

Sheridan College

SOURCE: Sheridan Institutional Repository

Student Capstones

Honours Bachelor of Computer Science (Mobile Computing)

Fall 12-6-2022

SWIFTLY

Arjun Suthaharan

Sheridan College, suthaarj@sheridancollege.ca

Madhavan Jaisankar

Sheridan College, jaisanka@sheridancollege.ca

Tobias Moktar

Sheridan College, moktart@sheridancollege.ca

Follow this and additional works at: https://source.sheridancollege.ca/fast_sw_mobile_computing_capstones

Recommended Citation

Suthaharan, Arjun; Jaisankar, Madhavan; and Moktar, Tobias, "SWIFTLY" (2022). *Student Capstones*. 9. https://source.sheridancollege.ca/fast_sw_mobile_computing_capstones/9

This Capstone Open Access is brought to you for free and open access by the Honours Bachelor of Computer Science (Mobile Computing) at SOURCE: Sheridan Institutional Repository. It has been accepted for inclusion in Student Capstones by an authorized administrator of SOURCE: Sheridan Institutional Repository. For more information, please contact source@sheridancollege.ca.



SWIFTLY

EDUCATION | CAPSTONE PROJECT
HONOURS BACHELOR OF COMPUTER SCIENCE
(MOBILE COMPUTING)

STUDENT TEAM

Tobias Moktar, 4th Year Student

E: moktart@sheridancollege.ca

Arjun Suthaharan, 4th Year Student

E: suthaarj@sheridancollege.ca

Madhav Jaisankar, 4th Year Student

E: jaisanka@sheridancollege.ca

SUPERVISOR

Prof. Jigisha Patel

E: jigisha.patel2@sheridancollege.ca

T: 905-845-9430 ext: 2065

Sheridan College

DOMAIN EXPERTS

Marc Fillion,

E: marc.fillion@ugdsb.on.ca

T: 519-341-4331

Community / Industry Partner

ABSTRACT

The computer science courses offered in elementary and secondary education teaches students the fundamentals of computer programming through theoretical understanding and basic application. The information learned in these courses is very useful, however, the issue at hand is that the process of teaching programming and application development fundamentals is tedious and frustrating to students without prior experience, and teachers are lacking the tools to make the teaching process interesting and engaging to them. The proposed solution will aim to solve this issue with a mobile-based application through which students can learn about the process of programming and developing software applications by means of theoretical and interactive methods. The proposed solution will incorporate a wide variety of computer science technologies like universal account progress, content creation tools, data analytics, and cognitive computing.

ABOUT CAPSTONE PROJECTS

TIMELINES • PROGRAM • SCHOOL

- **September 2021 – December 2021:** [Capstone Project Inception](#), 4-credit course (18 hours / week)
- **September 2022 – December 2022:** Capstone Project, 4-credit course (18 hours / week)

PROGRAM • SCHOOL

- [Hons. Bachelor of Appl. Computer Science \(Mobile Computing\)](#)
- [Applied Computing, Faculty of Applied Science and Technology](#)

Table of Contents

Introduction	4
Project Overview.....	4
Domain and Industry Overview.....	5
Problem Description.....	5
Solution Description	6
Mobile Computing.....	7
Cloud Computing.....	7
Advanced Areas of Computer Science	7
Solution Impact	8
Solution Feasibility	8
Design and Construction	8
Deployment.....	9
Adoption.....	9
Project Requirements.....	10
System Context.....	10
Use-Cases	11
User interface	12
Project Architecture	28
Architecture Overview	28
System Components.....	30
Deployment Model.....	31
Project Plan.....	32
Iteration Plan.....	33
Risk Management Plan.....	34
Validation and Testing.....	35
Testing Strategy.....	35
Validation Results.....	35

Conclusion 35

- Project Suitability 36
- Domain Expert Evaluation 36
- User Testimonials 37
- Future Work 37

Bibliography 39

INTRODUCTION

The purpose of this document is to cover all topics and information pertaining to this project. This document is structured to first provide an overview of the project, then it will discuss the requirements and architecture, following this, the project plan and validation and testing will be discussed, and finally, the document will finish with a conclusion followed by a bibliography.

Version 0.1 was completed on September 14, 2021 and contains all available information about the project.

Version 0.2 was completed on September 25, 2021 and describes the project plan for Swiftly.

Version 0.3 was completed on November 20, 2021 and describe all prior information with the addition of the project architecture.

Version 0.4 was completed on December 11, 2021 and is the final and completed version of the document for the Fall 2021 semester.

Version 0.5 was completed on September 16th, 2022 and is the first revision of the Fall 2022 semester.

Version 0.6 was completed on September 23th, 2022 and is the second revision of the Fall 2022 semester.

Version 0.7 was completed on October 15th, 2022 and is the third revision and alpha release of the Fall 2022 semester.

Version 0.8 was completed on November 26th, 2022 and is the fourth revision and beta release of the Fall 2022 semester.

Version 1.0 was completed on December 4th, 2022 and is the final revision of the Capstone Project Report.

PROJECT OVERVIEW

The project described in this document is called Swiftly. Swiftly is a mobile and web-based application that serves two different purposes depending on the platform. The mobile application is designated to work in conjunction with teachers and their iOS class curriculums. The Swiftly website is solely used for content creators to create new chapters or edit existing ones. The team behind this project consists of three students and one supervisor. The students on the team are Arjun Suthaharan, Madhav Jaisankar, and Tobias Moktar; and the supervisor for this team is Professor Jigisha Patel. In addition to this, Swiftly has a domain expert whose name is Marc Fillion - he is a computer science teacher at Erin District High School.

The project overview section of this document contains vital information surrounding Swiftly and its purpose. The first subsection of the project overview is the domain and industry overview, where a discussion will be held regarding the domain of the problem and the industry where our solution will reside. Following this is the problem description subsection. Here, the problem identified, and the proposed solution will be discussed, and their

components analyzed. In addition to this, this section will contain supporting evidence for the problem statement. The next subsection is the solution description; here, an overview of the solution and how it can solve the problem identified is discussed. This subsection will have additional subsections that dissect the solution on a technological basis where each core component is discussed in-depth. The next subsection is the solution impact subsection. This subsection will discuss the potential impact that the solution will have on certain areas of applicability. The final subsection of the project overview is the solution feasibility subsection. This subsection contains three subsections, where the design and construction, deployment, and adoption of the project are all discussed in-depth to fully analyze the feasibility of the project its projected life span.

DOMAIN AND INDUSTRY OVERVIEW

The domain that exhibits this problem is elementary and secondary education schools. This domain is identified under the Canadian Industry Statistics website, where it's listed with the NAICS code 61111 - a subsection of elementary and secondary educations schools (6111). This industry consists of all education establishments (excluding homeschooling, private tutoring, and pre-kindergarten), from kindergarten through 12th grade. According to the Canadian Industry Statistics website, there are several education establishments that are in this industry, and they are boarding schools, elementary schools, high schools, kindergartens, private schools, school boards, and schools for the physically handicapped. According to the Canadian Industry Statistics website, there are 2671 establishments, with an average revenue of \$647.5 thousand - giving the industry a profit margin of 76.6%.

PROBLEM DESCRIPTION

As smart devices continue to grow as an essential tool in our everyday lives, there has been a growth in the desire to create and develop applications. Children in Canada are showing great interest in creating and designing their own applications, inspired by the ones they grew up using. Currently, if a student from elementary to secondary school wanted to learn to program outside of school, there exist services and applications that teach the fundamentals, including *Coding for Kids: Learn to Code*, *Hopster Coding Safari for Kids*, *Hopscotch – Programming for kids* and *Swift Playgrounds* offered by Apple themselves. The problem arises when trying to take the next step and realizing there is significantly more involved in application development aside from the basics of programming – this only gets worse when realizing there is no service that can effectively teach such things to a younger demographic. The knowledge for application development is found behind long documentation on websites or video tutorials that already assume a strong fundamental understanding of programming and the tools used for application development. It isn't until secondary school where the classes and resources are finally available for students to learn the full application development process, whereby the process of learning application development may be seen as frustrating and tedious being expected to learn programming fundamentals and application development fundamentals from scratch at the same time. In addition to this, students may be overwhelmed by the amount of information and/or the abstract concepts being taught in these classes, and so there exists a need for an application that will work in conjunction with education level requirements that provides students with a welcoming and engaging approach to further understanding the topics in iOS.

A solution to this unexplored area of teaching app development would be primarily beneficial to the elementary and secondary school boards, which would account for domain 61111 as specified in the previous section, which accounts for 2671 establishments within Canada. It will allow school boards to provide a better roadmap for students who eventually want to create mobile applications by the time they reach high school and college. Earlier years would be spent teaching the fundamentals of programming through the many existing services that are available. This would then transition to this solution, which will teach the fundamentals of planning, designing, and maintaining a basic application, in a controlled and carefully evaluated environment. This solution will also benefit the software companies actively working in app development. Big Tech including Apple, Google, Microsoft, Amazon, and Meta are always looking for quality and passionate software developers to bring the many projects and future technology they're developing to life. In addition to this, providing the correct tools from a young age is crucial for building a large pool of potential employers - which in turn will positively affect the software industry.

The domain expert for this project, who is working as a computer science teacher in high school, acknowledges how currently "there are few resources (if any) that address this combination", referring to the combination of programming language foundations and application development foundations. They also elaborate on how the scaffolded process of learning syntax and programming principles then user interface design systems are not "interesting on its own, so the fundamental problem is engagement". They feel that creating a learning tool that allows users to learn both sides of application development simultaneously "will be well received". This problem statement has come from personal experience as well, as growing up wanting to learn how to build applications meant having to wait until junior and senior year of high school. The lack of available resources in teaching application development during elementary and middle school meant having several lost years of potentially learning about the fundamentals of application development.

SOLUTION DESCRIPTION

The proposed solution to this niche area of teaching iOS development is Swiftly. Swiftly is a native iPadOS application that will run on any iPad device that has an internet connection. The content provided by Swiftly will be modularized into chapters, where each chapter represents an important topic in iOS development. Each chapter will be segmented into a theoretical section and an interactive section. The theoretical section will be further segmented into lessons, where each lesson follows a supporting theme that when combined represents the overall topic of the chapter. The interactive section of the chapter will be more aimed towards a "test your understanding" approach, and so in this section, there will be questions pertaining to the information taught in the theoretical section of this chapter. To provide users with multi-style questions, the questions in the interactive section will be delivered to students either through a drag-and-drop code tile method or through the traditional multiple-choice type of question. This section of the chapter is fundamental in ensuring that students understand what was taught in the theoretical section, all the while familiarizing them with programmatic structure through real code questions. Moving forward, iPadOS is often the first platform people experience smart devices and app-based software on, so Swiftly will be accessible to those who have expressed an interest in it.

The main application will also be supported by Swiftly Expert, a web application hosted on the internet that will be used to create and upload chapter content to the main Swiftly application. This website will allow qualified users to create their own content involving specific aspects of application development, creating theoretical information for the chapter that will then be made available for the users of the main application.

Swiftly and Swiftly Expert will both utilize cloud-based storage throughout the applications. Cognitive computing will be utilized for users to communicate any questions and concerns about the content to the app which will provide useful answers or hints based on what the user is looking for, this will be achieved through a chat-bot, using IBM Watsons services. Data analytics will be collected on the users' scores in the quiz section, completed chapters, and time spent in each chapter; these scores can be shared and compared with other users' scores to see where the user stands. With all these features put together, there is the potential to create a learning tool that will help young students to learn the fundamentals of app development, before moving on to fully featured professional development environments.

Mobile Computing

The solution will require the use of mobile computing, as the problem itself is within the scope of mobile computing. Swiftly will target iPadOS and focus on iPads primarily. Nowadays iPads are children's first experiences with technology and software, so it will be a platform that our primary users will be familiar and comfortable with using. As the name of the solution suggests, the language of focus for the application's design will be Swift, as students will have experienced Swift Playground on their devices and are looking into XCode development, both of which utilize Swift as the primary programming language, creating a smooth transition between the two existing platforms. Likewise, the app itself will be developed using Swift, using SwiftUI for designing the application and XCode as the integrated development environment.

The Swiftly Expert companion tool will be hosted on a website, making it accessible to any internet-connected device with a modern web browser such as Google Chrome, Mozilla Firefox, and Microsoft Edge. The decision to make the content creation tool a web application came from the perspective of accessibility, looking to have as many qualified users to be creating chapter content for Swiftly as possible. This accessibility means that anyone with the knowledge has the capability to create content for Swiftly, through their desktop running Windows or Linux, to their laptop running macOS. The website will be developed using ReactJS for JavaScript functionality, and Tailwind CSS for styling.

Both services will utilize Git through Bitbucket to maintain a remote copy of the applications and to allow for development across all three developers.

Cloud Computing

Swiftly and Swiftly Expert will be utilizing cloud-based storage services to read and write crucial information from the internet remotely. Both services will utilize it to save user information, with Swiftly Expert using it to pull chapter information and push updates and content as needed, and Swiftly using it to track the users progress, to make it simple and effortless for the user to return to the exact section they were working on. Firebase Realtime Database will be utilized, as it offers robust real-time synchronizing through JSON packaged data and provides existing Swift and JavaScript packages to make the implementation process seamless. Firebase hosting is completely free to use within a certain level of usage, which will reduce server costs during early deployment. The cloud database will hold Swiftly user account information for users of the main application, Swiftly Expert account information, for the companion tool website, and the chapter information itself.

Advanced Areas of Computer Science

Swiftly will utilize cognitive computing, in the form of IBM's Watson API. Cognitive computing will be utilized to give users the ability to express any questions about chapter content, problem sets, or the terminology being discussed. Users will be able to type out a question, and the system will cognitively detect what the user is looking for, and provide the correct context and answers needed to answer their inquiry. Swiftly will also be utilizing data analytics, to track which chapters the user has completed, how many students in the Swiftly userbase have completed a chapter, user progressions like questions completed and chapters in progress, and potentially more. Swiftly will use these data analytics and display them to the user in such a way to give them an understanding of their personal progress as well as the progress of other users. These measures will help make Swiftly more exciting to go through, giving users a sense of accomplishment as they work their way through Swiftly's chapters.

SOLUTION IMPACT

The solution would need to have a positive, meaningful impact on the stakeholders in the elementary and secondary education domain 61111 and the Educational Tech (EdTech) industry. It aims to achieve this by providing a fundamental learning experience in an area of app development that is often overlooked in similar alternative solutions. More specifically, it focuses on concepts relating to the process of development and how to apply those development practices in app development.

This way it addresses that significant leap that comes from learning the fundamentals and moving on to the development environment, is by providing a transitional learning phase where users can learn how to apply the fundamentals they learned as a beginner. This key area for this transition usually begins within the computer science classroom setting, and so Swiftly would be applicable to users in this setting where it can be used alongside the course material that is being taught by the teacher. This transitional period can be key since it makes the difference between just learning the syntax and logic behind a programming language and realizing how to apply that knowledge through common development practices in the industry. This can help the user to maintain that initial interest and motivate them to expand their interests and apply to something more substantial. This approach of maintaining a level of interest can facilitate the user's learnings to be utilized in a meaningful way if they choose to make it more than a hobby. The solution can potentially provide significant individual benefits. Young users can cultivate career opportunities with knowledge of robust app development practices and, as a result, it leads to a positive impact on the software industry.

SOLUTION FEASIBILITY

This section of the proposal document focuses on the feasibility of the proposed solution. Three core components will be looked at in-depth and considered in this section. These three components are Design and Construction, Deployment, and Adoption. Each of these components is equally important as they are all fundamental to providing a solution to the identified real-world problem.

Design and Construction

Designing a solution such as this will require a lot of research on how to teach potentially “dry” topics such as software design principles and patterns in a compelling and engaging manner. A significant amount of time needs to be dedicated to ensuring the proposed solution delivers a compelling learning experience.

Having said that, it is only feasible once the research and structuring of the educational content are properly done early in production. The same can be said for any key design choices that we choose early in production. They need to be carefully planned initially otherwise we risk having to make late changes - which can cause significant disruptions or even complete overhauls deep in the middle of development

The **inability to curate a relevant and rich catalog of topics** and **failing to make each lesson an engaging experience** are the two most significant risks associated with this solution. This one is obvious. Regardless of how technically well designed the application is, it would fail the goal of keeping users interested in using the app for its intended purpose.

Aside from those two risks, most of the other risks would fall under the technical aspect of this solution. The most significant technical risks would lead to an unusable product.

Security breaches can compromise personal information. Attack vectors would be client-side. So proper restrictions need to be put in place to control what the user can and cannot access. Server-side attacks would be exceedingly rare when using a reliable service for data storage. Regardless every security option that the service provides would need to be taken advantage of to ensure user data is not compromised.

Deployment

Deployment of the solution can be done through the Apple Appstore, where it will be available for anyone to download for free. But the service that this app provides will be subscription-based. This will take advantage of APIs and frameworks offered by Apple, such as **Storekit**, to facilitate the subscription service offerings.

The app itself will need to follow Apple’s strict guidelines to pass the review. It would need to demonstrate optimal performance, privacy, design standards as listed by Apple’s app review guidelines. Achieving these should be feasible, provided these standards are met.

The services that Swiftly uses in an active subscription would require external APIs to achieve its goal. An on-demand, scalable data storage solution would need to be deployed to handle any varying number of users at any given time. Google’s Firestore is a prominent choice for handling the solution’s data storage needs. Advanced areas of this app

involve cognitive computing and data analytics. These two would require service deployment of specialized services such as IBM's Watson AI.

Adoption

Although the app is available to the average consumer, it will be deployed with the intention of targeting the education sector, particularly middle and secondary schools. These education-specific subscriptions can be purchased to provide access to an unlimited number of users. We believe that a successful deployment of this solution in the education sector can make Swiftly a great learning tool to complement the conventional learning experience they gain in school. However, there are potential hurdles that can come up that can prevent the mass adoption of the solution. Education boards usually have strict regulations and often follow their own curriculum. The educational content would need to be within the interests of the education boards. These interests can also vary wildly between different education boards.

Other factors can also define the feasibility of mass adoption. Primarily, accessibility within the schools. Schools or school boards would need to be willing to subscribe to an educational plan as well as having sufficient equipment to use the app. Not every school has this luxury, but it also will not put immediate pressure on us to ensure a highly scalable solution, to support 1000s of students, right from the beginning. Privacy is another matter that would need to be taken seriously. The solution would need to be trustworthy. Apart from the information required to have an active subscription, the solution would need to ensure that it will access and use data that is only used for the purposes of educating students, most of whom are under 18.

As for individual users, subscriptions would need to be affordable and flexible. A monthly or annual subscription is what this solution aims to provide. Pricing, however, is something that can't be confirmed at the time, but it can be expected to be in the \$4 to \$6 / month range.

PROJECT REQUIREMENTS

This section of the document will cover Swiftly's most important stakeholders, its capabilities through its major use-cases and the user interface users will be interacting with. The first subsection will cover the project stakeholder through its primary, supporting, and offstage actors. The VPository can be found at the following link: <https://online.visual-paradigm.com/w/dbokpyzr/drive/#diagramlist:proj=3&open>. The "Use-cases" subsection will go over Swiftly's major use cases and provide a general overview of the applications capabilities. Finally, the user interface section will provide a look at Swiftly's user interface and its elements, going over how information is presented and how users will interact with it.

SYSTEM CONTEXT

The high-level system context for Swiftly is based around its area of application and domain. Swiftly is meant to be used as a teaching aid or as a self-learning tool – therefore its main area of application is within the education sector. The stakeholders for this project are easily identified. Swiftly has several actors that are a necessity to the project and allow for the application to work as intended. Swiftly has two primary actors, and they are the student, and content

creator. The student actor is the actor that will use Swiftly for the sole purpose of learning, and they will do so through direct interaction with the iPad app. Lastly, the content creator actor will interact directly with the Swiftly website where they are able to create new content that will be provided to the students. In addition to these primary actors, Swiftly also has one supporting actor. This supporting actor is Google’s Firebase service. Firebase supports the system in the sense that it provides a remote database that will store Swiftly’s content and users. Finally, Swiftly has one off-stage actor and it is in the software industry. This actor does not interact with the system directly, but it has an interest in it through the potential of early-talent acquisition. Swiftly has three main top-level use cases. The first top-level use case is user account management. This use case’s focal point is about user account creation and account management. This is key in ensuring that the user can have a place to store their progress and account information, allowing them to access the app on any iPad device. The next use case is accessing chapter content. This use case is all about the user’s ability to retrieve chapters from the remote database and being able to work through them. This use case works in conjunction with the first one as they both involve saving and updating user progress remotely. The final use case is the use case of creating new content. This is only applicable to domain experts and involves the process of content creation through the Swiftly website. Content creation in Swiftly involves creating “Chapters”. Each chapter can hold a collection of lessons relevant to that chapter’s subject. The creation use case phase also includes the editing and deleting existing chapters/lessons.

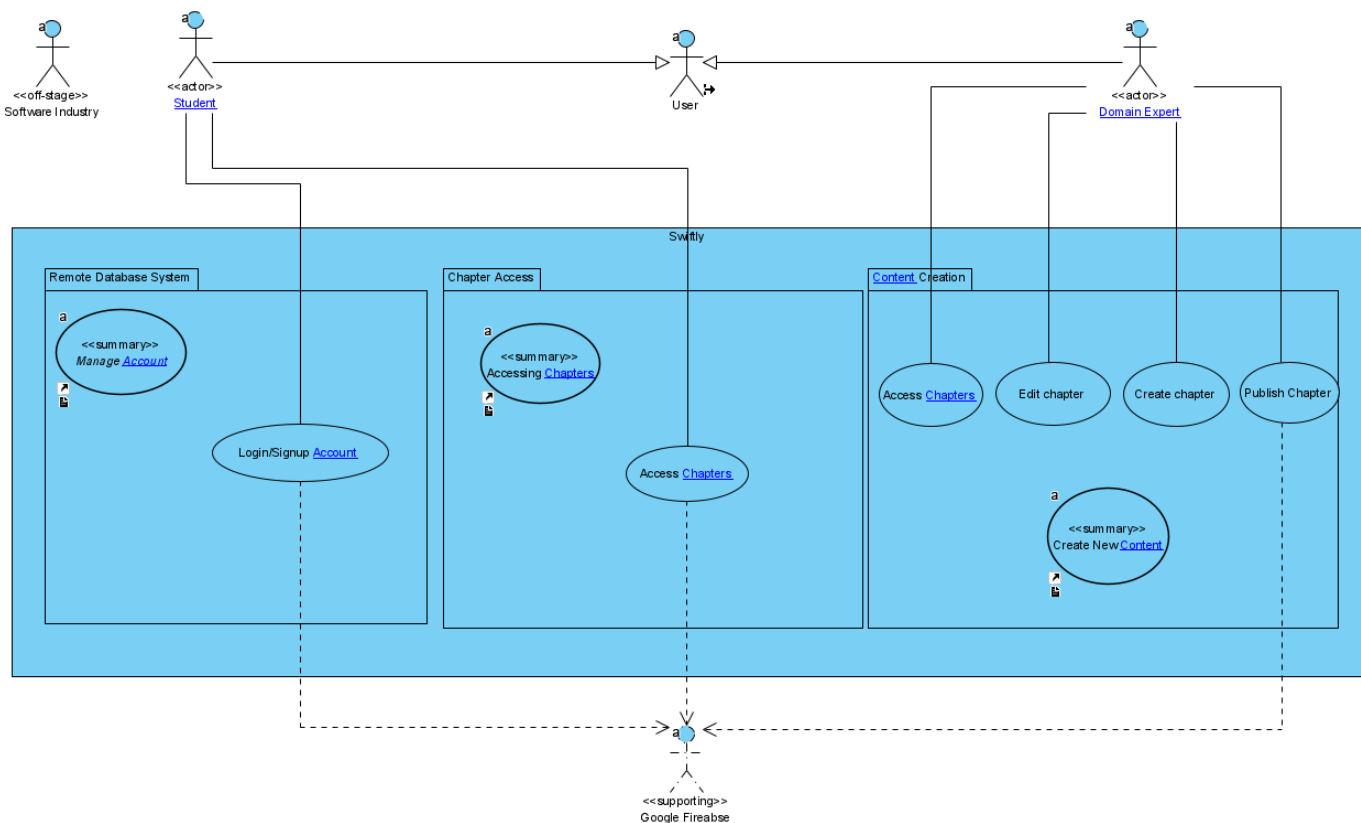


Figure 1: System Context Diagram

USE-CASES

Swiftly has five major use-cases, including the three previously mentioned. The first use-case is the chapter learning use-case. This use-case pertains to the access of chapter content and the comprehensive lesson plan that students will experience while using the app. The second use-case is the testing understanding use-case. This use-case works in conjunction with the previous one as it pertains to the interactive section of each chapter where the user can test their understanding of the topics taught in the chapter’s lesson section. The third use-case is the universal account progress use-case. This use-case is primarily concerned with storing user progress on a remote database so that no matter what iPad device they access the Swiftly on, their progress is not lost. The fourth use-case is the managing user profile use-case. This one works in conjunction with the previous use-case as it is concerned with user account creation and user account management. The previous use-case depends on this one greatly as user progress is tied to the associated user account. The final major use-case is the content creation use-case. This use-case is only applicable to the content creators who will use the Swiftly website to create new chapters and lessons that students will have access to in the app.

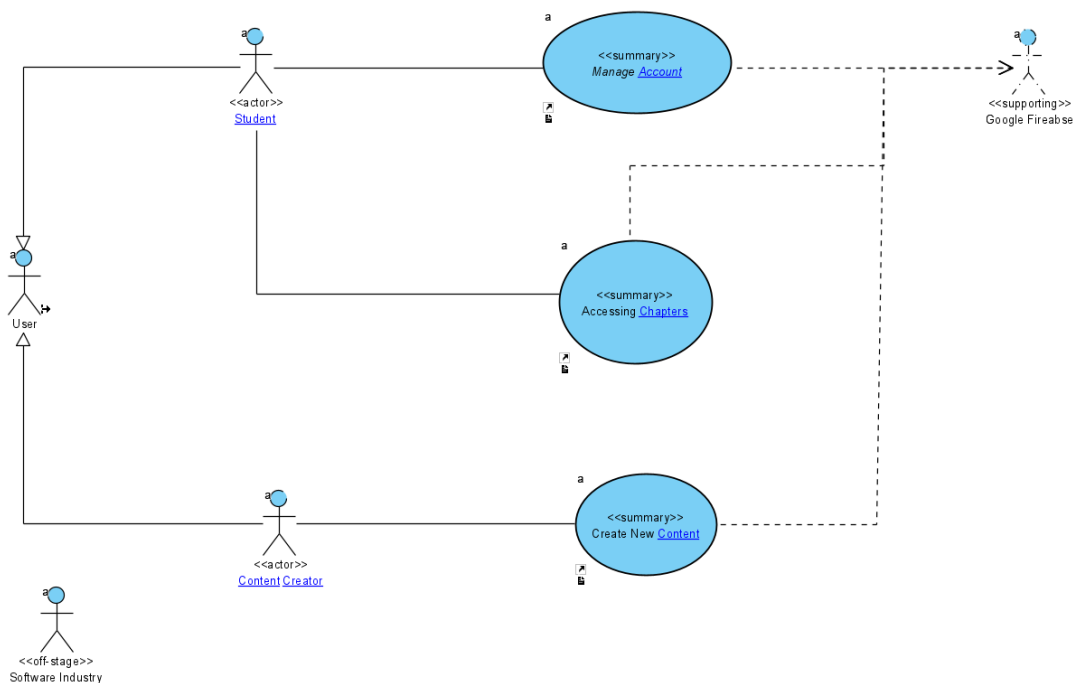


Figure 2: System Overview Diagram

USER INTERFACE

Swiftly App

Swiftly has several major use-cases and so, through a direct correlation, its user interface is diverse and incorporates these use-cases. The original wireframe for Swiftly was created based on a single-color scheme, whereas the actual interface for Swiftly was created to incorporate multiple color schemes (dark mode and light mode). The image on the left will represent the original wireframe design, and the images on the right will represent the actual design in Swiftly.

Login View

The first major user-interface in Swiftly is the login screen. Here the user can login into their pre-existing account which will let them access the app’s content, or they can proceed to the sign-up view if they do not have an account. The actual designed in Swiftly is like its wireframe counterpart – however it includes the eye indicator allowing for users to see their input password. If the user’s email and/or password is incorrect they will receive a pop-up view informing them. If the user is logged in successfully, they will receive a progress indicator informing them of the background process being done to set up the Swiftly environment – after which they are presented to the chapter's view.



Figure 3: Wireframe Login View

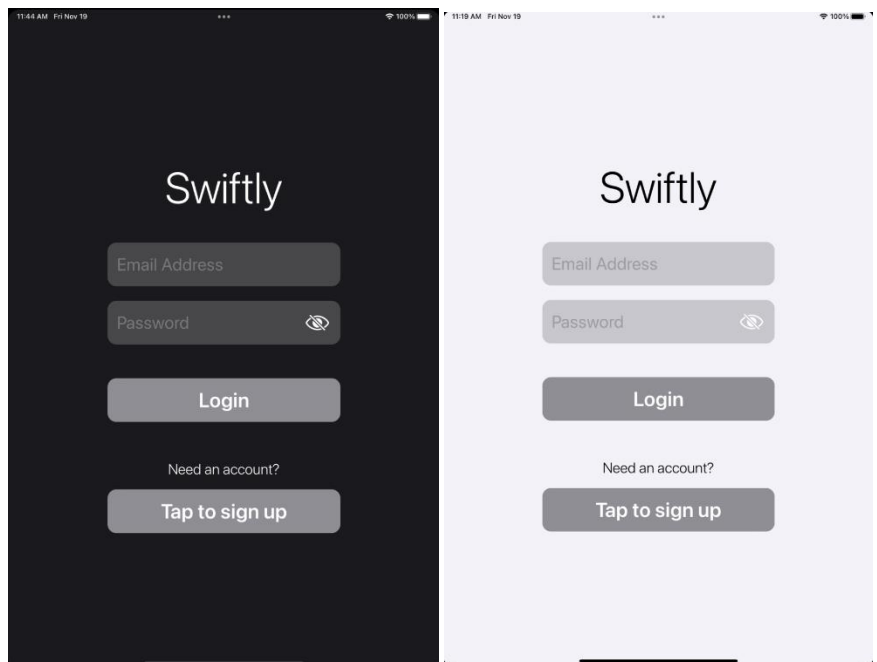


Figure 4: Swiftly Login View Dark and Light Themes

Password Recovery View

A user-interface view tied to the login view. This is where users will be able to request a password reset if they have forgotten the login credentials to their account. The user inputs their email, which Swiftly then checks to see if it is

tied to an existing account, and if it does, sends an email to the users email with the recovery link. If not, the app lets the user know that the account does not exist with the email input.

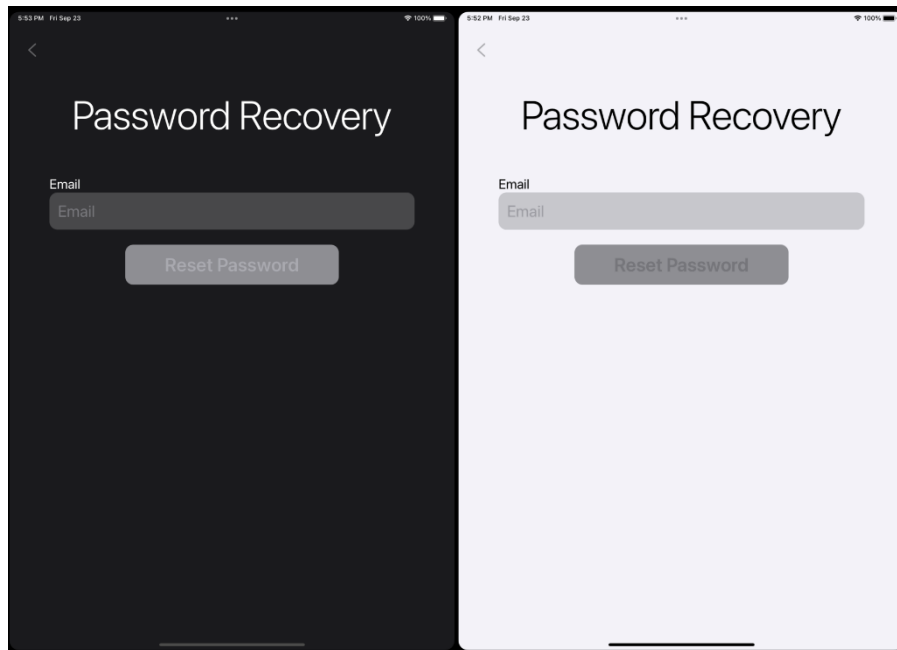


Figure 5: Swiftly Password Recovery View

Sign-Up View

The second major user-interface view is the sign-up view. Here users can sign up for an account by inputting all the appropriate information. Once the user has signed up for an account, Swiftly will pre-populate their account progress with all the default progress for each chapter in the Swiftly database. This gives users a clean slate of progress allowing them to start using the app. This page has many conditional checks to ensure that all the information that the user has inputted is in the correct format, and that the username and email has not been used already.

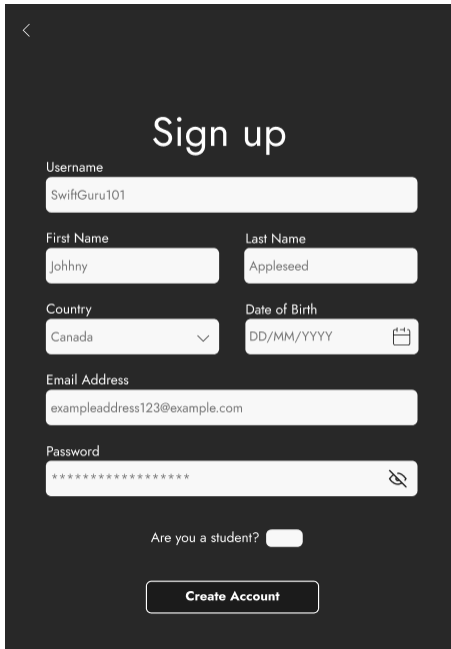


Figure 6: Wireframe Sign-Up View

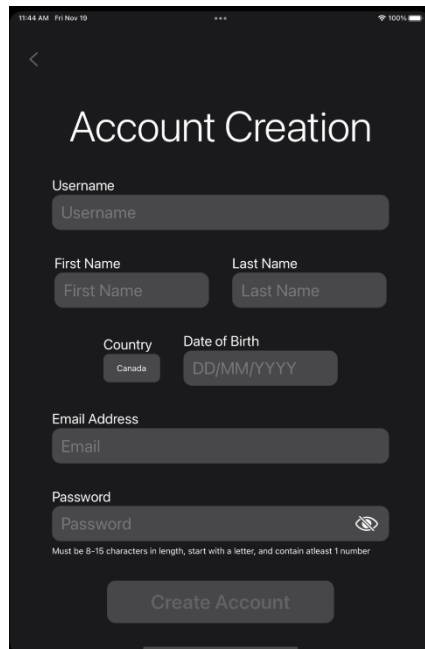
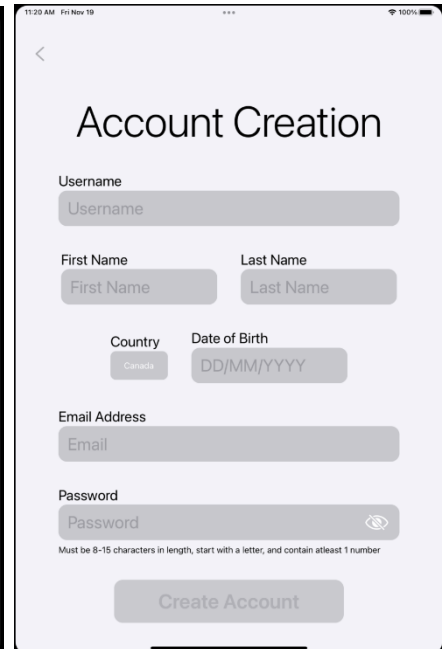


Figure 7: Swiftly Sign-Up View Dark and Light Theme



Chapters View

The third major user-interface view and the most notable one is the chapter's view. This view acts as the home screen where it displays all the chapters, their basic information, and the users progress for that chapter. This page also allows users to navigate to their account page view the navigate icon located in the upper left corner of the view. This page gets refreshed regarding the chapter completion upon every appearance, this will allow users to see real-time data regarding the chapter's completion count. Several features were removed from the wireframe design – the most notable being the “Join a Classroom” button, “View Leaderboard” button, and the lock icons on some chapters. These elements were removed to facilitate more important development features that were focused on.

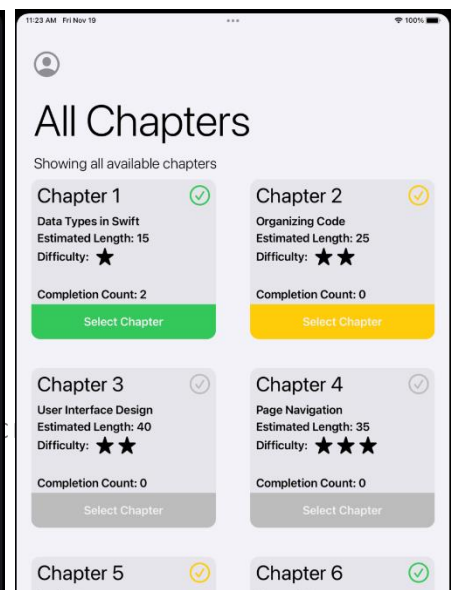
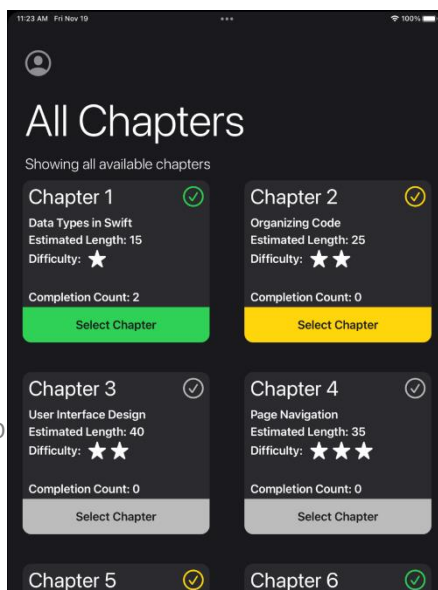
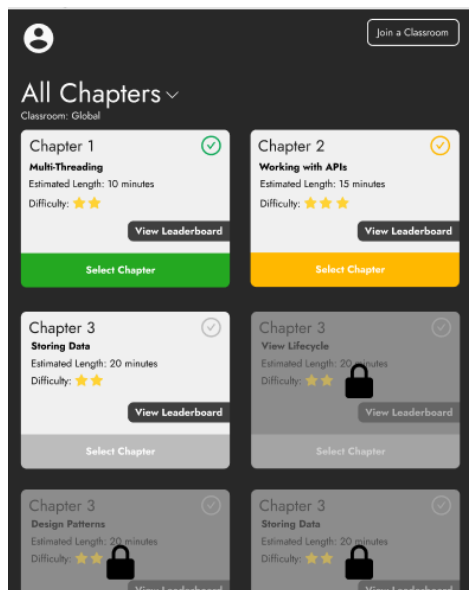


Figure 8: Wireframe Chapters View

Figure 9: Swiftly Chapters View Dark and Light Theme

Chapter Detail View

The next major user-interface view is the chapter detail view. This view appears when the user selects a chapter to start and provides the user with more information about the chapter.

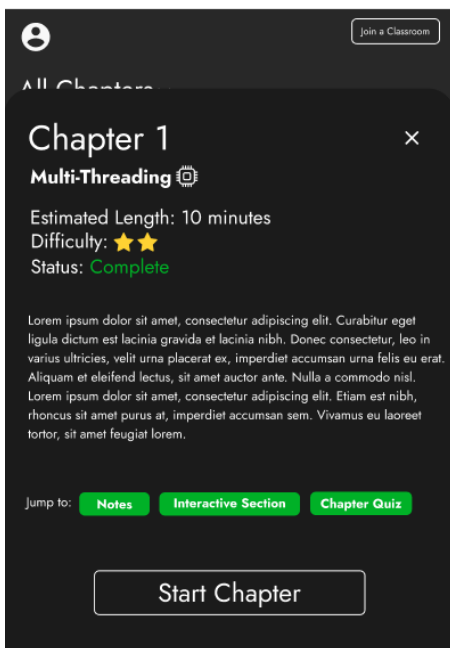


Figure 10: Wireframe Chapter Detail View

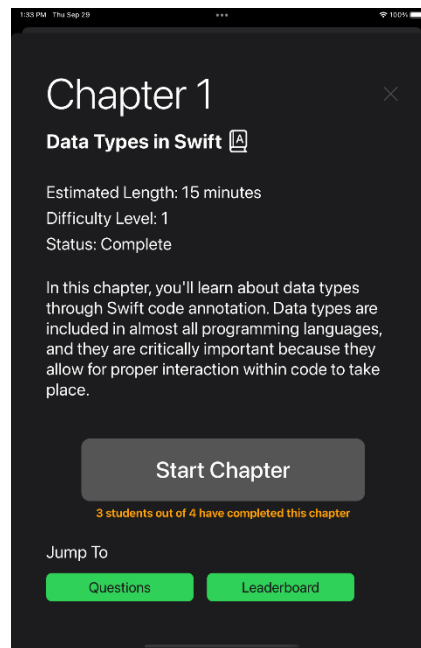
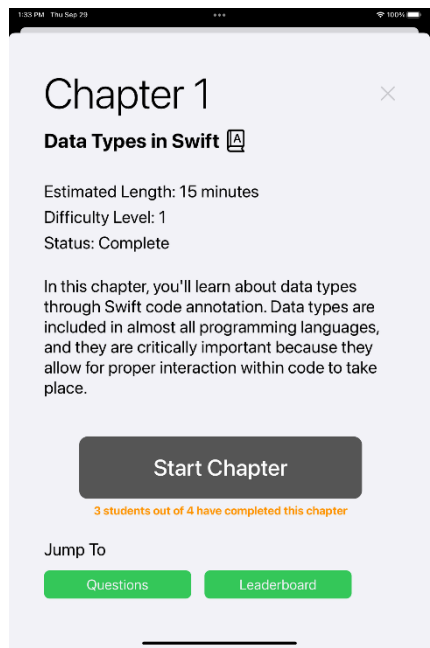


Figure 11: Swiftly Chapter Detail View Dark and Light Theme



Chapter Content View

The next major user-interface view for Swiftly is the chapter content view. This view appears when the user selects the “Start Chapter” button in the chapter detail view. This view consists of several lessons that teach the users about one subject under the topic of the selected chapter. The lessons are presented in sequential order in the sense that they are accessed one after the other. This view dynamically presents the lesson content through a vertical scroll view and a horizontal scroll view. The vertical scroll view allows the user to read the contents of a lesson from left to right and up to down – just like in a book. The horizontal scroll view allows the users to swipe horizontally to the next lesson, also just like a book. At the end of the last lesson there is a “Start Interactive Section” button that lets the

user start the next section of the chapter. In addition to this, through this view the user access to the Swiftly assistant chatbot that will allow them to ask questions about the chapters and their content. The chatbot can be accessed through the question mark icon on the upper right corner of the view.



Figure 12: Wireframe Chapter Content View

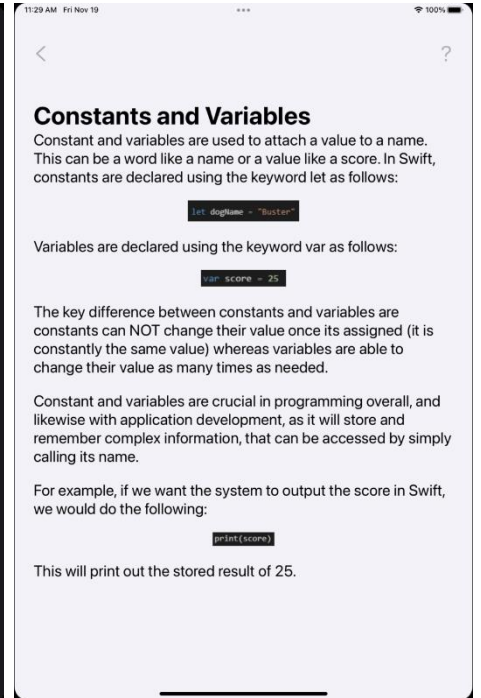
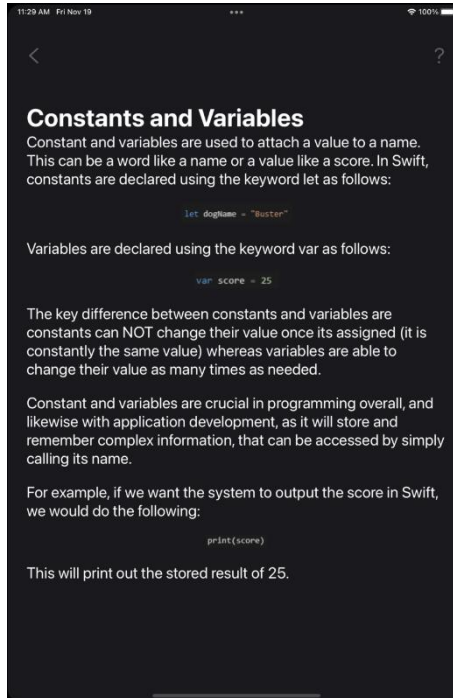
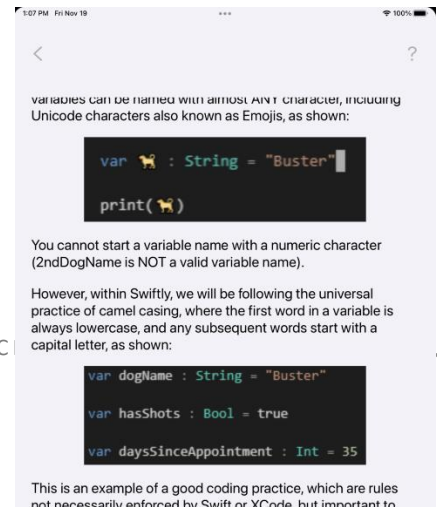
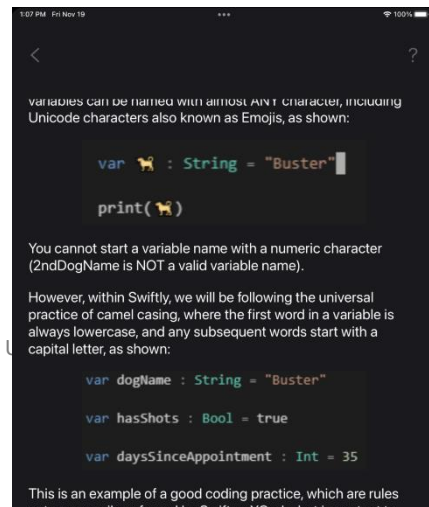
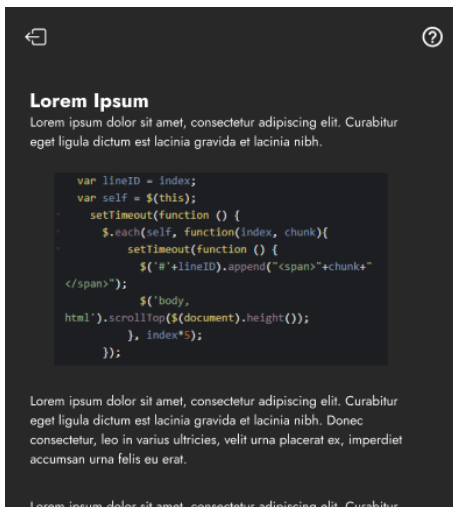


Figure 13: Swiftly Chapter Content View Dark and Light Theme

The lesson content that is provided to the user consists of text and images. The text in the lessons describes and/or references the images to help the user understand the actual implementation of the topic within Swift.



ONOL C

Figure 14: Wireframe Chapter Content View

Figure 15: Swiftly Chapter Content View Dark and Light Theme

Chatbot View

The next major user-interface view is the chatbot page. In this view the user can ask the Swiftly assistant any questions pertaining to the chapters and their content.



Figure 16: Wireframe Chatbot View

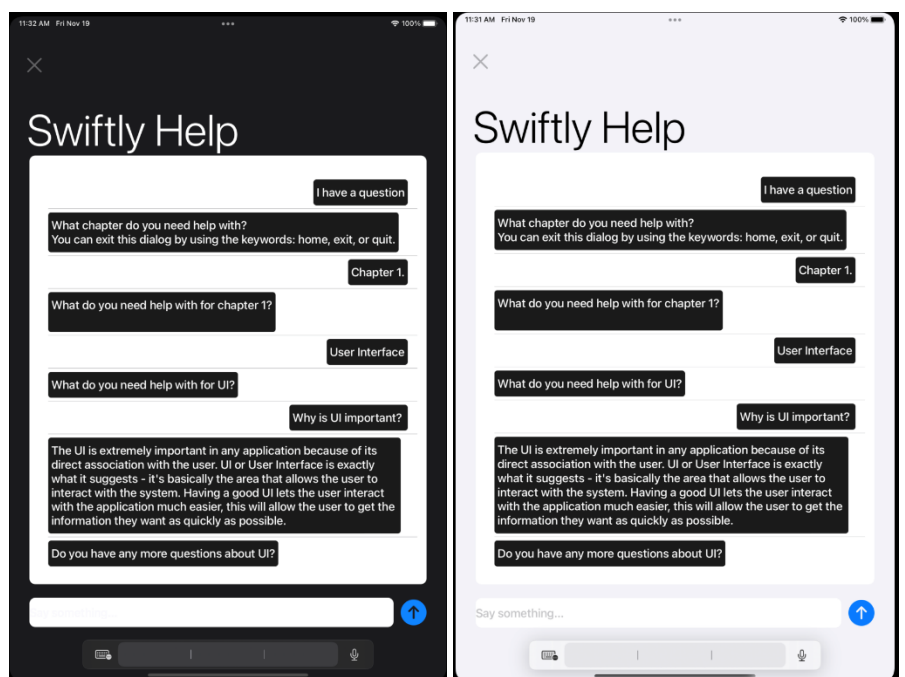


Figure 17: Swiftly Chatbot View Dark and Light Theme

Chapter Questions View

The next major user-interface view is the chapter questions view. In this view the user is shown several questions pertaining to the information that they were taught in the theoretical section of the chapter. The questions are presented so that they show the question's title, type, difficulty, and associated user progress. Questions that are marked green mean that the user has completed it. Questions that are marked with yellow mean that this question is currently in progress. Lastly, the questions marked in grey mean that the user has no progress for them, and they are

currently locked. The user must progress through these questions sequentially to unlock the next question. Once they unlock all the questions, the “Next Chapter” button at bottom of the view becomes available and allows them to start the next chapter. The wireframe design did not incorporate a view with multiple questions – it was only designed to have one single view for one single question. Therefore, no wireframe view exists and so it cannot be included in the screenshots below.

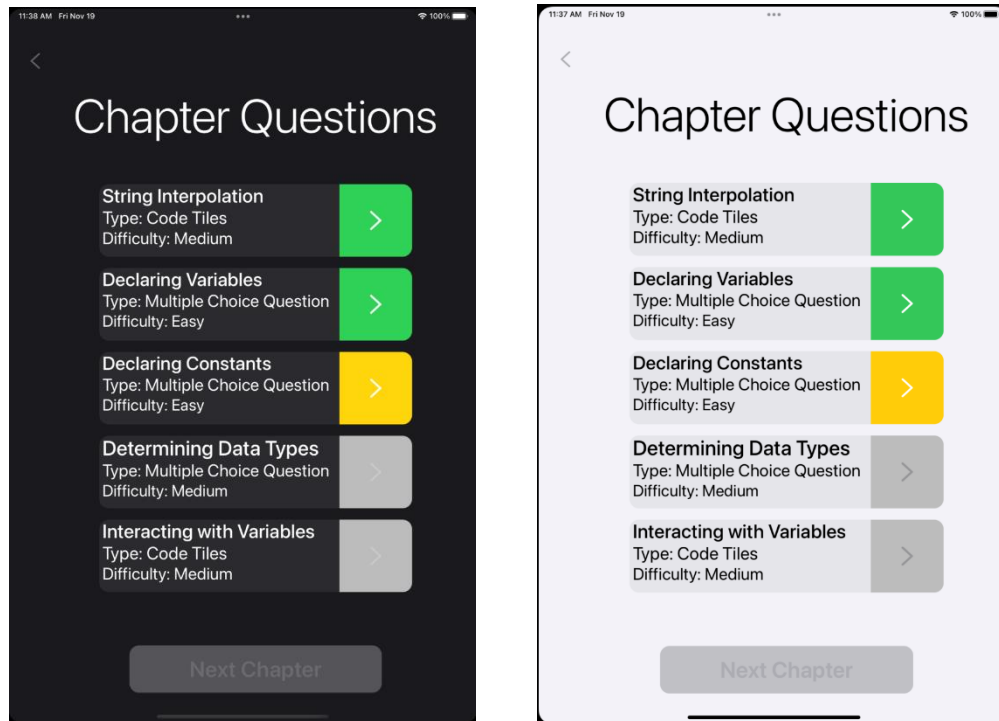


Figure 18: Swiftly Chapter Questions View Dark and Light Theme

Chapter Question View

The next major user-interface view is the chapter question view. This view is presented when the user selects a question from the previous view. Depending on the question a different user-interface will be presented. For the current implementation of Swiftly, there are only two types of questions, and they are code blocks and multiple

choice. The multiple-choice question allows users to select the answers that they think are correct, and the code blocks type question allows users to drag-and-drop the code blocks into the order they think is correct. At the bottom right of the view, is a button that will allow the user to go to the next question. The wireframe design only had a view for the code blocks type of question and so the screenshots for multiple choice questions are only for Swiftly.

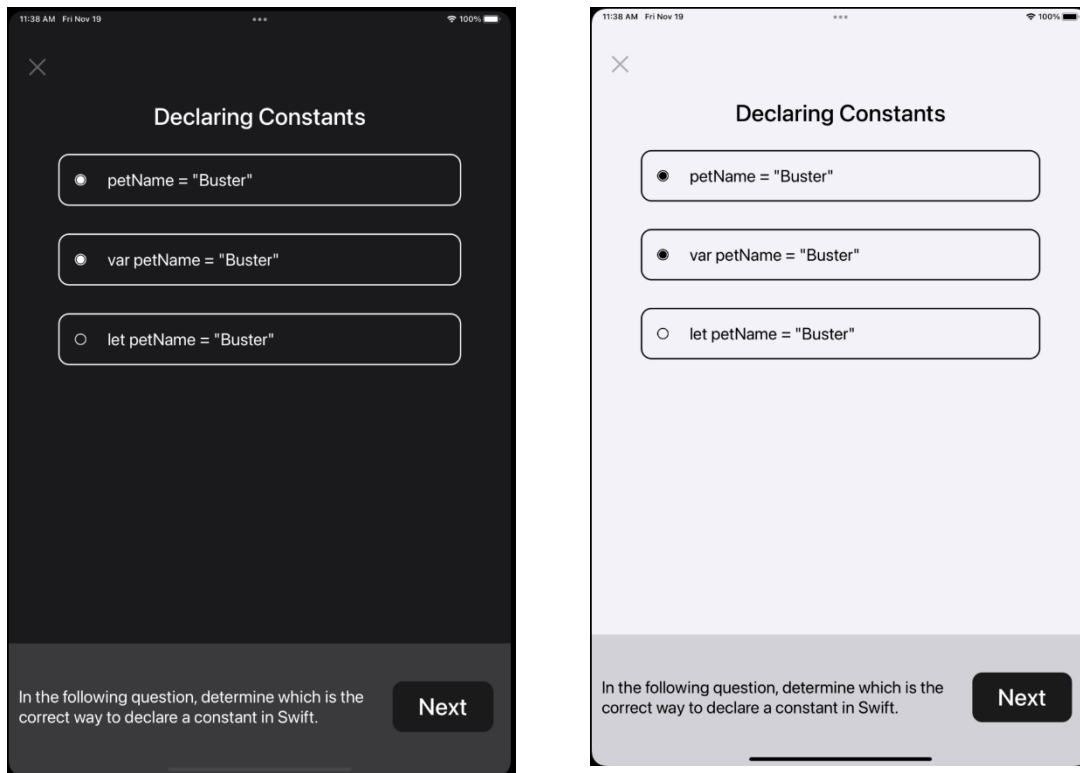


Figure 19: Swiftly Multiple-Choice View Dark and Light Theme

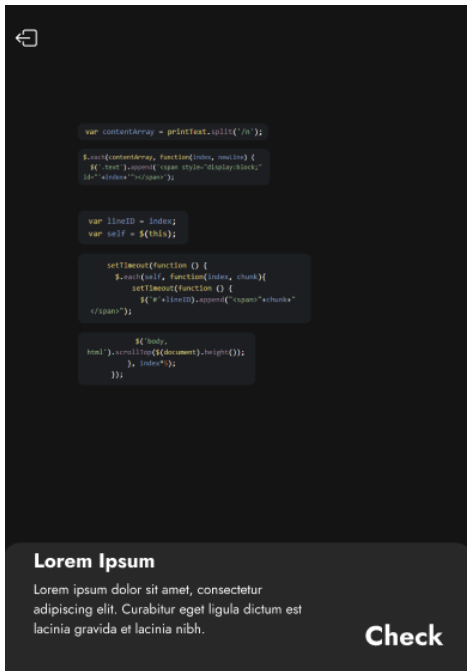


Figure 20: Wireframe Code Block View

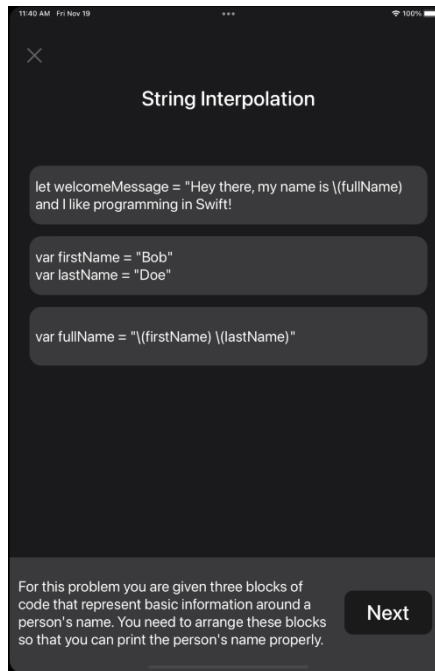
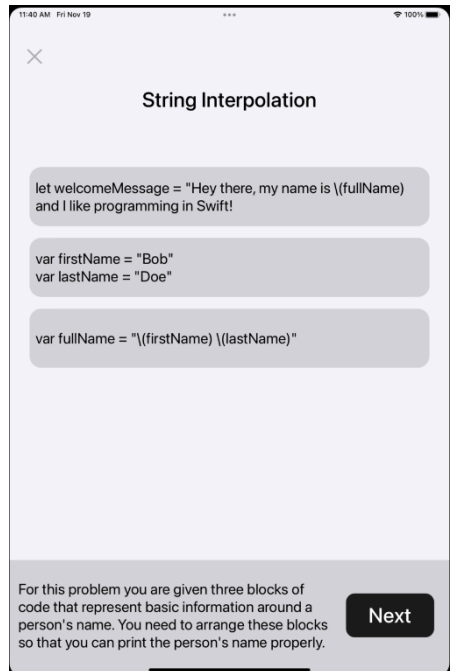


Figure 21: Swiftly Code Block View Dark and Light Theme



User Account View

The next major user-interface view is the user account view. This view can be accessed through the chapters view via the avatar navigation link in the top left corner. In the user account view the user will be able to see all the valuable information pertaining to their account, like their username, full name, country, date of birth, and email address. In addition to this, this view will also show the users some statistics on their progress throughout Swiftly. This area will show them their chapter completion percent, chapters in-progress percent, and their total questions completed percent. At the bottom of this move is the “Logout” button. This button will allow the user to logout out of their account and doing so will bring them back to the login view. Lastly, at the top right corner of the screen is the navigation settings icon, this will present them with a new screen where they can edit their account information.

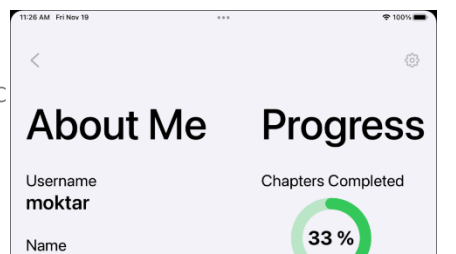
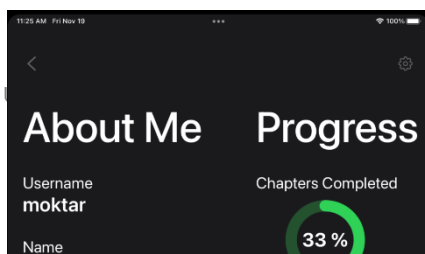
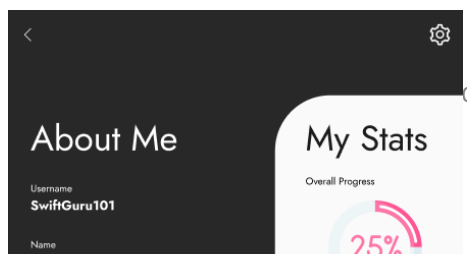


Figure 22: Wireframe Account View

Figure 23: Swiftly Account View Dark and Light Theme

Edit Account View

This is the final major user-interface view of the Swiftly app. In this view, the user is allowed to edit their account by updating their account information.

Figure 24: Wireframe Edit Account View

Figure 25: Swiftly Edit Account View Dark and Light Theme

Leaderboard View

Swiftly also provides users with a Leaderboard View where they can see their progress alongside the progress of all other students who are learning with Swiftly. On this screen the user can also filter via the two filter pickers. The first picker allows the user to filter via chapter – allowing them to be presented with a refined version of the leaderboard specific to the selected chapter. The second picker allows users to filter via country, this will allow them to compare their progress to those in their country.

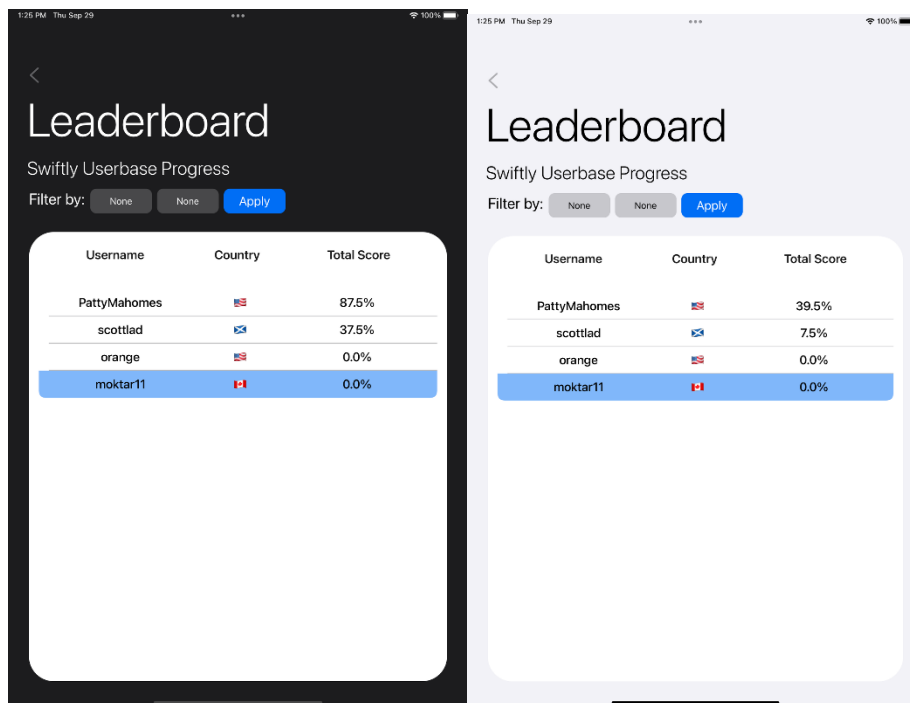


Figure 26: Swiftly Leaderboard Page

Swiftly Website

Overall, the Swiftly website is far less complex and so requires less user-interface screens. The website contains only the necessity views that allow the content creators to accomplish their task.

Login View

The first main user-interface view in the Swiftly website is the login view. Here the content creator can login and start editing and creating chapters. If the user does not have a content creator account, they can sign up for one by following the sign-up link. Note that the user-interface design for the website was based on the wireframe design for the application and so no wireframe was created solely for the website.

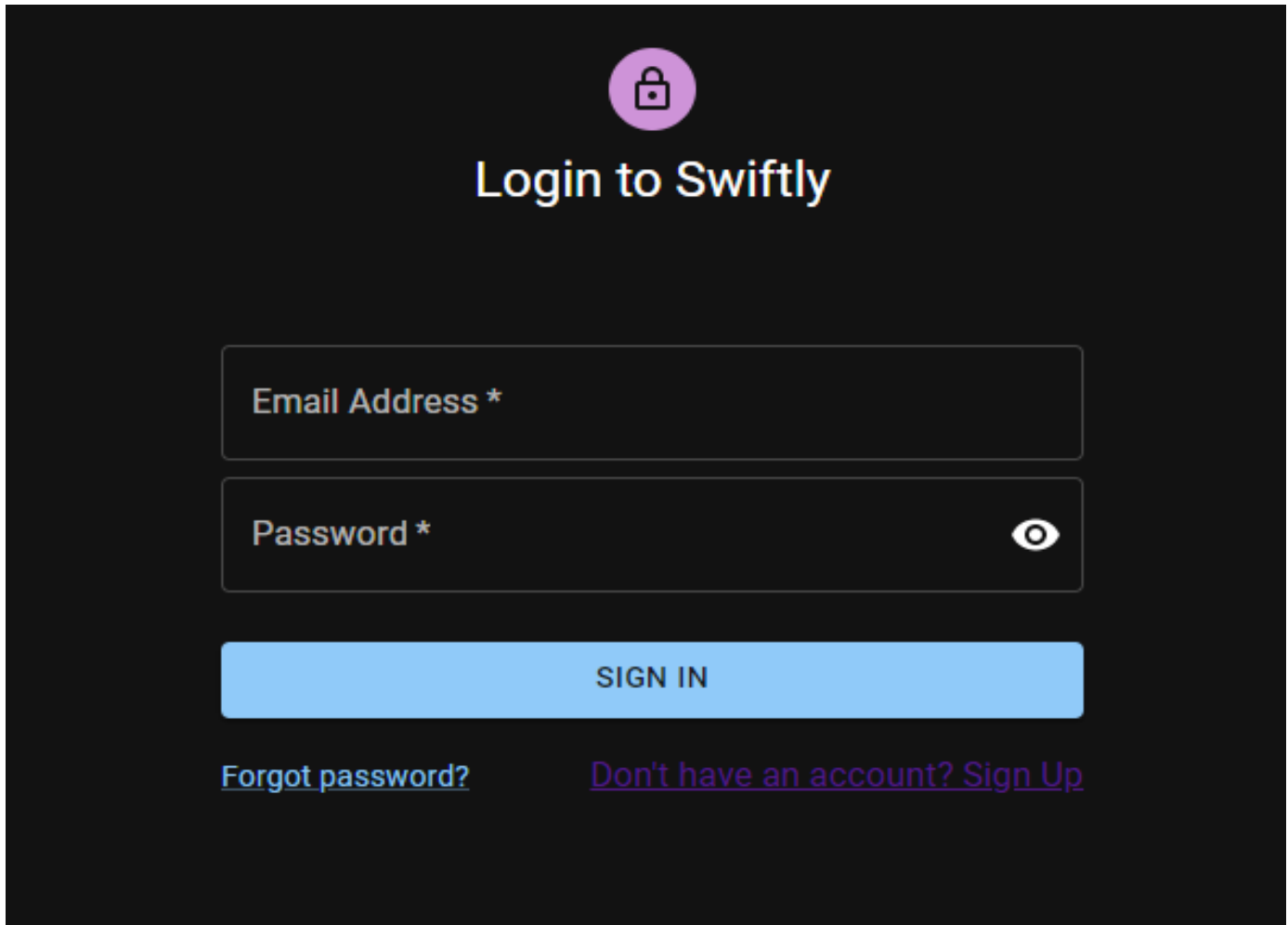
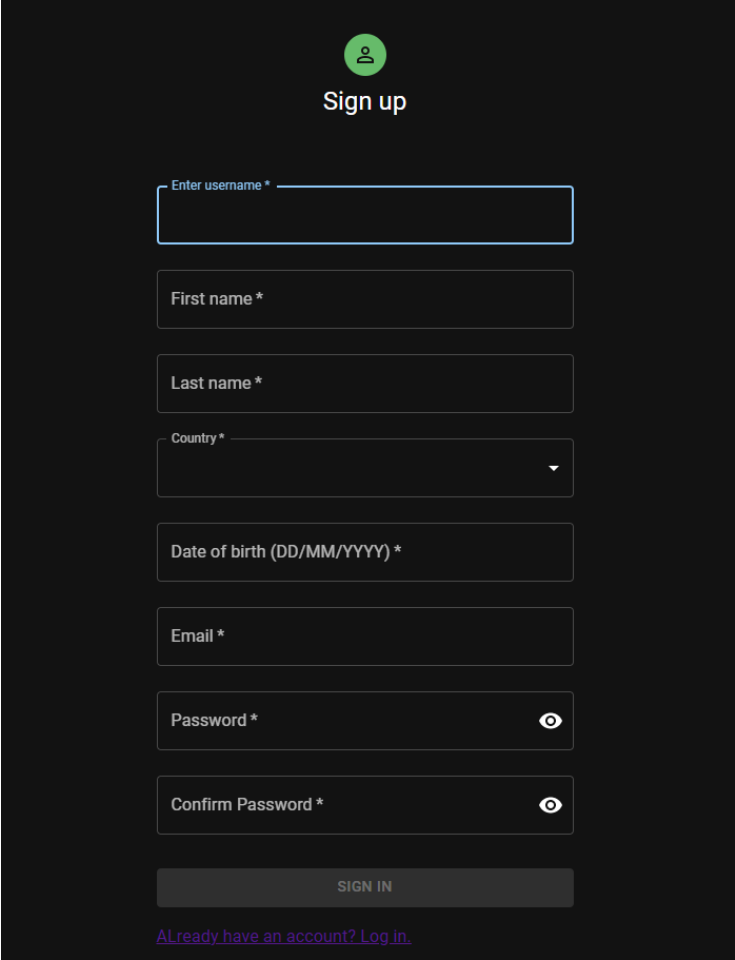
The image shows a login form on a dark background. At the top center is a purple circular icon with a white padlock. Below it, the text "Login to Swiftly" is displayed in a white, sans-serif font. The form consists of two input fields: "Email Address *" and "Password *". The "Password *" field includes a white eye icon on the right side, indicating a toggle for password visibility. Below the input fields is a wide, light blue button with the text "SIGN IN" in white, uppercase letters. At the bottom of the form, there are two links: "Forgot password?" in white text and "Don't have an account? Sign Up" in purple text.

Figure 27: Swiftly Website Login View

Sign Up View

This is the second major user-interface view for the Swiftly website. In this view, the user can create their content creator account. This view checks for all the valid information and ensures that the username and email are not already taken. If these checks pass, then the content creator account has been created.



The image shows a mobile application sign-up screen with a dark background. At the top, there is a green circular icon with a white person silhouette, followed by the text "Sign up". Below this, there are several input fields: "Enter username *" (a wide text box), "First name *" (a text box), "Last name *" (a text box), "Country *" (a dropdown menu), "Date of birth (DD/MM/YYYY) *" (a text box), "Email *" (a text box), "Password *" (a text box with an eye icon for visibility), and "Confirm Password *" (a text box with an eye icon). At the bottom, there is a grey button labeled "SIGN IN" and a link that says "Already have an account? Log in."

Figure 28: Swiftly Website Sign Up View

Dashboard View

This is the main view of the Swiftly website. It can only be accessed with an authenticated login session (achieved through a successful login from the login page). There are a few major elements inside of this view. Do note that the UI for the main dashboard is still in its early stages and is subject to extensive changes.

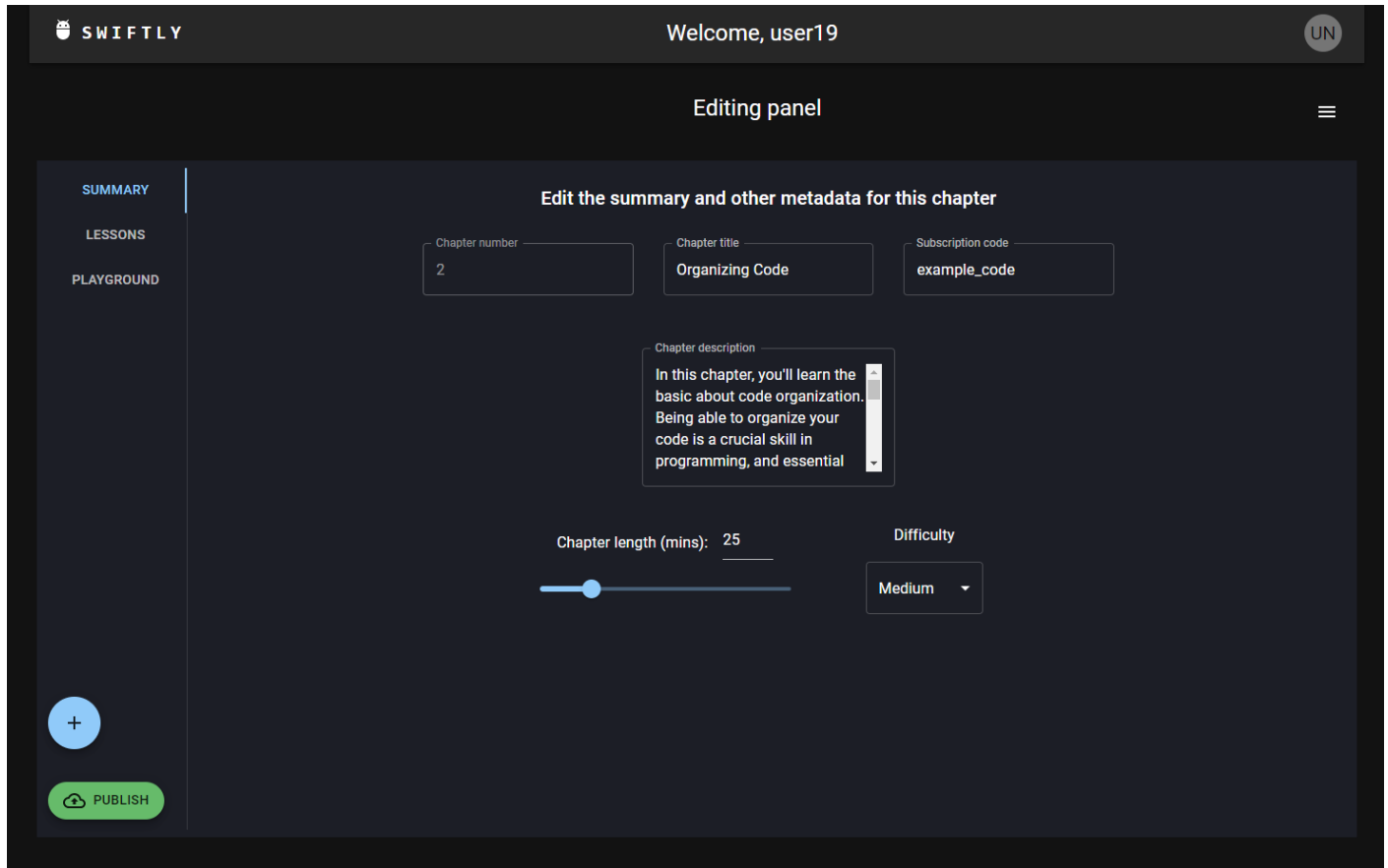


Figure 29: Swiftly Website Dashboard View

The top of the dashboard states the currently logged in user as well as a clickable icon that presents the option to logout and manage profile details

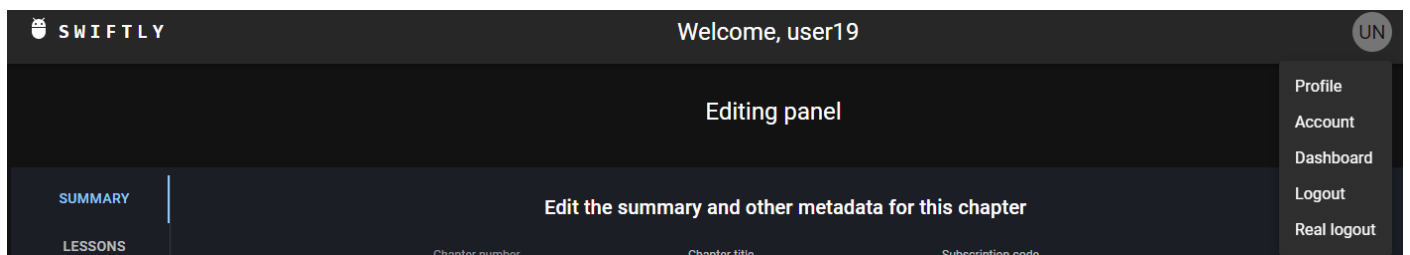
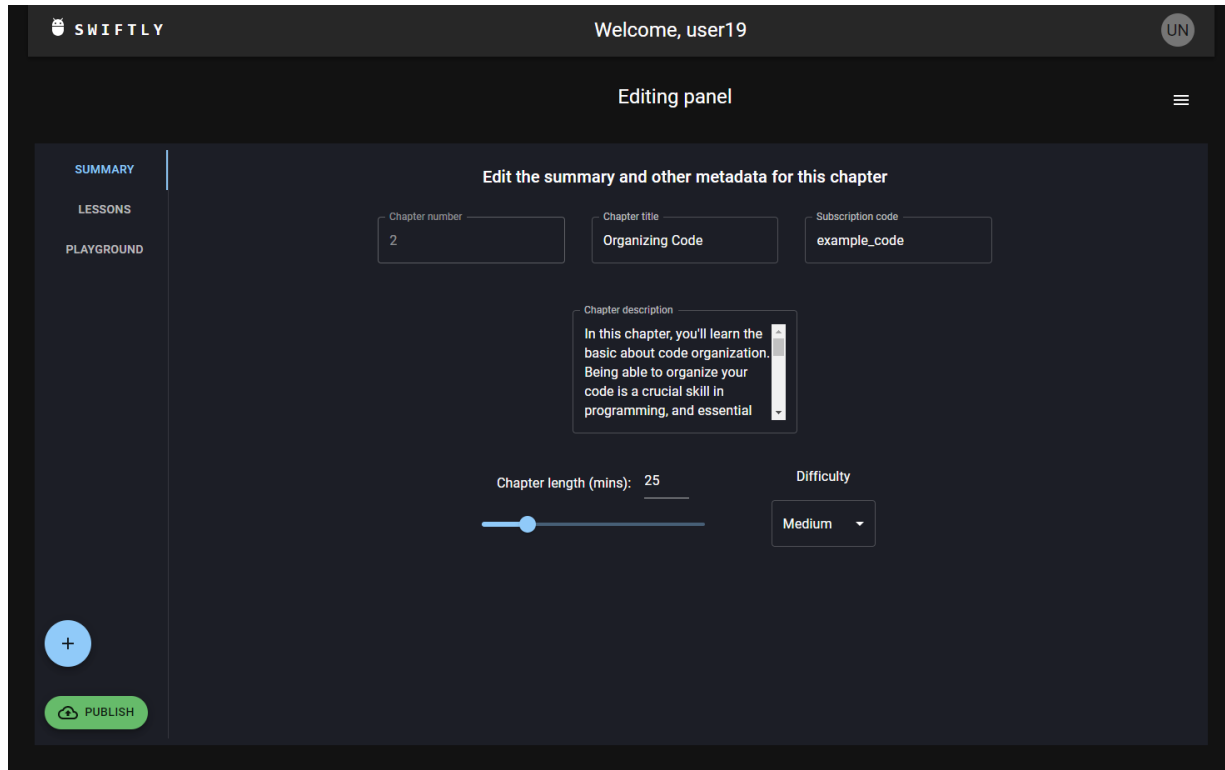


Figure 30: Dashboard status bar

The center of the page consists of the editing and Chapter drawer. The chapter drawer can be accessed from the three lined button on the top right. It displays the list of chapters that belongs to the logged in user with the ability to refresh the list at any time.



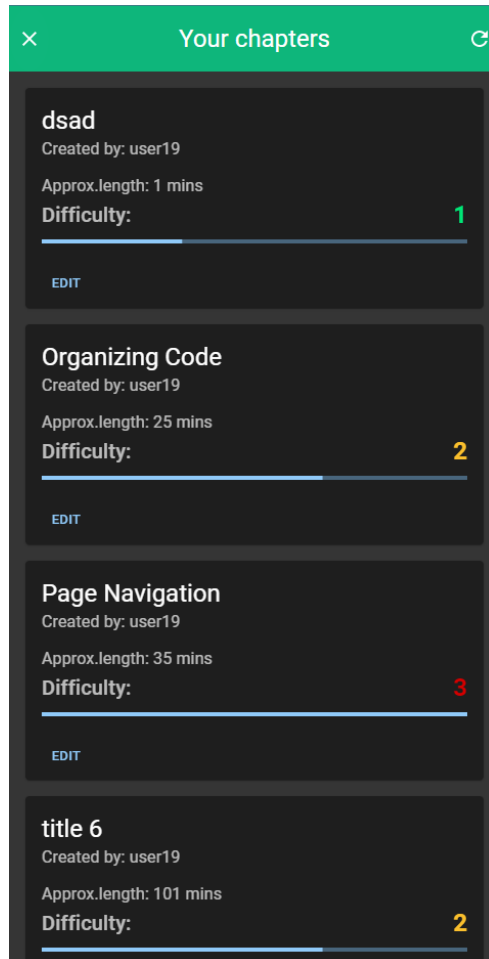


Figure 31.1 & 31.2: Main content creation interface and chapter drawer

The list contains chapter cards, each providing a summary of a chapter such as chapter name, author, length of chapter's content and its difficulty level. The difficulty level ranges from 1 to 3 and is also represented by its corresponding color as pictured below. Clicking on the edit button loads that chapter's content into the editing panel for modification.

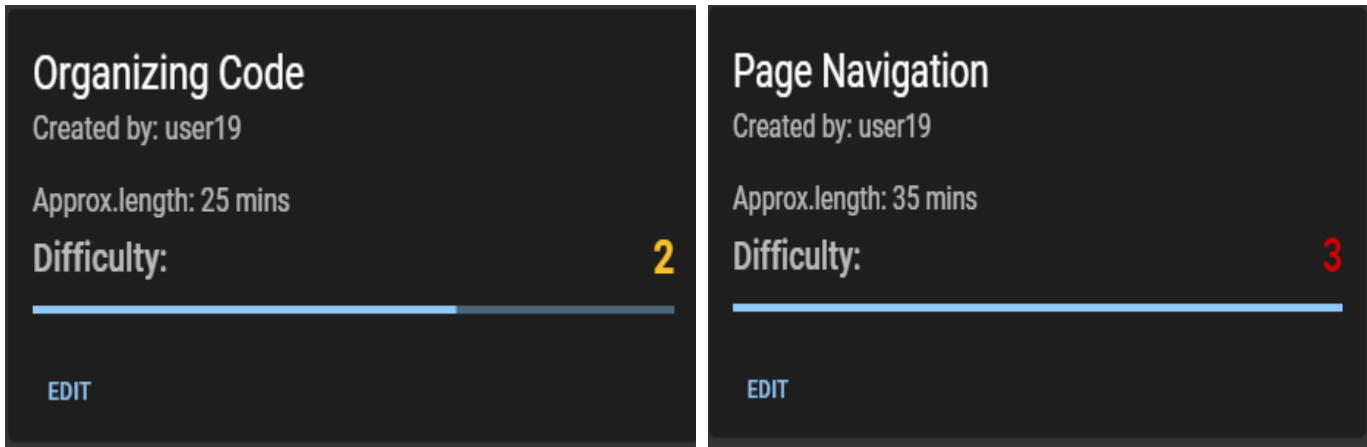


Figure 32: Chapter cards

The editing panel is where chapters can be created and modified. This panel contains three different “tabs”. The metadata tab acts as a place to enter the key details about the chapter such as its title, description, length, and difficulty. There is also the option to create a new chapter which provides the option to start from scratch. The publish button publishes the chapter into Firestore.

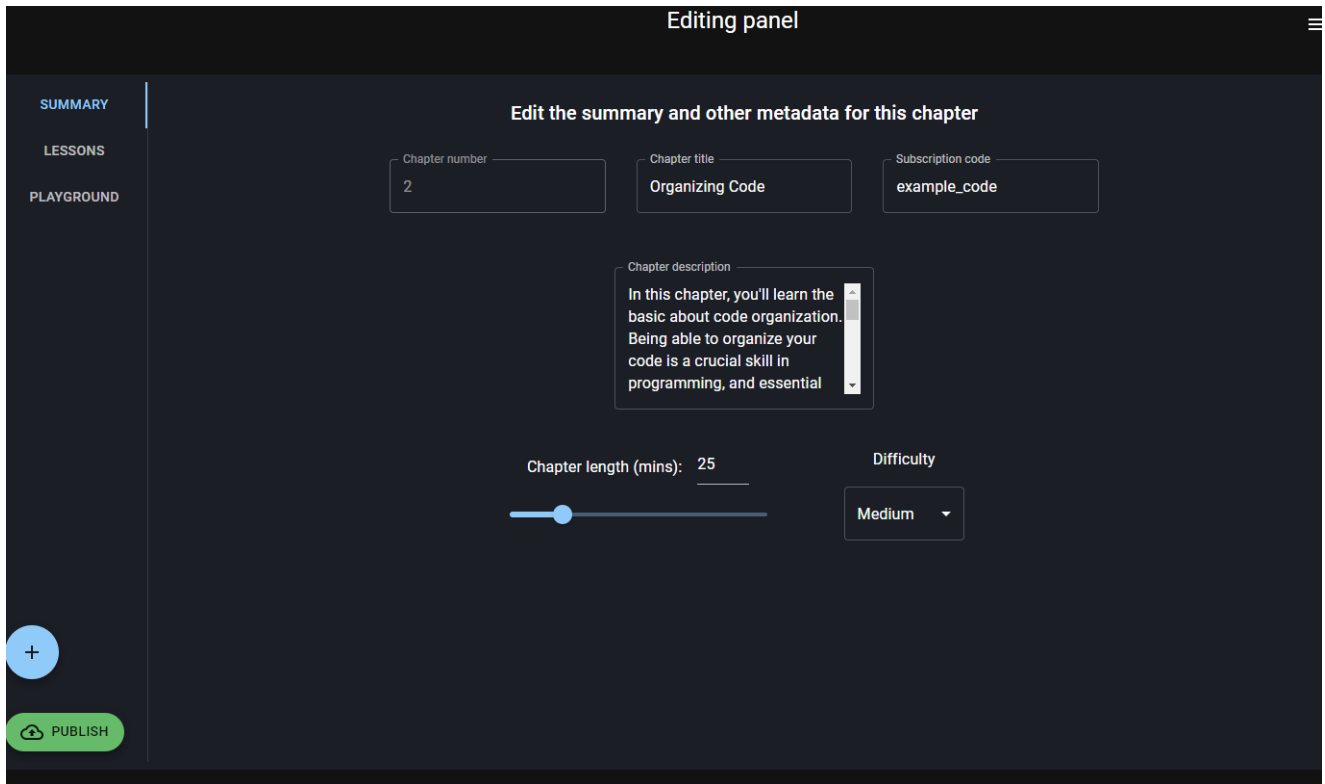


Figure 33: Metadata tab of the editing panel

The lessons panel provides the ability to create and alter a chapter’s lesson. The user has the option to add snippets of text and images which will then be rendered dynamically into the Swiftly app. The lessons can be easily switched to from the dropdown located on the right side.

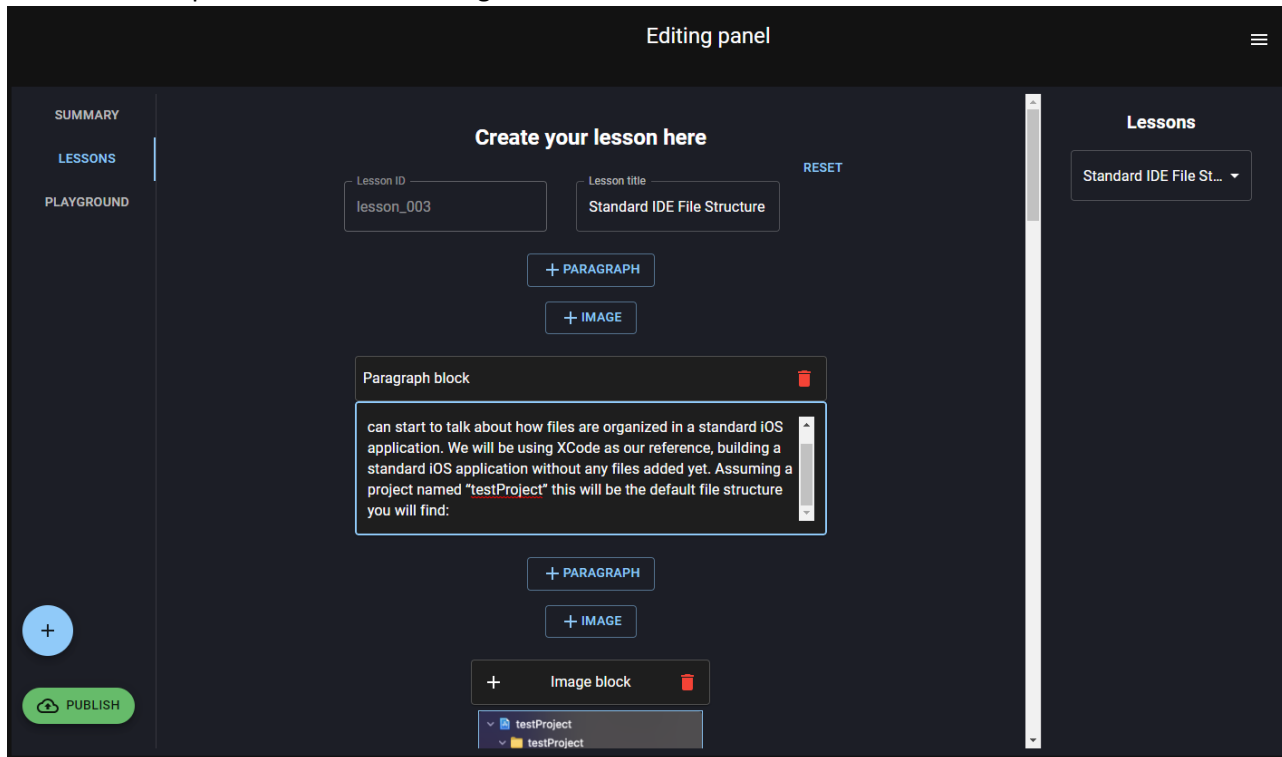


Figure 34: Lesson tab in editing panel

Lastly, there is the playgrounds tab which will be used to add and modify playground content. This includes modifying, adding and deleting code block and multiple-choice questions. It has a similar layout to the lesson editor tab with a dropdown menu on the side where the user can select existing playground questions.

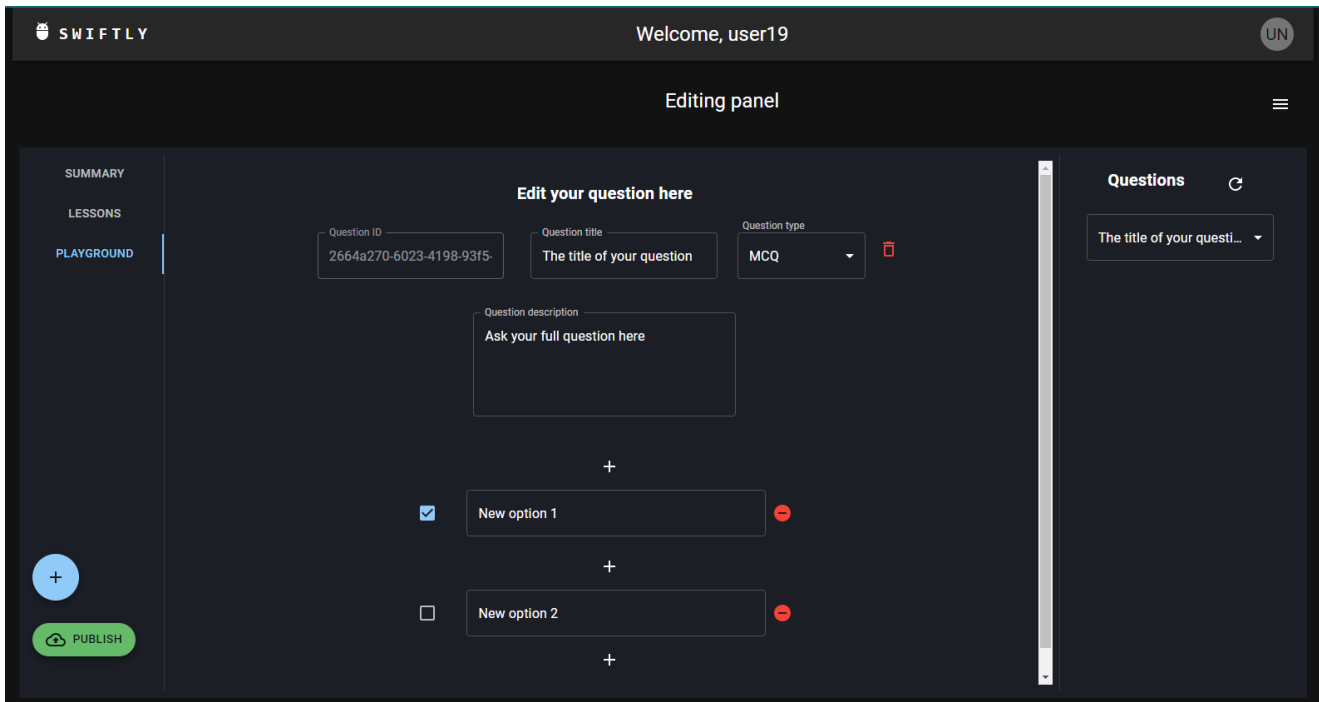


Figure 35: Playground editor tab

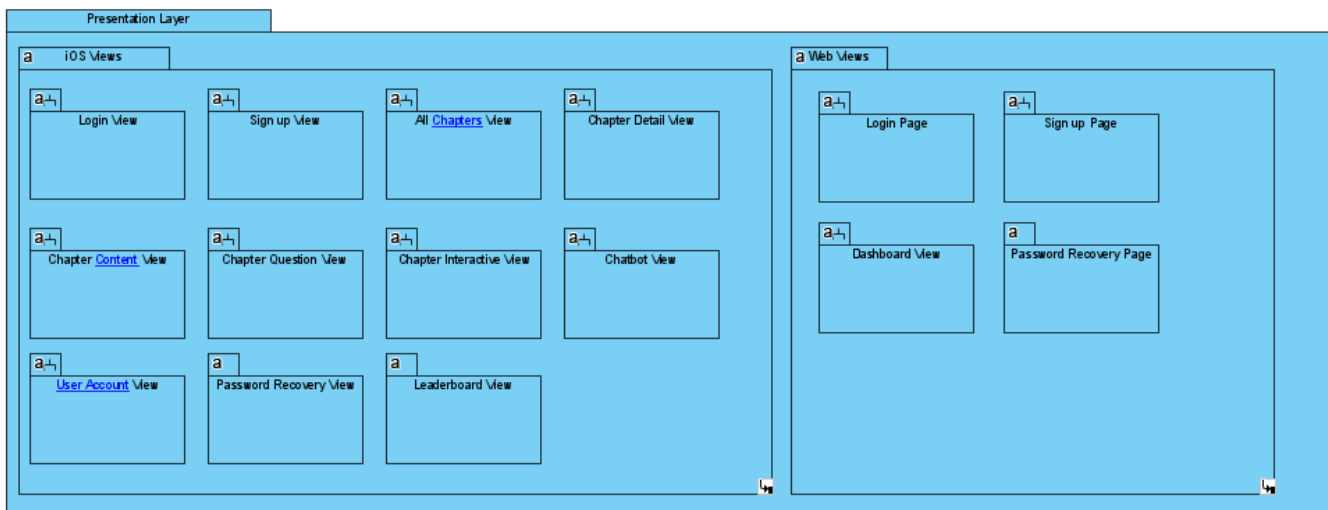
PROJECT ARCHITECTURE

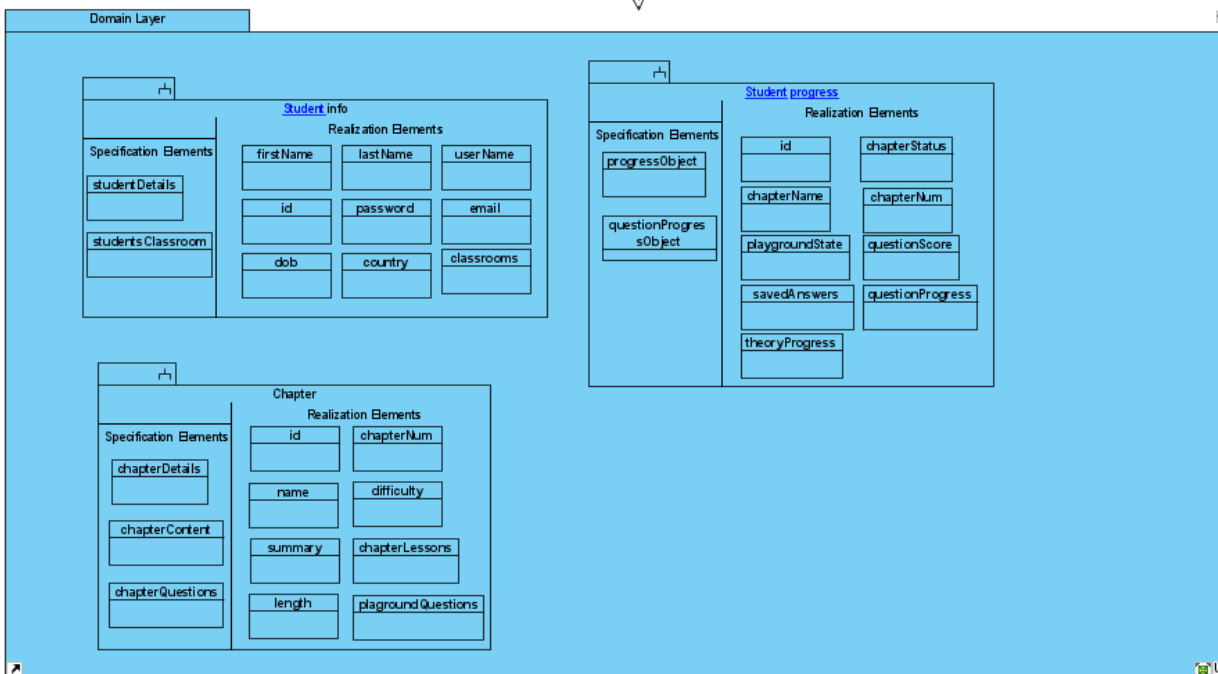
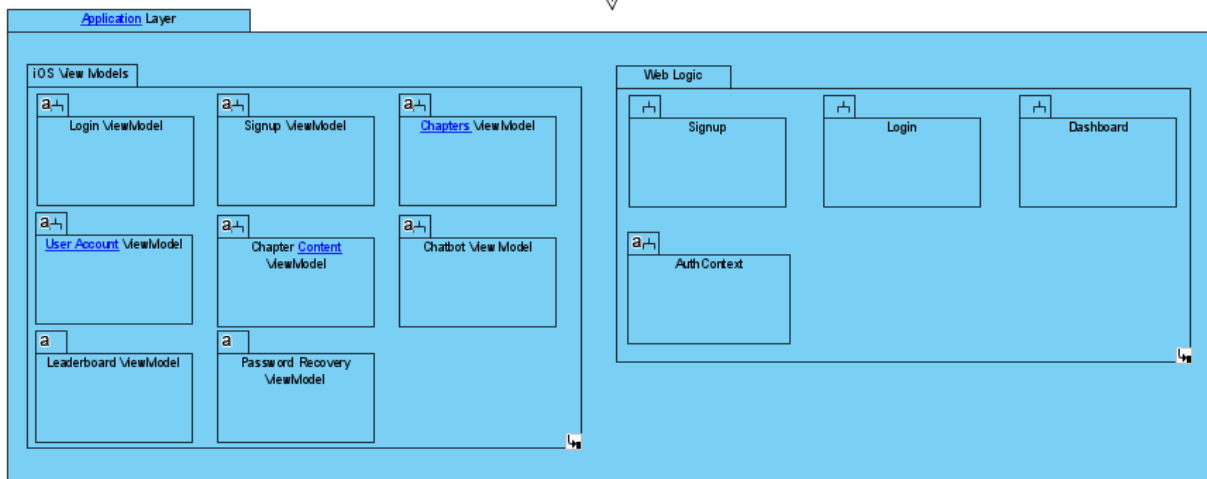
This section of the document will be covering the Software Architecture layout that makes up the main Swiftly application and the Swiftly Expert companion website. Everything involving the project architecture will be covered within this section, including diagrams. The architecture overview section will be used to go over the overall architectural design of the project. The system components section will cover the main components that make up the application and website, and their roles they play to get the software running as intended. Finally, the deployment model section will cover The VPository can be found at the following link: The VPository that includes the design, interaction and deployment model can be found at the following link: <https://online.visual-paradigm.com/w/dbokpyzr/drive/#diagramlist:proj=3&open>.

ARCHITECTURE OVERVIEW

Swiftly was identified to be an information management system, and so it took architectural reference from the information systems architecture. Due to the nature of the Swiftly and the development environment of the project,

the Model-View-ViewModel software design pattern was implemented – this is worth noting because it greatly influences the architecture of the project; because of these reasons, the layered architecture was selected. This architecture best represents Swiftly and its inner components – representing the unique layers within the system and their dependencies to one another. The layered architecture was split up into four unique layers, and they are the presentation layer, application layer, domain layer, and infrastructure layer. It’s worth noting that some of these layers are modularized into an iOS and web component, this was done to ensure that each application is represented properly according to the respective platform. First, the presentation layer encompasses all the views within our project, it focuses solely on the user-interface aspect of the application. Next, the application layer focuses on the logic associated with the views. These classes represent the logic behind the scenes, incorporate user-interface changes, and do most of the heavy lifting on the client side. Next is the domain layer, here are the classes that do not interact with the presentation layer at all, and only focus on computational processes. Following this is the infrastructure layer. This layer is modularized into two sublayers: the services layer and the persistence layer. The services layer consists of behind-the-scenes classes and objects that work to make sure all authentication is correct, data is correctly validated, and logic for remote connections is working. It primarily deals with remote database connections and API integration. Next, is the persistence layer. Here all the forms of persistence in our application are found, and they consist entirely of remote databases. All the layers within the architecture have cascading dependencies on one another as per their order – however, regardless of their order, they all depend on the significant importance of the infrastructure layer. The overall architecture, its layers, and respective packages can be seen in the figure below.





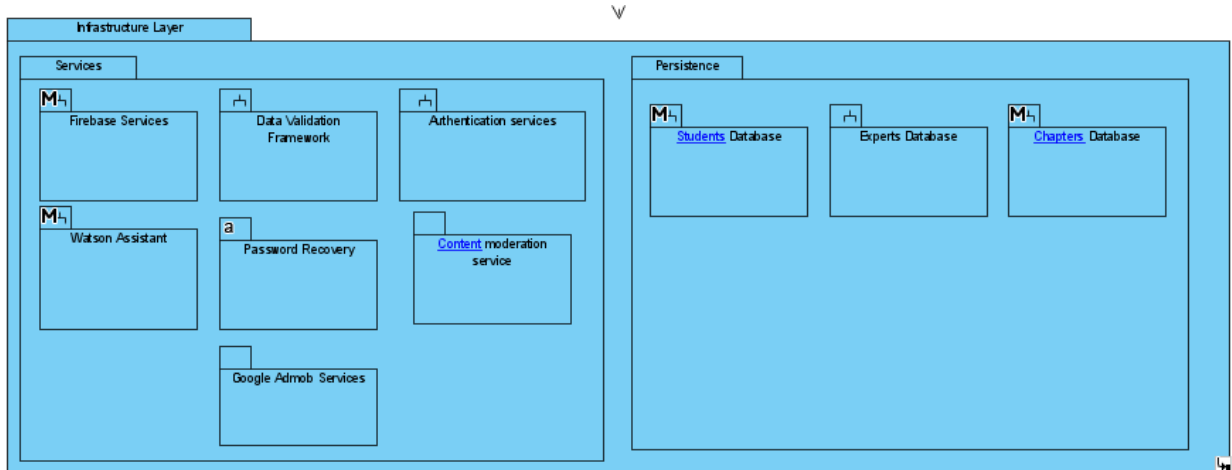


Figure 35: Swiftly Architecture Overview

SYSTEM COMPONENTS

Cloud Database

A system component that is crucial to carrying out the requirements of Swiftly is the remote cloud database. Swiftly is an online iOS app development teaching tool that provides users with their own accounts for tracking and saving their progress in applications and allows for content to be created and posted to the app through a separate website. This requires a database that holds the students account information, their user progress, the lesson content itself including the lessons and interactive playground information, and another expert account collection which will be used to access the Swiftly Expert website.

The Chapters collection within the database holds all information regarding each chapter of the application. This includes chapter information such as the description, author and title all stored as strings, and the difficulty, estimated length of completion, the chapter number all stored as integers. This information will all be shown to the user when they first tap the chapter, to get a general understanding about the chapter before starting it. Then, within each chapter is a lesson and playground subcollection. Each lesson subcollection holds an array of strings called lesson content, which includes the text for the lessons and the images for the lesson stored as data64 strings. Each playground subcollection holds the code blocks, an array of strings that holds each multiple-choice answer or the lines of code needed to be rearranged if it's a code-blocks style question, the description, title, and type of question stored as a string, and a string array for the list of answers if the question type is multiple choice based.

The Student collection contains all the information regarding each account created through the main Swiftly application and uses it to go through the chapters provided by Swiftly. Each student user contains a username, country, date of birth, email, first and last name and password stored as strings as general account information. The email and username are fixed to each account, meaning they are unable to be changed through the edit user page, but every other field including the password can be updated. Within each user collection, a user progression collection is automatically generated. Within the progression is a subcollection for tracking the users answers and scores as arrays of strings and integers respectively for the number of questions in the chapter, and the overall progression status for the lesson, playground, and chapter, stored as a string in the state of either “incomplete”, “in progress” or “complete”.

To be able to accurately create this user progress collection, it requires an association with the chapters collection. This is done by analyzing the downloaded chapters and determining how many questions and answers there are for each; with this data, an appropriately sized array is created to reflect the chapter correctly. This process can be seen in the behavioral diagram for the “creating user account” use case. On top of creating the user progress collection, it also needs to be constantly updated as the user progresses through the lessons, completes the playground questions, and finishes the chapters. This involves the app updating the values within the collection, and then sending the updated collection back to the cloud database in real time which can be seen in the behavioral diagram for the updating user progress use case. Finally, the Expert collection contains all the information regarding each “expert” account made through the Swiftly Expert website, used to represent users who are creating the content for Swiftly as opposed to the actual users of the main application. Like the student collection, each expert user has a username, country of origin, date of birth, email, first name, last name and password stored as Strings.

The cloud platform chosen to host all this information was Google’s Firebase. This was because of its robust and dynamic storing and retrieving capabilities, being able to store multiple data types, and creating subcollections within collections. Firebase was also chosen for its excellent support of Swift and JavaScript, both having extensive and feature rich packages, APIs, and documentation. This was important as Swift was the language used to develop the application, and JavaScript (ReactJS) was used to develop the Single page application (SPA) website. The SPA website takes advantage of the authentication and database capabilities for firebase to create an SPA through which Swiftly Experts can create and publish chapters into Firestore’s database. The process of creating/modifying chapters is shown in the behavioral diagram for creating chapter content.

Cognitive Chatbot

Another main system component for Swiftly is the cognitive chatbot system. The chatbot was designed in correspondence with the chapter’s content and aims to assist users by answering their questions about the topics taught in the chapter. The actual development of the chatbot was done through IBM Watson Assistant, which is a platform that IBM offers to create, implement, and deploy cognitive chatbots using their Watson technology. The chatbot is developed using IBMs provided IDE on their website, which provides a dialog builder that utilizes intents, entities, and user inputs to detect, recognize and respond to what the user is looking for. The chatbot is then deployed to the main Swiftly application using APIs and packages provided by IBM for Swift development, which can be seen in the structural overview for the “accessing chatbot services” use case.

Once fully implemented, the user simply only must press a question mark button found on the top right corner of the theoretical section, and will automatically be taken to a messaging page, where the cognitive chatbot will automatically send a message to the user, prompting them to ask any questions about the question they're struggling in, utilizing the implemented intents and entities on the backend to understand what the user is looking for. The process of starting the chatbot service, retrieving the API call from IBMs cloud server, and retrieving questions and delivering responses can be observed in the behavioral diagram for the "accessing chatbot services" use case.

DEPLOYMENT MODEL

The deployment model for the overall project scope of Swiftly is shown in the figure below.

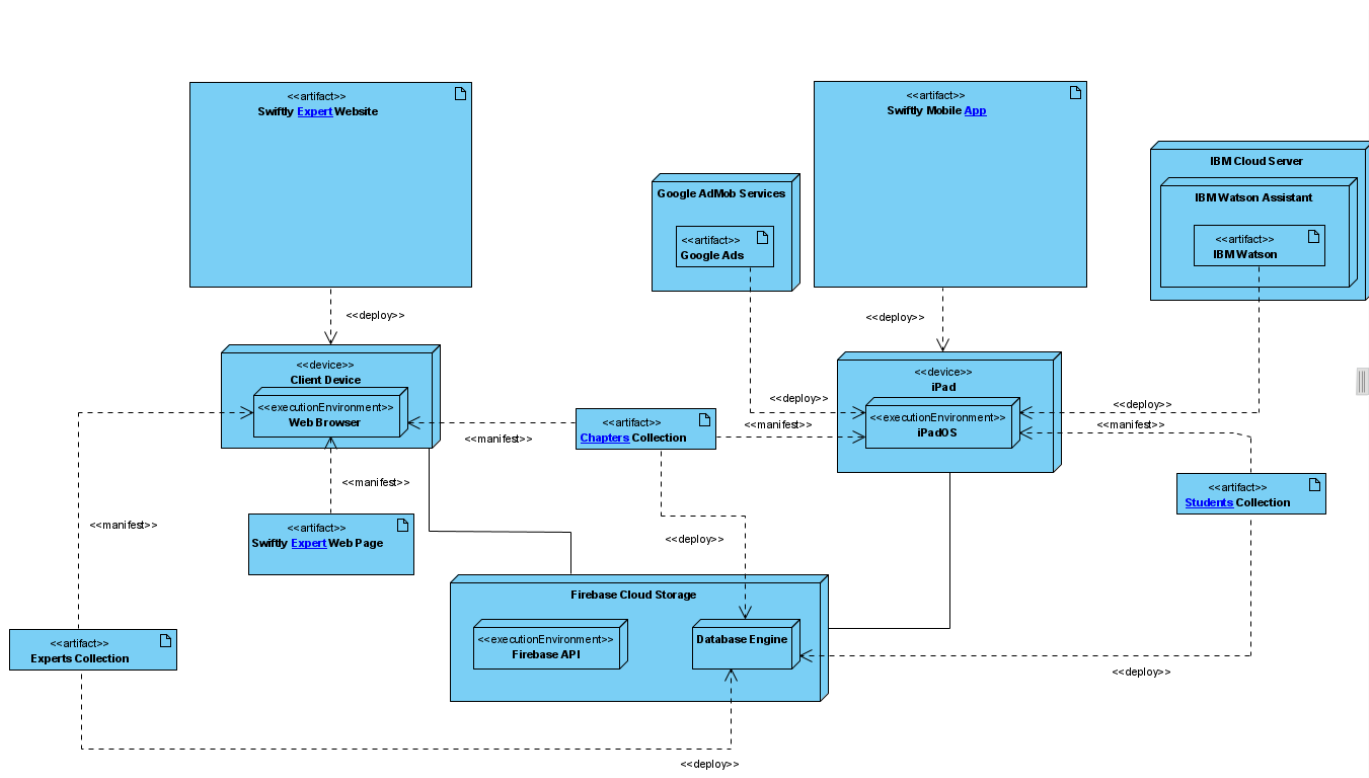


Figure 36: Deployment Model for Swiftly

The large squares on the left and right of the top center of the page represent the main artifacts – the Swiftly Expert website, and the Swiftly mobile application. The Swiftly mobile application is deployed on iPads through iPadOS, with the website being deployed onto any modern web browser through any internet connected device. The main Swiftly application manifests the student and chapter information to the device, from the student's and chapters collection which deploys to the Firebase Cloud Storage Database Engine, utilizing Firebase APIs and the Firebase packages built for Swift and XCode. The app also contains the chatbot, which is deployed from the cognitive chatbot system from Watson's Chatbot API and packages. For the website, it gets the expert and chapter information manifested to the

web app, developed in React JS using Tailwind for styling and UI. The expert and collections artifacts are retrieved and deployed to the database engine within the firebase cloud storage, utilizing firebases provided JavaScript packages to work with the website.

PROJECT PLAN

This section of the document covers information pertaining to the development plan laid out for Swiftly. The different plans that are being presented were all developed through Jira, they can be found at: <https://moktar.atlassian.net/jira/projects>. The first subsection under this section is Iteration Plan, where the development iterations and their goals are discussed. Following this, is the Risk Management Plan. Here the process of risk management is discussed; additionally, all information pertaining to identified risks and their mitigation is discussed. The table below represents a responsibility matrix that indicates what each team member is responsible for over the project plan.

Project Responsibility	Team Member 1	Team Member 2	Team Member 3
Project Management			
Project Owner	Tobias Moktar		
SCRUM Master		Arjun Suthaharan	
Risk Analyst			Madhav Jaisankar
Requirements Engineering			
Requirements / Business Analyst	Tobias Moktar		
Stakeholder Champion (by Stakeholder)	Tobias Moktar – Individual Consumer	Arjun Suthaharan – Education System	Madhav Jaisankar – Software Industry
Functional Area Champion (by Functional Area)	Tobias Moktar – Accessing Chapters	Arjun Suthaharan – Account Management	Madhav Jaisankar – Content Creation
User Experience Design Lead	Tobias Moktar		Madhav Jaisankar
Software Architect			
Software Architect		Arjun Suthaharan	
Requirements Model Lead	Tobias Moktar		
Domain Model Lead	Tobias Moktar		
Design Model Lead		Arjun Suthaharan	
Deployment Model Lead		Arjun Suthaharan	
Interaction Model Lead		Arjun Suthaharan	
Construction			
Full-Stack Developer	Tobias Moktar	Arjun Suthaharan	Madhav Jaisankar
Integration / DevOps Lead			Madhav Jaisankar
Testing			
QA Lead			Madhav Jaisankar
Verification and Validation Champion (by functional area)	Tobias Moktar – Accessing Chapters	Arjun Suthaharan – Account Management	Madhav Jaisankar – Content Creation
Test Model Lead	Tobias Moktar		

Support		
Tools and Devices Support		Madhav Jaisankar
Communication Support	Tobias Moktar	

ITERATION PLAN

The development plan for Swiftly is to follow the Scrum development framework, meaning that the work term will compose of iterations. For this development, sprints, or iterations, will be once a week in length - ensuring that enough time is allocated for the designated tasks. For this first term of development, there will be a total of seven iterations starting from September 12, 2021, until December 11, 2021. Each of these development iterations will contain critical use-cases that ensure customer requirements are being met. The first sprint was designated to setting up project environments, researching the domain, contacting the domain expert, and fine-tuning project details. The goals for the second sprint were to create a wireframe UI design, start project development, research computer science curriculums from across Ontario, and to start developing content for our chapters. For the third iteration the goals will be primarily based on database development, database integration, and accessing chapter content. The goal for iteration four is in regard to chapter test access, user leaderboard access, and chatbot research. Following this iteration five will aim towards developing a chatbot, integrating the chatbot, and implementing the 'join a classroom' feature. Following this, iteration six will aim to implement the update account functionality, retrieve findings on device specific features, and more testing. Finally, the goal of the seventh iteration is to incorporate the personalized learning functionality if all goes according to plan. These goals and the structures of these iterations are very variable, meaning that their goals and tasks are not finalized and will most likely change as development on Swiftly progresses.

For the second term of development, there will be a total of thirteen iterations, starting from September 6, 2022, until December 16, 2022. This term has the development divided into three releases, the alpha release (September 6 to October 15), the beta release (October 15 to November 22) and the final release (November 22 to December 16). For the alpha release, each iteration focuses on adding new features and functionalities to strengthen the existing functional areas and use cases and updating the existing features to better accommodate the design goals for Swiftly. The eighth iteration focuses on implementing a rank system using data analytics, implementing advertisements for providing relevant resources to the user, and implementing the playground editor on the website so that domain experts can create and edit playground questions for their chapters. The ninth iteration focuses on improving the chatbots logic, increasing the variety of playground questions, and creating a new page for the website to allow users to browse their existing created chapters. The tenth iteration focuses on planning and implementing automated testing for the project, creating a dashboard view for evaluating student performances on the website, and improved UI for the signup view for better user experience. Finally, the Eleventh iteration, the final iteration before the alpha release focuses on updating the chapter content itself, applying content moderation for the application, improving the image selector on the website editor and implementing security steps on the cloud database for chapter creation. Once the Alpha is released, all the primary features are implemented, and the beta release focuses on testing and improving the existing features, before the final release. In these iterations, any use cases that were not able to be made by the alpha release are moved towards these iterations, which are otherwise just used for final touches and improvements for the project.

RISK MANAGEMENT PLAN

For the start of development up to the elaboration release, we tried to prioritize monitoring risks that would be detrimental to the final product in terms of user experience. This resulted in identifying risks that would affect the core learning experience of the end user. The biggest risks, as listed below, were related to the curriculum:

1. Unable to create a feasible and engaging curriculum for the user
2. Content too difficult or too easy to complete.
3. Swiftly is hard to integrate into the education sector

These curriculum risks are closely related to each other, but they've been separated into three different risks for the sake of simplicity and ease of understanding. An engaging curriculum would need to also strike a good balance in terms of how challenging it is. An engaging curriculum would also need to be appropriate enough for it to be used in classrooms. Specifically, Swiftly would need to offer content that does not conflict with what is being taught in conventional class. These curriculum related risks are being actively monitored and mitigation strategies are being planned. The main goals involve curating the best selection of content appropriate for the target audience and implementing it in a way that will take advantage of the technological means that is platformed on. These risks would need to be constantly assessed throughout development and even post development. In addition to the curriculum risks, there are many other risks that are experienced on the user side. Risks such as being unable to complete interactive questions, losing track of progress, application crashing, and invalid account management. These risks are more oriented towards the user itself and not so much the curriculum that Swiftly provides. Just like the previously mentioned risks, these ones too are being planned for and mitigation strategies are being put in the place to avoid such risk from ever occurring.

After the Elaboration release, the focus was on risks regarding the fundamentals and foundation of the final product. This risk is only an issue to current development as they only present themselves in the middle of the dev process. These risks include implemented components not behaving as intended and being unable to handle unexpected inputs. This includes functions such as account management, which includes validation, database calls, chapter content moderation, involving pulling and pushing data to and from Firestore, the chatbot services, which is expected to make API calls to IBM Watsons services, and website chapter content and playground editing functionalities. These risks were all considered when creating tests cases for the Software Plan, where they were exhaustively tested, evaluated and modified as needed. Finally, risks such as having to access a single IBM Watson account to update and expand the chatbot are risks that have been mitigated through future considerations, where instead of letting the risk be present, the ability to build on the chatbot catalog is mitigated until a safer and more reliable implementation utilizing cloud functions between Firebase and IBM services is implemented.

VALIDATION AND TESTING

This section of the report pertains to the validation and testing done for Swiftly to ensure no major faults or errors may arise. This section is split up into two subsections - the testing strategy subsection and the validation results subsection. These subsections will follow and comply with the test cases identified in the Swiftly Test Plan (STP) workspace in Jira.

TESTING STRATEGY

Because Swiftly is a multi-platform application, the test strategies that will be utilized will share a foundation but will differ slightly based on the respective platform. Regarding the iPad app, most test cases done will have to be executed manually as they require some form of intervention that is not achievable through software. These tests are primarily concerned with internet connectivity and remote database access. In the Swiftly iPad app, the test cases will be focused on user account management, accessing chapters, updating user progress, and chatbot integration. Regarding the Swiftly website, many of the test cases will follow a similar pattern to the app in terms of test cases that involve internet connectivity and remote database access. As previously mentioned, these tests involve a manual testing strategy as initiating the test case is best done through manual intervention at these initial stages as most failing scenarios are unknown to us. However, we have plans to take an automated approach in the near future by utilizing continuous integration/continuous deployment (CI/CD) by making test cases for the most common use cases on the website and hopefully in the app as well. As of this moment, testing the website has been mostly in the scope of its core features such as retrieving, creating, editing chapters and lessons.

For the alpha release and onward, unit testing was implemented into both the iPad application and Website using XCTestCase and the JEST testing library and the firebase-jest-mock library respectively. These unit tests allowed for testing each individual component in both the application and website in isolation, which allowed for easy resolutions if a certain test case failed. These unit tests were used to test every functional area and use case that was identified and implemented for the alpha release, with the entire process being described in the Swiftly Software Test Plan document.

VALIDATION RESULTS

The tests developed for the Swiftly platform proved to be extremely useful – particularly those tests which are considered automated. Due to the high complexity of the application and the high degree of data processing, developing tests and ensuring their validity was of utmost importance. The tests developed were executed successfully, allowing the development team behind this project to validate that functions and features tested are working as intended. This is crucial because of the many key areas of this application, it's especially important to API calls and data processing logic. In particular, tests on both the application and website that focused on pushing and pulling data from the cloud database proved to be essential, as it is the most crucial feature for overall functionality within Swiftly, so being able to automatically test the API calls saved crucial time that would have been otherwise lost troubleshooting.

CONCLUSION

In conclusion, the problem identified in this report is concerned with the lack of tools that students can use that will help bridge the gap between the fundamentals of iOS programming and actual iOS application development. This problem is coupled with iOS classes in middle and/or secondary schools, and so the proposed solution aims to fill this niche within the classroom setting where it can be used as a teaching aid. Through Swiftly, the fundamental issue of the identified problem was solved. This was done by creating a curriculum within Swiftly that provides users with chapters that cover a wide variety of topics within iOS. Currently, Swiftly only provides a handful of chapters, and so there are still many areas of iOS that the app can explore and provide content for. To provide the best possible experience and broaden the usage of Swiftly, many more chapters must be added. This can be done with ease through the Swiftly website – here content creators can develop and add chapters to the Swiftly curriculum on the fly, allowing for new content to be delivered to the users continuously. The links below provide a video demonstration of the Swiftly app and website.

Swiftly App: https://youtu.be/_YKaRrPzGZo

Swiftly Website: <https://www.youtube.com/watch?v=ldJgs7iebhU>

PROJECT SUITABILITY

To reiterate, the overall goal of Swiftly is to provide a tool that would help middle and secondary school computer science students bridge the gap between knowing the fundamentals of Swift and actual iOS development. This niche area contains information that is vital to iOS development and is more than just the syntax of Swift. Swiftly, at its current state, provides sufficient material and functionality to be deemed suitable for its original purpose. It provides users with a base set of chapters that covers fundamental Swift topics, as well as core iOS topics. Through both a theoretical and interactive approach Swiftly attempts to reinforce the details and significance of these topics. By being used in conjunction with iOS classes in the education setting, Swiftly can be used as a teaching aid where it will help assist students and/or clarify topics being taught in the class.

The creation process and decision-making behind the initial chapters was influenced by computer science class curriculums gathered from schools in Ontario. Taking this into consideration was critical as Swiftly needed to deliver content that was relevant to today's computer science courses so that it could be deemed as a feasible and suitable application to be used in the education setting. That being said, the domain expert for this project deemed Swiftly's vision as suitable and appropriate quoting: "I think this is a very narrow but relevant niche that does in fact exist in education in the sense that there are few resources (if any) that address this combination". Following this, when presented with the final demo of Swiftly, the domain expert had said "[Swiftly] has a good feel to it and I think the content layout works". In addition to this, the domain expert complimented Swiftly's ability to deliver new content on-demand as they said "I really like the setup between the app and the website and having the ability to update the content on the fly. I think this is a great option and provides the ability to have the content be responsive to the changing needs of the class". The reassurance from the domain expert gives merit to the feasibility of Swiftly as a

concept and a platform - especially because the domain expert is a secondary school computer science teacher, who teaches classes on iOS development.

Moving forward, the demo videos provided in this document demonstrate the functionality of the Swiftly app and website. By watching these videos, it's clear that Swiftly provides a functioning mobile and web-based application through which new content can be delivered straight to the user's device. In addition to this, the method for content delivery presented in the mobile application provides a balance between a theoretical approach, and an interactive / testing understanding approach. This will provide some diversity to the user experience allowing them to solidify their understanding of the topics more efficiently.

DOMAIN EXPERT EVALUATION

The domain expert evaluation for Swiftly is critical as it reflects the suitability of the project from an insider's perspective. Due to the importance of this, the resources provided to the domain expert for evaluation was a walkthrough video demonstration of the Swiftly app and website. The domain expert watched these videos and provided his thoughts on them. First, the domain expert complimented the relationship between the app and the website, especially how it allows new content to be added to Swiftly on the fly, quoting "I really like the setup between the App and the website and having the ability to update the content on the fly. I think this is a great option and provides the ability to have the content be responsive to the changing needs of the class". In addition to this, he compliments the usage of user progress data analytics within the app, stating that "The progress tracking from the user side in the app looks great and seems useful to know what parts have been completed". However, he also states that this area could be extended to the website app so content creators see which students have completed which chapters, "Features I could see becoming useful would be a progress tracker to see which students are completing which modules on a dashboard of sorts". The domain expert goes on to raise the question of the potential feature which would enable a sharing feature that provides a progress report to relevant parties, "Is there a sharing tool maybe that could be added to create a progress report that can be logged at certain intervals to give an update to interested parties?". Next, the domain expert goes on to compliment the user interface of the iPad app, albeit while recommending adding more graphical information, "The App has a good feel and I think the content layout works. There is an opportunity to provide some more graphical information I suppose". Finally, the domain expert recommended video lessons as part of the chapter contents, stating that "It would also be beneficial to add video lessons perhaps to the App to provide interactive content delivery to complement the text content." The feedback given by the domain expert is very useful as it highlights the strengths and weaknesses of Swiftly, and through this, it brings to light new ideas that will only make the project better.

USER TESTIMONIALS

Due to limited testing available and lack of knowing direct stakeholders (other than the domain expert), no user testing was done outside of the Swiftly development team.

FUTURE WORK

While Swiftly achieves its goal of being a teaching tool that provides users with the information and evaluation tools to learn and test their understanding of application development, and content creators with the tools to create lesson content for the main application, there are still many more features that can be iterated upon and implemented to improve the experience for the students using the application, the teachers using the application as a teaching tool and content creators creating the chapters.

Iterating further on the flexibility of lessons and interactive questions is another area of consideration for future implementation. For the alpha release, each chapter in Swiftly consists of a knowledge section containing text and images, and an interaction playground section containing multiple choice, true and false and code-blocks type questions. For future releases, there is the potential to introduce new types of questions for the interactive section and new ways of delivering knowledge in the knowledge sections. This could include embedded videos in the knowledge section, IDE simulations for interactive questions, and more. This will not only give content creators more flexibility in how they teach and deliver the information in their chapters but provide the users with a better curriculum overall.

Based on our long-term plans for greater flexibility for Swiftly, we also aim to provide a more comprehensive and rich feature set for the chapter creator suite. This will make the expert's user experience of creating and publishing chapters a more streamlined experience. We also aim to implement a few analytics based features for the creator website. Specifically, the ability for experts to analyze the various performance statistics doing their courses. We believe giving features like this would provide valuable feedback to the expert so they can take it into consideration future iterations and updates to chapters they publish. In addition to the analytics feature, we also want to implement a community page within the website so that experts can browse chapters that are publicly available to the rest of Swiftly. Experts can choose to allow their chapters to be featured in this page by toggling their chapter's visibility setting

A future implementation to consider is regarding the risk of the quality of the chapter content being uploaded to the database using Swiftly Expert. As it is implemented right now anyone can upload whatever they want to the main application, regardless of the subject and quality. A content moderation and validation system where the content is put on hold from appearing in the main application until approved by verified users would address this current risk. Another feature under consideration that could achieve this is a system to pre-screen check the content being uploaded to ensure it is relevant to a chapter's subject matter. This would ensure that the only chapters being made available for users to access are chapters that are deemed relevant and qualified for the design goal of Swiftly.

Finally, in future releases there is the idea of introducing programming languages and application development platforms beyond Swift and XCode. Swiftly initially targeted Swift as a programming language and XCode as the IDE for application development considering the increased usage of smart devices including iPads, and existing software that teaches the basics of Swift such as Swift Playgrounds. However, with the way Swiftly has developed into an open-ended platform, we've found that it doesn't have to be restricted to providing education content for just Swift. This opens the opportunity to introduce a greater variety of languages and will give our student users more opportunity to learn concepts that are relevant to their existing curriculum from a more broad and diverse content

pool. Teachers would also be able to use Swiftly to complement their existing curriculum for any computer science and application development-based courses.

BIBLIOGRAPHY

- [1 Statistics Canada, "NAICS 2017 Version 3.0 - Classification structure - 61111 - Elementary and secondary schools," 17 March 2021. [Online]. Available: <https://www23.statcan.gc.ca/imdb/p3VD.pl?Function=getVDStruct&TVD=1181553&CVD=1182006&CPV=61111&CST=01012017&CLV=2&MLV=5>. [Accessed 9 December 2021].
- [2 Government of Canada, "Elementary and secondary schools - 61111 - Summary - Canadian Industry Statistics - Innovation, Science and Economic Development Canada," Government of Canada / Gouvernement du Canada, 1 March 2019. [Online]. Available: <https://www.ic.gc.ca/app/scr/app/cis/summary-sommaire/61111>. [Accessed 9 December 2021].