"
25.   }
26. ]
```

4.3 User manual

Login/Logout

Before comparing access models, the user needs to login to his/her account or create new one (figure 40)

Step one: Enter the username, password and click on “LOG IN” button (1) or click on Register (2) for a new user. For registration enter username, first name, last name, password and click on “REGISTER NOW” button (3)

Step two: Verify that your username is visible in the top bar (figure 41). Click on the photo on the right side of the top bar (1), drop down menu will appear where user will see his/her username (2) and “Log out” button (3) to end the session.

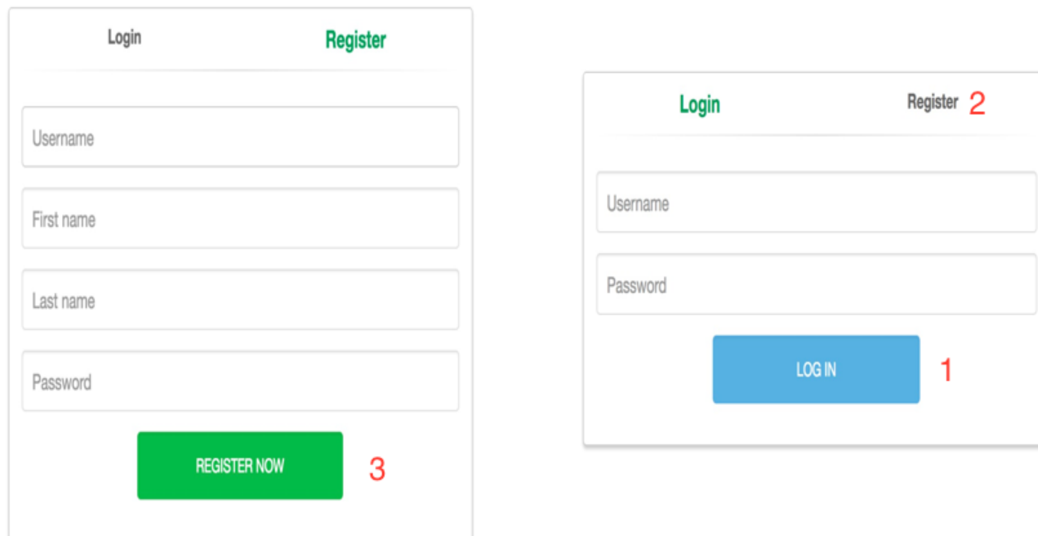

Figure 40 Log in and registration forms.

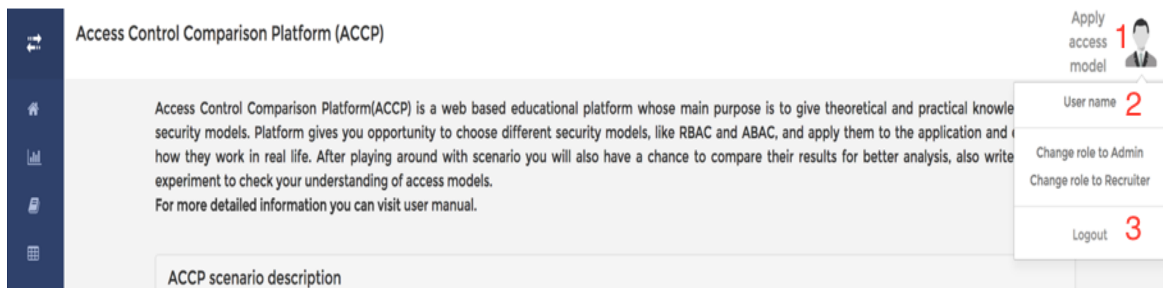

Figure 41 Dropdown menu for user name and logout.

Sidebar menu

Navigation in the system is done from sidebar menu (figure 42). User can toggle menu using (1). From this menu user can browse several pages like introduction page (2), Comparison Results (3), Quiz (4), and Scenario (5).

Introduction page

Introduction (figure 43) page is a first page user will see after logging into the system. In this page, he/she can read small summery of the prototype (6), also description of the scenario (7) and description of supported access models (8).

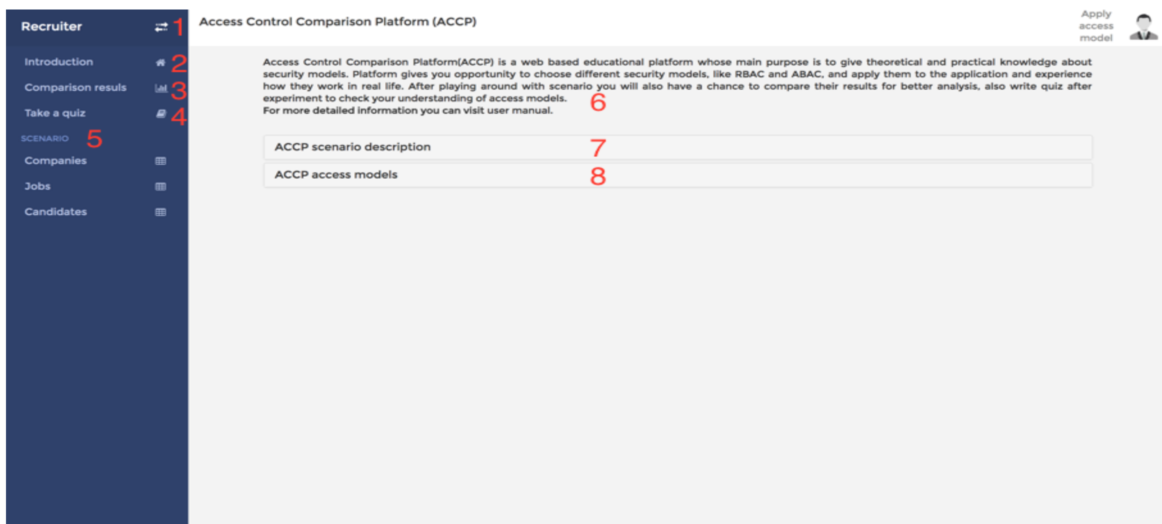

Figure 42 Sidebar menu and introduction page

Comparison results

The following steps describe how to make comparison analysis based on results (figure 43).
Step one: For making comparison, user clicks on Comparison results (1) menu item from sidebar menu.
Step two: A comparison page is opened, containing two tabs: Analytical comparison (2) and Comparison result analyse (3). Analytical comparison tab is opened by default. User can read theoretical comparison analysis of RBAC and ABAC.
Step three: User clicks on Comparison result analyse (3).

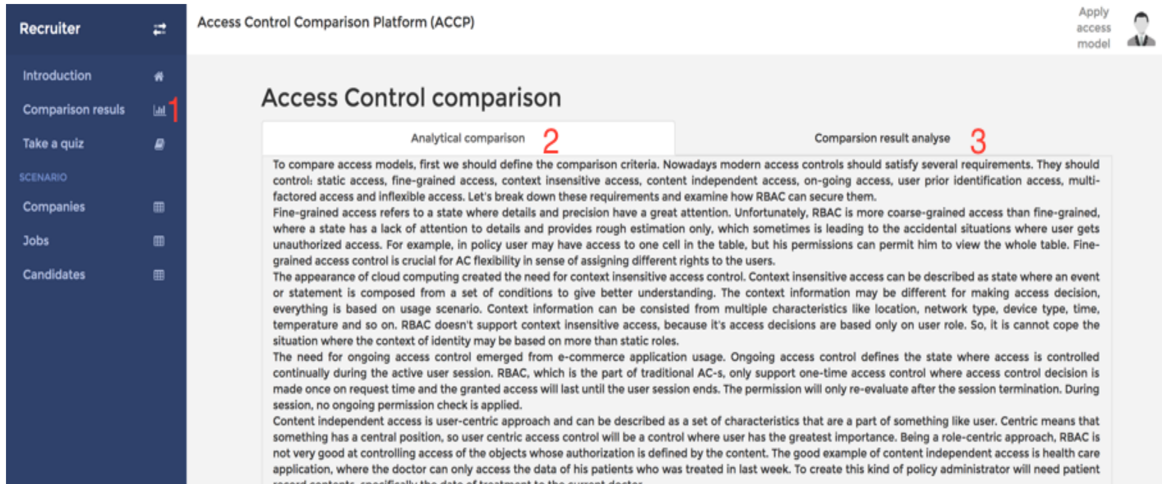

Figure 43 Access control comparison result page.

Change user role

By default, when user is created in the system it has Recruiter role so, changing user role and get new permissions is very important feature of the prototype. Following steps explain how to change roles in the system (figure 44).
Step 1: click on the photo on the right side of the top bar (1)
Step 2: A dropdown menu will appear where user will see “change role to Admin” (2) and “change role to Recruiter” (3).
Step 3: clicking on each item will change roles accordingly.

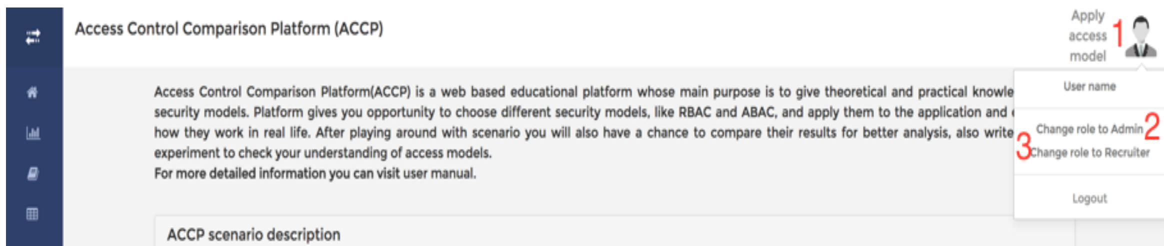

Figure 44 Change user role dropdown menu.

Apply access model

At any point of prototype execution user can apply model. Which means that chosen model will process request in the server. The specific steps are presented below (figure 45).
Step 1: Click on the "Apply access model" button in the top bar (1).
Step 2: A dropdown menu will appear where user will see "Apply RBAC" (2) and "Apply ABAC" (3) menu items.
Step 3: Click on each item will apply models accordingly.

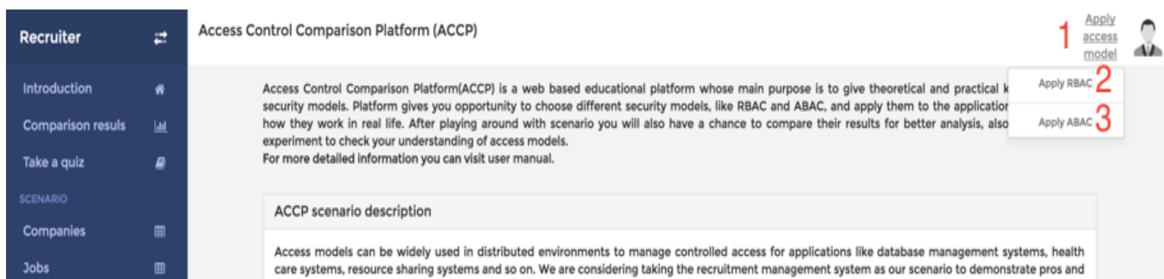

Figure 45 Apply access model dropdown menu

Scenario-companies

Clicking on the "Companies" menu item from sidebar menu will get company list (1). Each company item (2) has a link to its details, "Delete" button (3) and "Create new" button (4). Click on the "Delete" button will remove company from the list. Clicking on "Create new" button will open new company creation form (5). For creating new company user will enter name, address, city, country, email address, phone and press "Submit" button (6). Clicking on the company item will open company details page. Where user will see company details (7), update button (8) and assigned job list (9). Click on the update button will make company detail fields editable. (Figure 46).

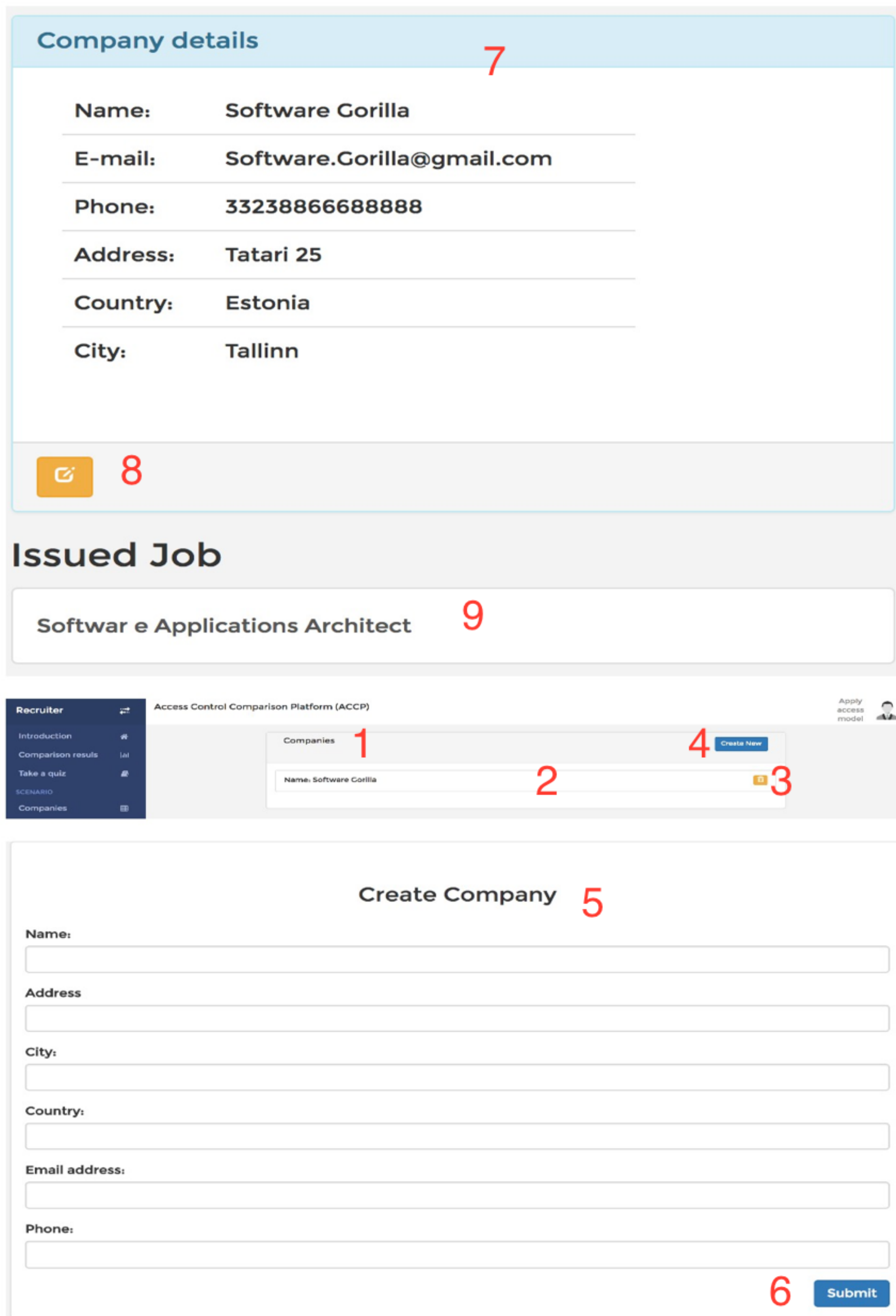

Figure 46 Company detail view, list view and create view

Scenario-Jobs

Clicking on the “Jobs” menu item from sidebar menu will get job list (1). Each job item (2) has a link to its own details, “Delete” (3) and “Create new” button (4). Click on the “Delete”

button will remove job from the list. Clicking on “Create new” button will open new job creation form (5). For creating new job user will enter title, issuer company, which he/she will choose from existing companies via dropdown list, description and press “Submit” button (6). Clicking on the job item will open job details page. Where user will see job details (7), update button (8) assigned candidate list (9) and issuer company (10). Click on the update button will make company detail fields editable. (Figure 47).

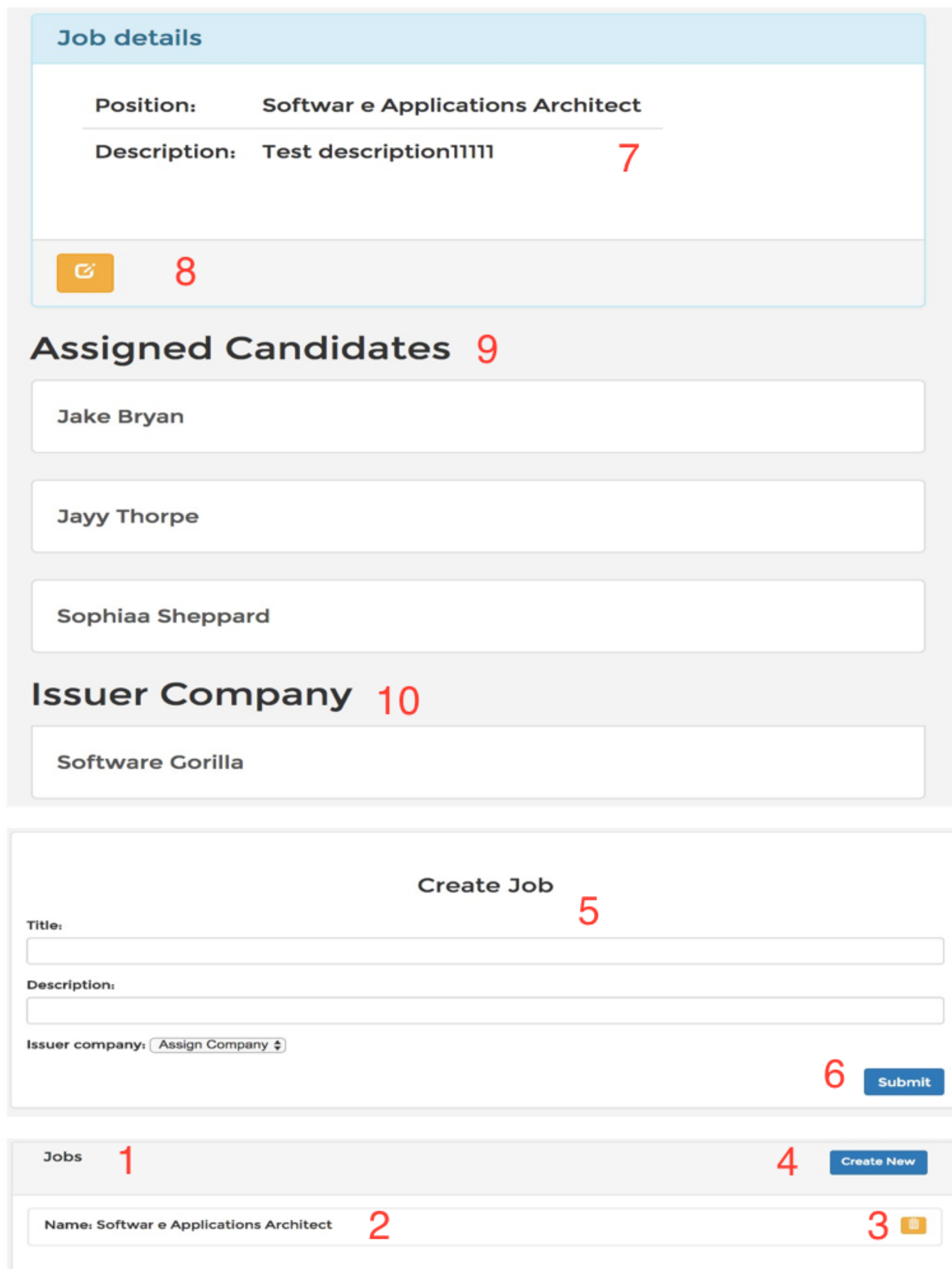

Figure 47 Job detail view; list view; create view;

Scenario-Candidates

Clicking on the “Candidates” menu item from sidebar menu will get candidate list (1). Each candidate item (2) has a link to its own details, “Delete” (3) and “Create new” button (4). Click on the “Delete” button will remove job from the list. Clicking on “Create new” button will open new candidate creation form (5). For creating new candidate user will enter first

name, last name, address, city, country email, position, which he/she will choose from existing jobs via dropdown menu, phone and press "Submit" button (6). Clicking on the candidate item will open candidate details page (7). Where user will see candidate detail, update button (8) and job list when candidate is assigned on (9). Click on the update button will make company detail fields editable. (Figure 48).

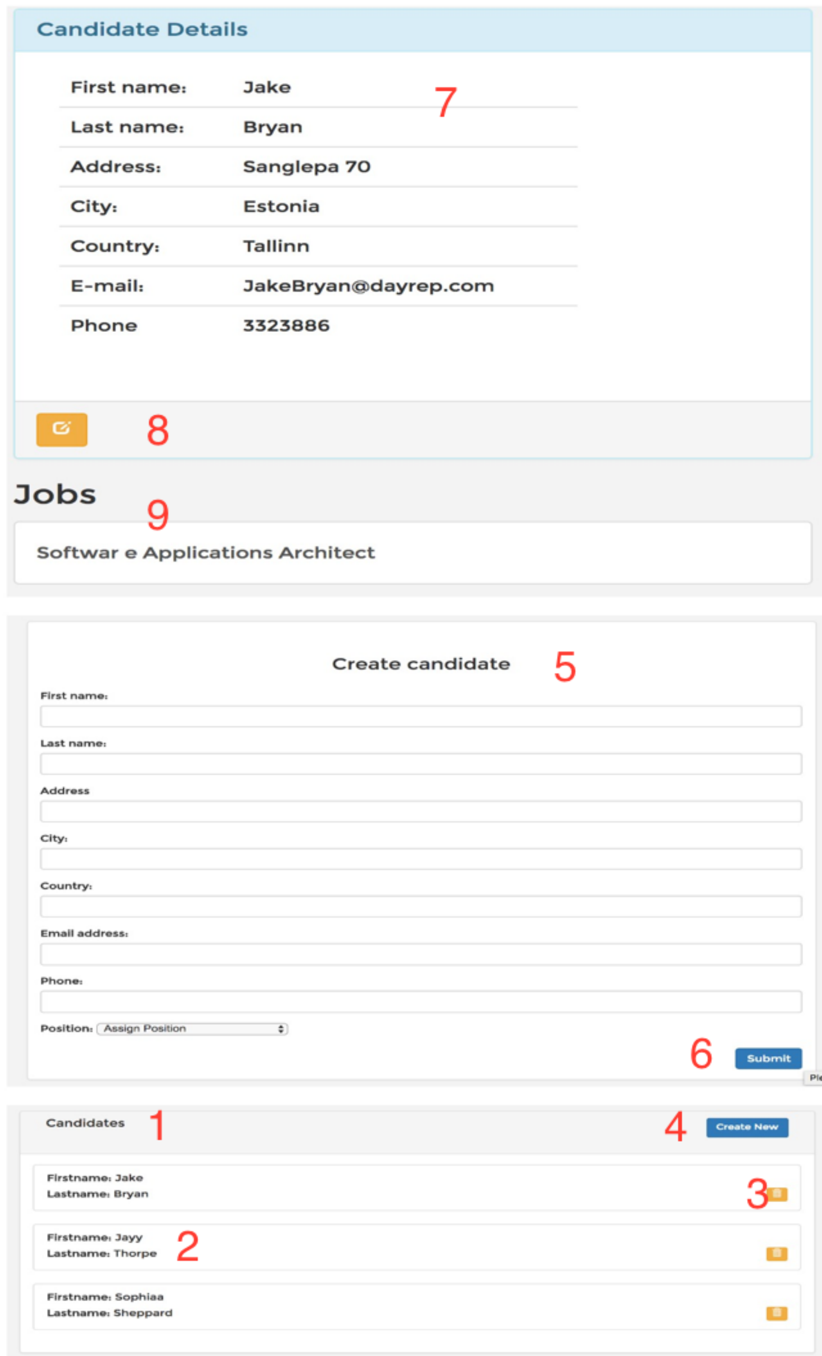

Figure 48 Candidate detail view, list view and create view.

Manage user quiz results

For the actions presented in this feature, the user must be logged in, and he/she must have super admin rights. Super admin only has one view, where he/she can see the list of people (1) who took a test and their test results. Each row of table contains person's name, last name and the test result. User can filter list by person's first name and last name using search

input (1). At any point of application execution user can reset the list of quiz results and start over by clicking “Reset results” button (3). (figure 49)

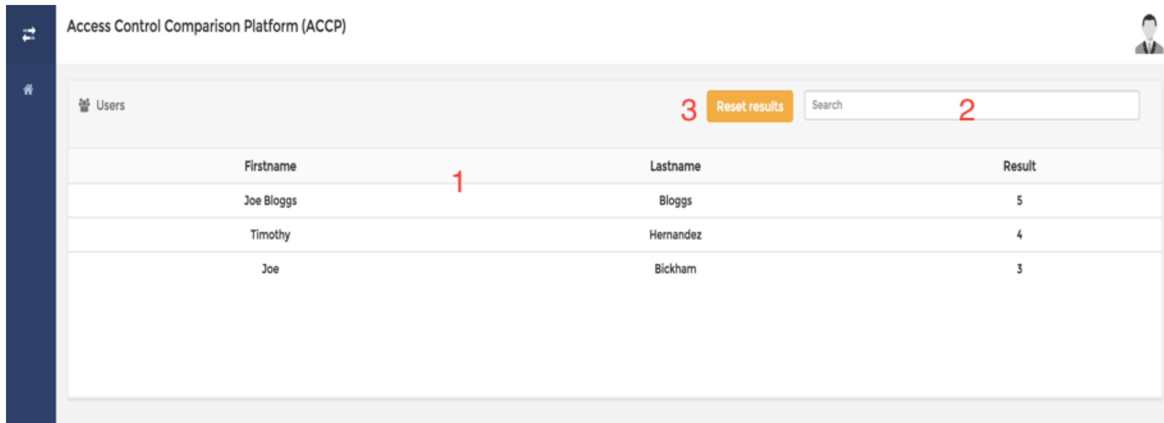

Figure 49 Quiz result management system.

4.4 Summary

In this chapter, we concentrated on defining a description, implementation and user manual of the prototype. First, we introduce the perspective of the application and described scenario. After that we started defining possible use cases and based on this outcome we created the requirements. Next step was to illustrate the implementation of the prototype, where we presented RBAC and ABAC policies and explained codebase. In the end, we provided step-by-step user manual. In the next chapter, we will conclude our thesis.

5 Conclusion

In this paper, we developed prototype application for analyzing Role-Based and Attribute-Based Access Control models. We divided the prototype creation process in five steps. First step was to define the access control policies for each model. In order to accomplish this, we had to understand how these models work. thus, we provided an overview of Role-Based and Attribute-based Access Control models, where we described the main concepts and meta-model of each model. As a next step was analytical comparison of ABAC and RBAC. We defined the requirements that should be satisfied by that modern access control models and analyzed how each model applies security in each case. Only After that we started defining prototype. First of all, we described overall structure, then we continued with defining use cases and features of this product which gave us a picture of how prototype should look like. Next, we started implementing those features and illustrated the architecture of the prototype. In the end, after finishing prototype we created a user manual of it where is illustrated the GUI of the application and explained how it functions with step by step description.

There is not yet a research which addresses the same problem in this field. This means that we created use cases and requirements based on our vision of the problem. On our point of view, the biggest limitation of the prototype is the lack of user input regarding the features and GUI.

Despite the lack of enhancements, this prototype should be served as a starting point for further deeper investigation and creating similar tools.

5.1 Future work

For a future improvement, we would like to make this prototype more dynamic. In current solution, we made static policies. In future releases, we think would be great if we give users possibility to dynamically change the policies and apply them to the scenario. Another great improvement for this prototype is adding more access models like Usage-Based access control. This would make prototype even more challenging and interesting for experiments, as Usage-Based Access Control has a completely different approach compared to RBAC and ABAC

References

[1] Ferraiolo, D., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST standard for role-based access control. ACM Transactions on Information and System Security 4(3), 224–274 (2001)

[2] Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-Based Access Control Models. IEEE Computer 29(2), 38–47 (1996)

[3] Araujo R. and Gupta S., Design Authorization Systems Using SecureUML, Foundstone Professional Services, 2005.

[4] Lodderstedt T., Basin D. and Doser J., "SecureUML: A UML-Based Modeling Language for Model-Driven Security," in UML '02 Proceedings of the 5th International Conference on The Unified Modeling Language, Dresden, 2002.

[5] Goyal V., Pandey O., Sahai A., Waters B., "Attribute-based encryption for fine-grained access control of encrypted data", Proc. 13th ACM Conf. Comput. Commun. Security, pp. 89-98, 2006.

[6] "Best Practices in Enterprise Authorization: The RBAC/ABAC Hybrid Approach", white paper EmpowerID, 2013, [online] Available: http://blog.empowerid.com/Portals/174819/docs/EmpowerID-WhitePaper-RBAC-ABAC-Hybrid-Model.pdf.

[7] Yuan, E., Tong, J.: Attributed based access control (ABAC) for Web services. ICWS 2005 IEEE International Conference on Web Services (2005)

[8] V.C. Hu et al., Guide to Attribute Based Access Control (ABAC) Definition and Considerations, NIST Special Publication 800-162, Nat'l Institute of Standards and Technology, Jan. 2014; http://nvlpubs.nist.gov/nistpubs /specialpublica tions/NIST.sp.800 -162.pdf.

[9] Haibo S, Fan H. A context-aware role-based access control model for web services. E-Business Engineering, 2005. ICEBE 2005. IEEE International Conference on, IEEE: Beijing, China, 2005; 220–223.

[10] Covington MJ, Fogla P, Zhan Z, Ahamad M. A context-aware security architecture for emerging applications. Computer Security Applications Conference, 2002. Proceedings. 18th Annual, IEEE: Las Vegas, NV, USA, 2002; 249–258.

[11] Zhang G, Parashar M. Dynamic context-aware access control for grid applications. Proceedings of the Fourth International Workshop on Grid Computing, 2003, IEEE: Phoenix, AZ, USA, 2003; 101–108.

[12] Hansen F, Oleshchuk V. Srbac: a spatial role-based access control model for mobile systems. Proceedings of the 7th Nordic Workshop on Secure it Systems (NORDSEC03), Citeseer: Gjovik, Norway, 2003; 129–141.

[13] Damiani ML, Bertino E, Catania B, Perlasca P. Geo-rbac: a spatially aware rbac. ACM Transactions on Information and System Security (TISSEC) 2007; 10(1): 2. CrossRef | Web of Science® Times Cited: 30.

[14] Toahchoodee M, Ray I, Anastasakis K, Georg G, Bordbar B. Ensuring spatio-temporal access control for real-world applications. Proceedings of the 14th ACM Symposium on Access Control Models and Technologies, ACM: Stresa, Italy, 2009; 13–22.

[15] Ray I, Toahchoodee M. 2008. A spatio-temporal access control model supporting delegation for pervasive computing applications. In Trust, Privacy and Security in Digital BusinessSpringer: Turin, Italy; 48–58.

[16] Kim YG, Mon CJ, Jeong D, Lee JO, Song CY, Baik DK. 2005. Context-aware access control mechanism for ubiquitous applications. In Advances in Web IntelligenceSpringer: Lodz, Poland; 236–242.

[17] Bertino E, Bonatti PA, Ferrari E. TRBAC: A temporal role-based access control model. ACM Transactions on Information and System Security (TISSEC) 2001; 4(3): 191–233.

[18] Shafiq B, Samuel A, Ghafoor H. A gtrbac based system for dynamic workflow composition and management. Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005. Eighth IEEE International Symposium on, IEEE: Seattle, WA, USA, 2005; 284–290.

[19] Jin X, Sandhu R, Krishnan R.Rabac: role-centric attribute-based access control. In Computer Network Security Springer: Petersburg, Russia 2012; 84–96.

[20] Wang W. Team-and-role-based organizational context and access control for cooperative hypermedia environments. Proceedings of the Tenth ACM Conference on Hypertext and Hypermedia: Returning to our Diverse Roots: Returning to our Diverse Roots, ACM: Darmstadt, Germany, 1999; 37–46.

[21] Thomas RK. Team-based access control (TMAC): a primitive for applying role-based access controls in collaborative environments. Proceedings of the Second ACM Workshop on Role-Based Access Control, ACM: Virginia, USA, 1997; 13–19.

[22] Fatima A., Ghazi Y. Towards Attribute-Centric Access Control: an ABAC versus RBAC argument. 6 July 2016.
http://onlinelbrary.wiley.com/doi/10.1002/sec.1520/full

Appendix

I. List of Acronyms

ACM	Access Control Mechanism
ABAC	Attribute-Based Access Model
ACCP	Access Control Comparison Prototype
AMF	Assurance Management Framework
AC	Access Control
ACL	Access Control List
ACE	Access Control Entries
API	Application program interface
CRUD	Create, Read, Update, Delete
CH	Context Handler
DAC	Discretionary Access Control
XACML	eXtensible Access Control Mark-up Language
UML	Unified Modelling Language
GUI	Graphical User Interface
OCL	Object Constraint Language
NPE	Non-Person Entity
PEP	Policy Enforcement Mechanism
PDP	Policy Decision Point
PIP	Policy Information Point
PAP	Policy Administration Point
XML	eXtensible Mark-up Language
RBAC	Role-Based Access Model
RB-RBAC	Rule-Based RBAC
MAC	Mandatory Access Control
TBAC	Team-Based Access Model
JSON	JavaScript Object Notation
MVC	Model–View–Controller
REST	Representational State Transfer
JAVA EL	Expression Language
JSP	Java Server Pages
JSF	Java Server Faces
SpEL	Spring Expression Language

II. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Lasha Tsintsabadze**,
(*author's name*)

1. herewith grant the University of Tartu a free permit (non-exclusive license) to:
   1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and
   1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis
A Prototype to Analyze Role- and Attribute-Based Access Control Models,
(title of thesis)
supervised by Raimundas Matulevicius,
(supervisor's name)

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive license does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **14.08.2017**