SSO PROJECT REPORT

CS4850 W03

David VanAsselberg, Vincent Cradler, Fidel Ngo, Rene Lisasi

Website Link: https://protect-sso.github.io/SeniorProdWeb/

GitHub link 1: https://github.com/Protect-SSO/SSO-website

GitHub link 2: https://github.com/Protect-SSO/SSO-API

GitHub link 3: https://github.com/Protect-SSO/RedirectApp1

12/3/2023

David 

Rene 

Vincent 

Fidel 

FINAL SUMMARY

This document is a combination of the Requirements, design, test plan/report, and project plan all in one about our single sign on application. Just to summarize, this single sign-on application lets its users authenticate themselves onto multiple different independent software systems usings a single JSON web token. The SSO Website application creates a simple and convenient system for employees to log into applications necessary for them to do their job. Ideally the employee will be able to log in once to our interface, and then have access to all their work apps without needing to log in. This also allows organizations to add new apps to the SSO dashboard for users to have access to. The requirements document will cover all the functional and nonfunctional requirements for the project requested by the stakeholders of the project. The design portion represents how we built the application using abstract UML diagrams such as the sequence diagram, deployment diagram, entity relationship diagram, model view controller diagram. It also contains detailed explanations on how we built it and how we designed our program to protect against certain security risks. The project plan portion talks about how we worked together to put this project together. The test plan talks about how we went about testing our application for errors.

SP-30 PROJECT RQUIREMENTS

PROJECT OVERVIEW:

Project Title: Single-Sign-On

Project Team: David, Rene Lisasi, Vincent Cradler, Fidel

1.0 INTRODUCTION

1.1 Purpose

The purpose of the Single Sign On requirements document is to go over every functional, non-functional, and interface requirements needed within the application. The design constraints section is divided up into three sections: environment, user characteristics, and system. In the functional requirements section we will go over every feature the website must provide and the non-functional requirements will cover all things security. In the interface requirements section, we will go over everything that the user will need to interact with the website.

1.2 Project Goals

The goal of this application is to create a simple and convenient system for employees to log into applications necessary for them to do their job. Ideally the employee will be able to log in once to our interface, and then have access to all of their work apps without needing to log in. We also want a way for organizations to add new apps to the SSO dashboard for users to have access to.

1.3 Definitions and Acronyms

- SSO: Single sign-on
- API: Application Programming Interface
- SSL: Secure socket layer
- SQL: Relational database query language
- AWS: Amazon Web Services

1.4 Assumptions

That the Users' applications will eventually be registered into our domain. We also assume that we will be the first point of contact for a lot of information that the companies will give us. Due to the nature of the Web Application constantly, constantly updating itself and its many different components. User may lose information by attempting to update an isolated component.

2.0 USER CONSTRAINTS

2.1 User Characteristics

Should be used by companies that would like to manage their users' access to many different applications. System admins will be able to add an application and enroll users programmatically, and as users come and go to the corporation, they are given the correct privileges to all required groups of applications.

2.2 System

Users must have access to computer with modern browsers such as chrome, Firefox, edge, etc. The browser must have cookies enabled. The system must have access to a secure internet connection.

3.0 FUNCTIONAL REQUIREMENTS

- Users should be able to log in.
- Have access to some organization’s apps.
- Be able to request access to apps.
- Different looking test apps
- Database should persist data and test app data.
- Users should be able to add apps to okta within the organization.
- There should be different types of users.
- When users log in, they are given a json web token.
- Got to have a domain name.

On application level, there will need to be a few features. In order for a user to be registered with our SSO website, they need to be brought into an organization registered with the website. In order for that to happen, there needs to be different categories of users. For now, there should be admin users and base users. The admin users should be able to add other users to the company and add new applications to the organization SSO application list. All users should be able to log in, request access to apps they don’t have access to, and authenticate with app on their dashboard without needing to enter in a username and password. When a user logs in they will have a json web token placed as a cookie on their browser that will contain all the info needed for authorized applications to log them in.

4.0 NON-FUNCTIONAL REQUIREMENTS

4.1 Security

- SSL
- Hashed/salted passwords
- SQL injection
- XSS
- Token stealing
- Two step auth
- Log SQL queries

The above list is everything we need to protect against or implement. We need SSL, hashed/salted password, two step authentication, and log SQL queries. We need to protect against SQL injection, cross site scripting, and token stealing.

5.0 EXTERNAL INTERFACE REQUIREMENTS

5.1 User Interface Requirements

UI must be user friendly, minimal, and easy to navigate. Incorporating seamlessness and some modern design trends is the goal, but we'll take less. The main focus is for it to run, anything else is nice to have.

With that being said, we are function focused, so our interface requirements are minimal. They are to:

- Provide users with feedback.
- Provide consistent error messaging.
- Provide clear instructions.
- Make navigation seamless.
- Most importantly, keep it simple.

5.2 Software Interface Requirements

Fast, responsive data transfer is a key component to this project. The software should be lightweight yet secure.

- Reliability
    - Fault tolerance without breaking the entire system.
    - Availability meaning not being unavailable during high volume.
    - No combination of actions leads to the breaking of the system.

        - This would involve garbage collection.
- Performance
    - Load times should be minimal even in high volume.
    - Database access should not cause a bottleneck.
    - Core functions' speed should be prioritized.
- Usability
    - The software should be easy to use.
    - Instructions on how to use it should be clear and consistent.
    - It's easy to fall into the trap of making decisions that benefit the developer but go against the user's ability to learn how to use the software quickly without prior knowledge.
- Scalability
    - When designing this software, our aim is for it to not only be functional but also be scalable. That means we would have to build it with modularity. Each piece would have to be contained on its own like a Tesla battery.
    - To achieve this, a microservices approach would be optimal because it allows for incoherent parts of the system to be replaced to handle larger volumes without needing to rebuild the entire system from the ground up.
    - Thus, saving on development cost and time.
- Security
    - Authentication is our primary source of security.
    - Any unauthorized users should be prevented from accessing a site they should not have access to.
    - The tokens should be encrypted so that no one can intercept the tokens to and from their destination.

SP-30 DESIGN

1.0 INTRODUCTION

1.1 Purpose

The purpose of this document is to show the design of our system and all its interconnected components. This document also goes over security concerns and how we plan to deal with them. Towards the end of the report, there are some UML diagrams to show an abstraction of our infrastructure and data flow.

2.0 DESIGN CONSIDERATIONS

2.1 General Constraints

There will be no hardware requirements. A WebUI will serve as the end user environment. All we will require is a device for website access with moderate specs, a minimum of 4GB of RAM is required, anything that can run Google Chrome can access our site. All of the user data will be stored, backed up and managed by us. Our Organizational end user will be in charge of managing internal permissions after adding all of the own internal users. Applications that our Organizational users want to manage and would want to manage can be added to our integration network. The integration network is a central location that enables our Organizational users to add sites to their portal for their end user.

To properly test we must ensure that our API doesn't allow for unauthenticated resource access, period. In addition to that, all Organizations and their respective users should be siloed from each other. Any user part of an organization should not be able to get information about another user without the necessary permissions.

3.0 SYSTEM ARCHITECTURE

3.1 Components

3.1.1WEBSITE

The website is the main component users interact with and is the main product we are building. The website gives users access to all of the features Single sign on provides, but it is not what provides the services. The website just uses the SSO services.

3.1.2 AUTHENTICATION API SERVICE

The Authentication API is what provides the SSO services. The website will make requests over to the Auth API for services such as logging a user in, registering an organization, registering a user, and adding applications. It will process the requests and then respond accordingly based on the service being requested.

3.1.3DATABASE

The database will store information about users, organizations, applications and who is authorized to use certain apps.

3.2 Interface

3.2.1 AUTHENTICATION

There are several pages related to authentication. There is a registration page where an organization can be registered with our SSO website. With that, a user that represents the owner is created. This owner account will be able to register other users with the organization through the UI. Then there is a page for logging in where a user can enter in their username and password which will lead to a page to enter in a code that was sent to the user's email. Once authenticated the user is redirected to the landing page.

3.2.2 LANDING PAGE

The landing page is where users can see the apps, they have access to. There is a nav bar that will lead them to several other services.

3.2.3 NAVBAR:

- All Users
    - Dashboard
    - Request access to other applications
- Admin / Owner
    - User registration page
    - App registration page

The app registration page contains a place where you can enter the name of the application and a redirect URL. The request app access page will have a list of apps your org has with a request access button on each app.

3.3 Security design

- SSL
- Hashed/salted passwords
- SQL injection
- XSS
- Token stealing
- Two step authentications
- Log SQL queries

The list contains things we either need to protect against or what we will use as protection. Our first challenge to tackle is how to transfer sensitive information across the network without people reading what it says. Solution: Use SSL. An SSL will encrypt any data that goes through so anyone using a program like Wireshark to look at data going through a network, all they will see is encrypted text. Next, we need to protect sensitive passwords in the database from attackers that have gotten access to the database. To do this, we will hash the passwords and put them through 10 rounds of salting. Next, we need to protect against SQL injection and Cross site scripting, so what we will do is escape the inputs using EJS. On the client and server side we will also limit what the user is allowed to enter. Our next threat is token stealing. In this situation, the token is refreshed daily so the user who steals the token has a short window to operate if they do manage to steal the token. All users will have to enter a code sent to their email when they initially log in. To protect against malicious employees looking at the data in the database and making changes, we will log all SQL queries with the user that did it so actions can be reversed. to maintain integrity of data

3.4 Architectural Styles

3.4.1 MVC

The model view controller architecture is used in this project because the application is a web application. Most web applications are based on the model view controller. For our Single sign on website, the controller is the part of the web server, and it takes in requests from user's browsers, does some functionality, and sends a response to the client in the form of an EJS page. It controls where the user navigates based on the client's requests, The view for this application is in the form of EJS pages. EJS is basically just HTML but with some extra features. The controller serves the EJS pages to the users to view and interact with the application. The model is configured in Express.js, the module we are using to create the webserver, and it will form the view to be served to the user's browser.

3.4.2 CLIENT SERVER

This Architecture is a given for this application. In this application, there are 2 clients. The first client is the users browser. The browser sends requests over to the web server, the web server processes the request and sends a response back to the users browser. The second client is the website. The website make requests over to the Auth API service which processes the request and responds back to the website in the form of JSON objects.

4.0 DIAGRAMS

4.1 MVC class diagram

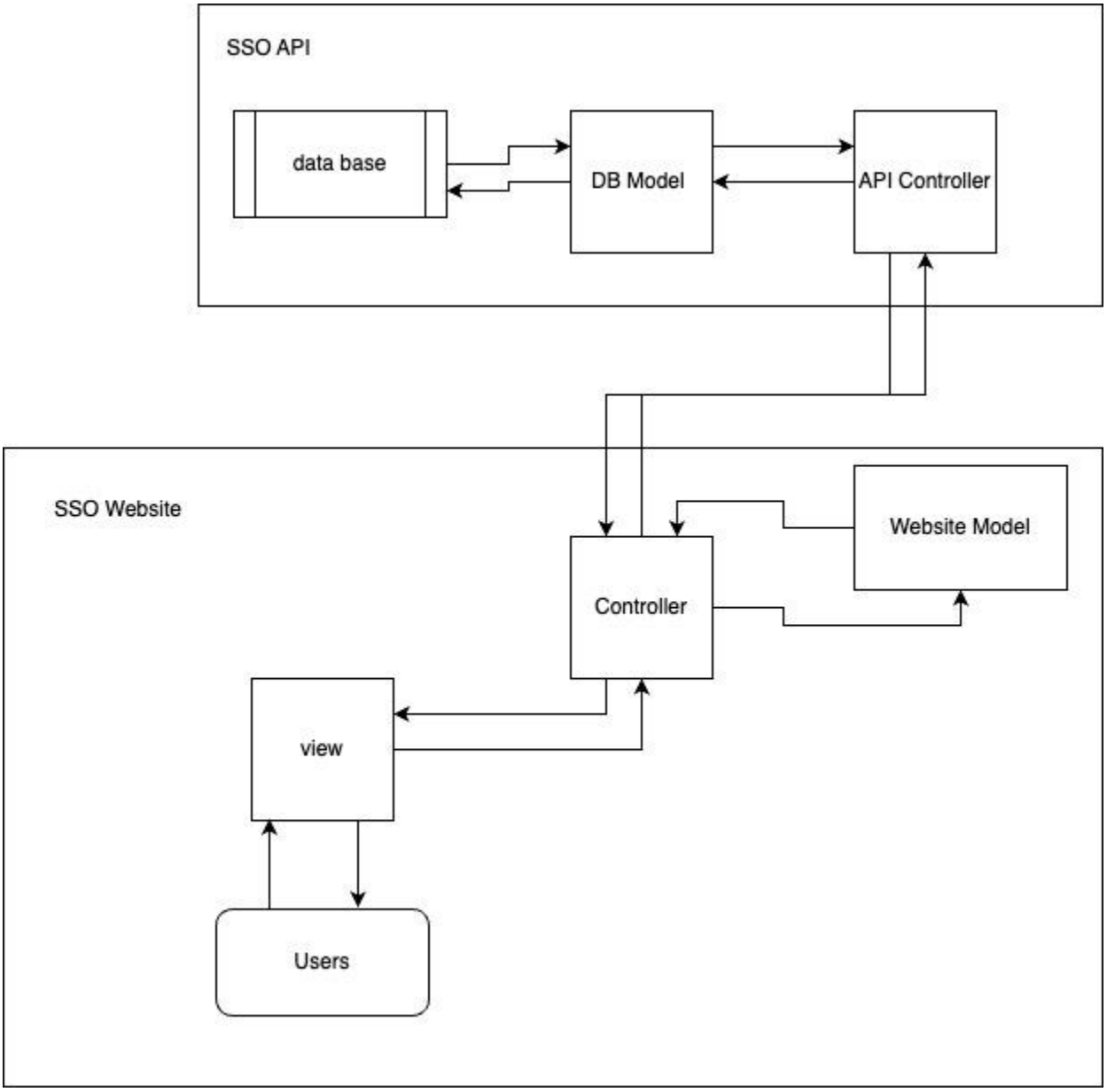

FIGURE 1: MVC

The diagram above represents the MVC architecture we are using. The view for our app is an EJS page. It's essentially HTML but with some extra features. The controller is the web server, and it serves these EJS pages to the client's browsers. The controller also requests services from the Auth API service. The model is part of the webserver built in Express.js and forms the EJS page before it is served to the user's browser.

4.2 Entity relationship Diagram

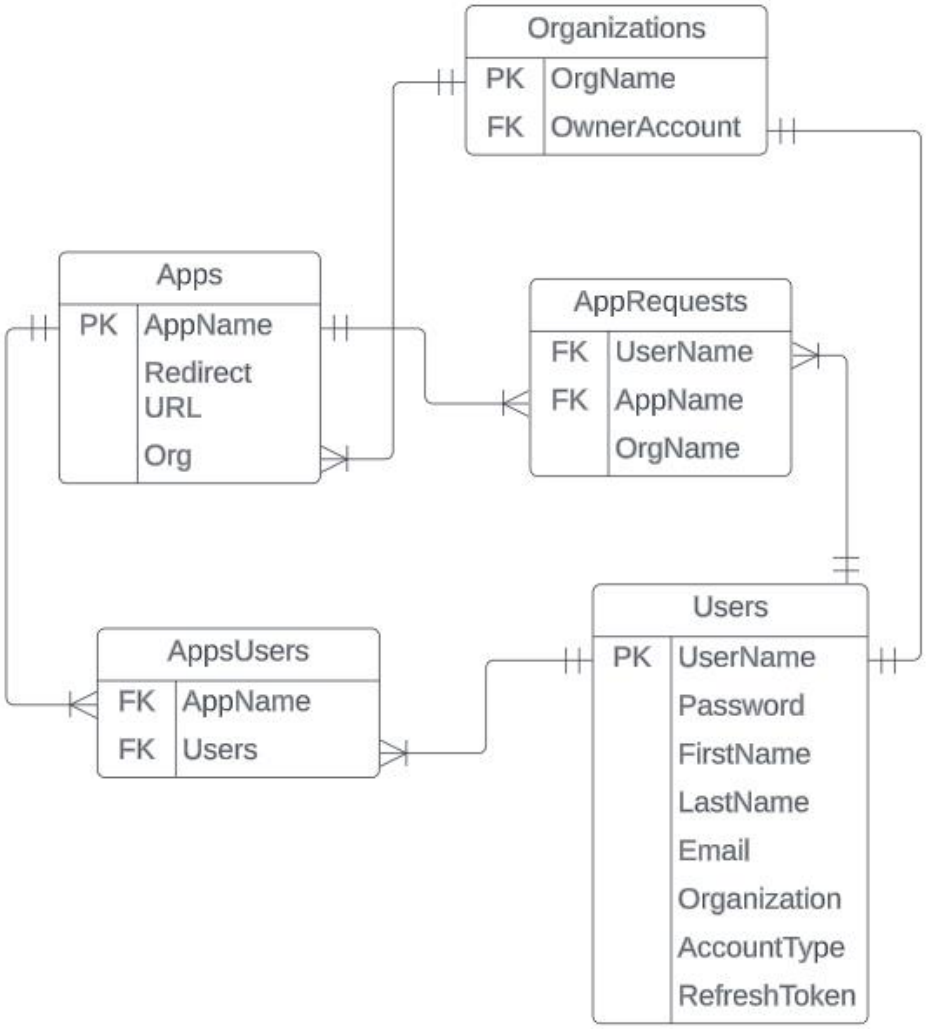

FIGURE 2: ER DIAGRAM

The ER diagram represents the relationships between the tables in the database. There are four tables contained in this ER diagram: Apps, AppsUsers, Users, and Organizations. The apps database will store different apps, each app containing an app name, redirect URL, and an organization. Each app will have multiple AppsUsers, which are users signed up with that app. The AppsUsers table contains the appname along with the user of that app. Users are the users of the SSO application, all the users of the SSO application will be stored in the user table. Every user in the SSO application requires a username, password, first name, last name, email, organization, account type, and a refresh token. A user can be registered for many apps in the AppsUsers table since users can be an app user of multiple apps. The organizations database

contains an Orgname and an OwnerAccount. Every organization requires one owner account and an organization can have many apps under its ownership in the apps table.

4.3 Sequence Diagrams

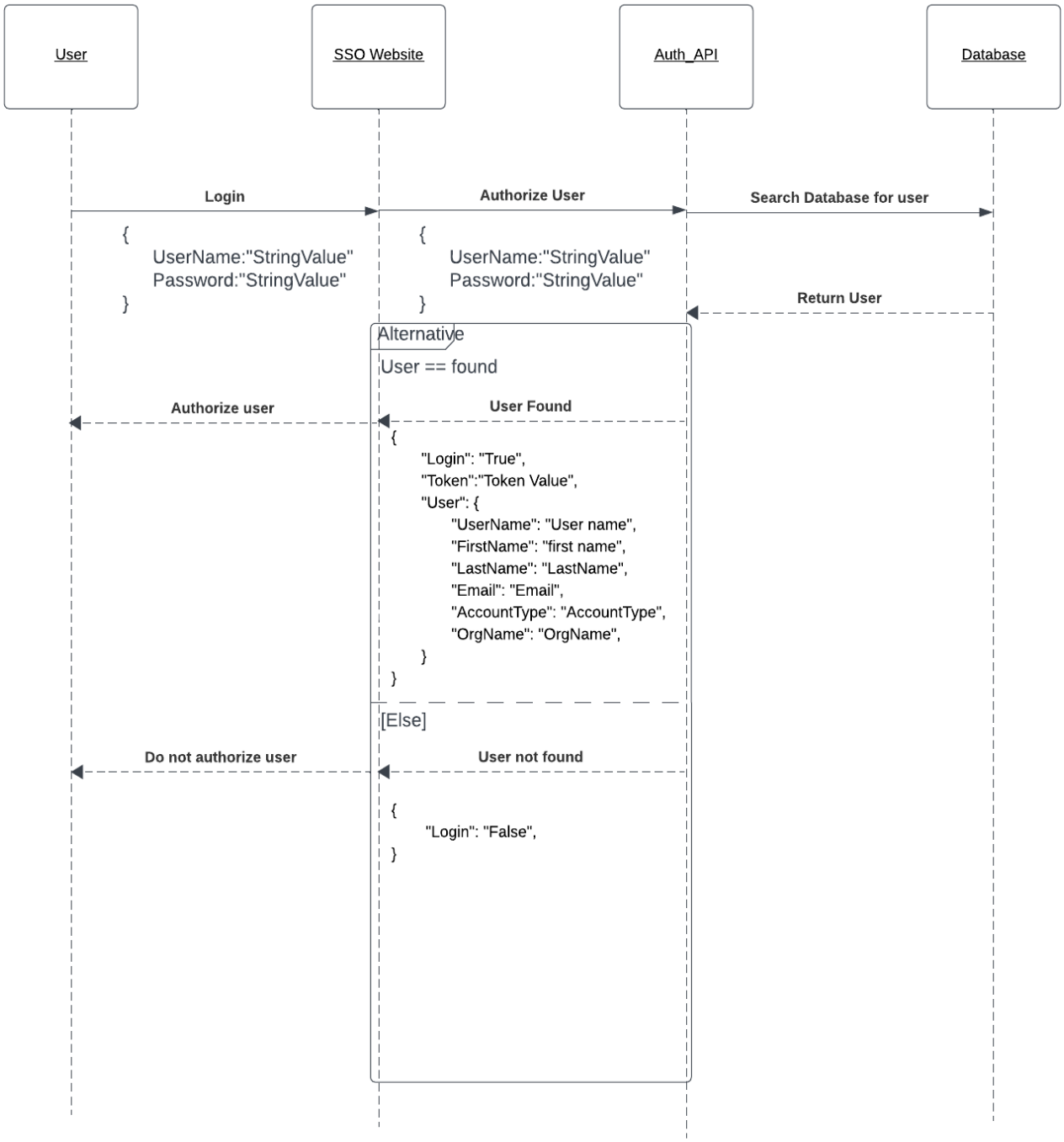

FIGURE 3: LOGIN SEQUENCE

In this scenario, the user requests to log in with its credentials. The SSO website will make a request to Auth API to log the user in sending it the users credentials. The Auth API will check the database to see if user exists. If user doesn't exist, the Auth API will send a failure message, if user does exist the Auth API will send a success message with the users info and token to the SSO website. If SSO website receives a failure message, the user receives a failure error. If success message is received the user gets logged in.

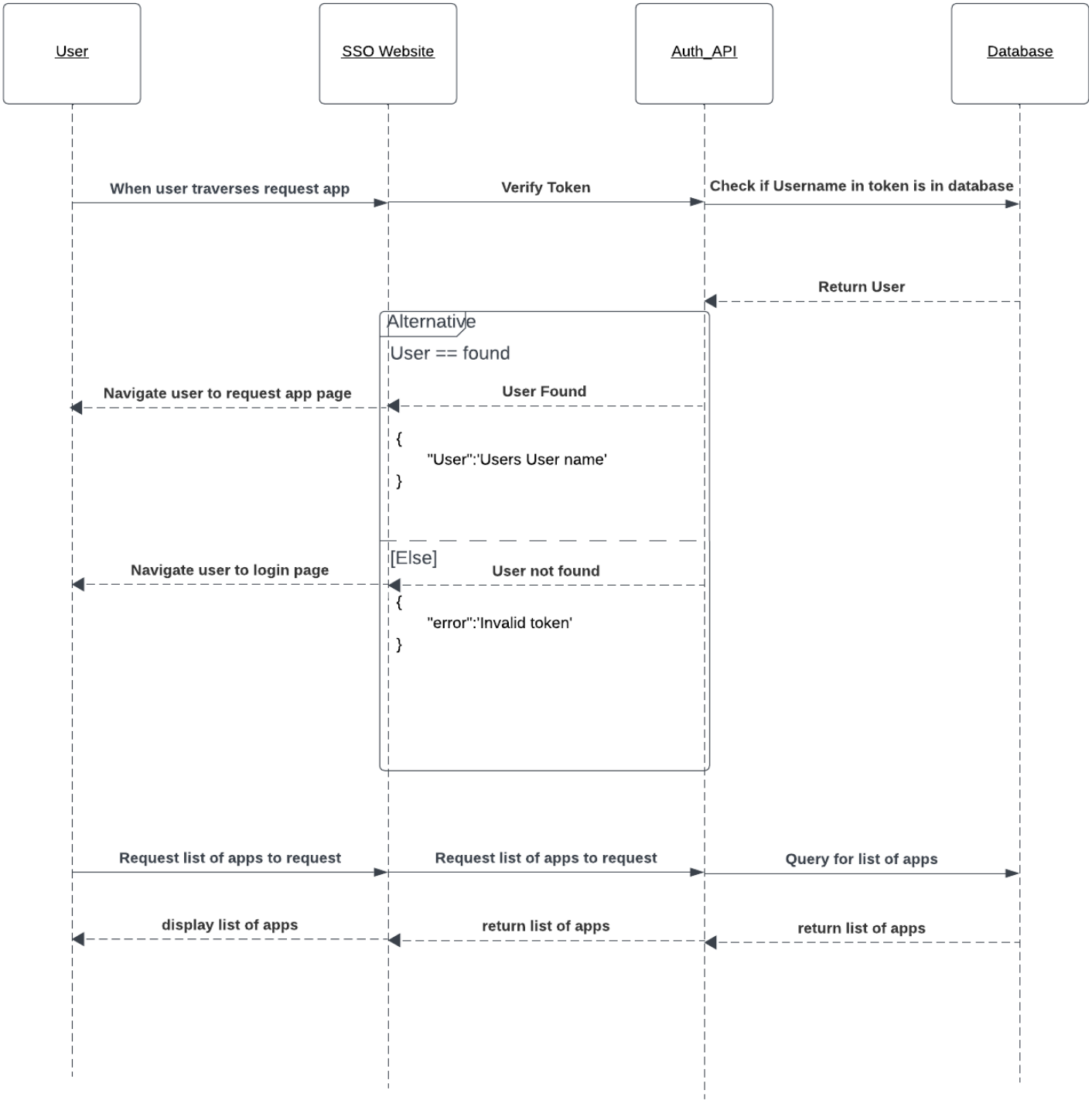

FIGURE 4: REQUEST APP PAGE SEQUENCE

In this scenario, the user makes a request over to SSO website to go to request app tab. The SSO website runs a middleware function that verifies the users token before redirecting the

user. That function makes a request over to the Auth API that verifies the token, it also asks the database if the user in the token is registered in the database. If verified and registered the Auth API sends a verified message over to the SSO website with the users username. If not verified of found then the Auth API sends over an invalid token message over to SSO website. If SSO website receives an invalid token message then the user gets sent to login page. If SSO website receives a success message then the user is sent to the request app page. Once on the page, the SSO website will request the list of applications to be displayed to the page from the Auth API. The Auth API will find the list of apps in the database, and return them to the SSO website. SSO website displays apps to the page.

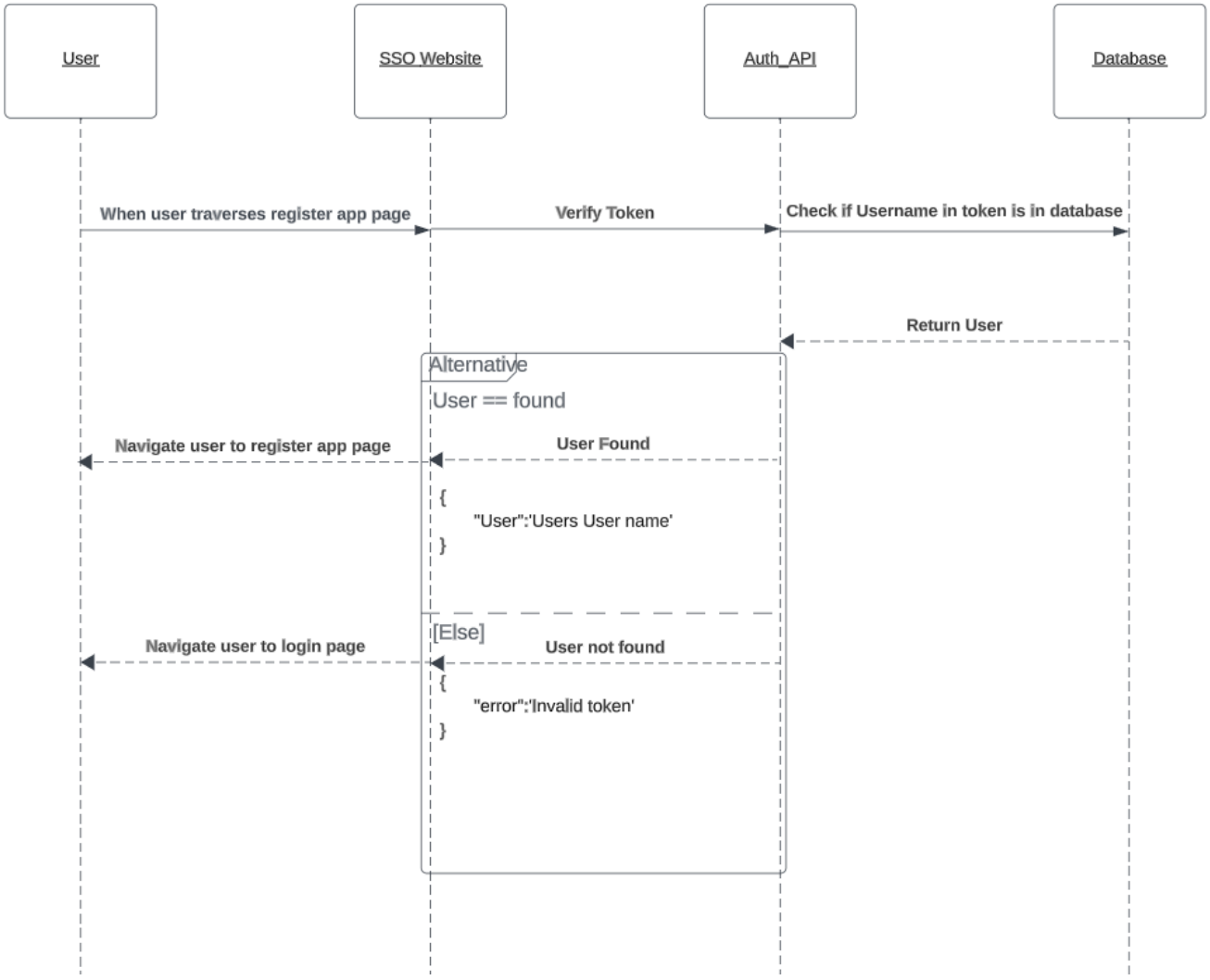

FIGURE 5: REGISTER APP PAGE SEQUENCE

In this scenario, the user makes a request over to SSO website to go to register app tab. The SSO website runs a middleware function that verifies the users token before redirecting the user. That function makes a request over to the Auth API that verifies the token, it also asks the database if the user in the token is registered in the database. If verified and registered the Auth API sends a verified message over to the SSO website with the users username. If not verified of found then the Auth API sends over an invalid token message over to SSO website. If SSO website receives an invalid token message then the user gets sent to login page. If SSO website receives a success message then the user is sent to the register app page

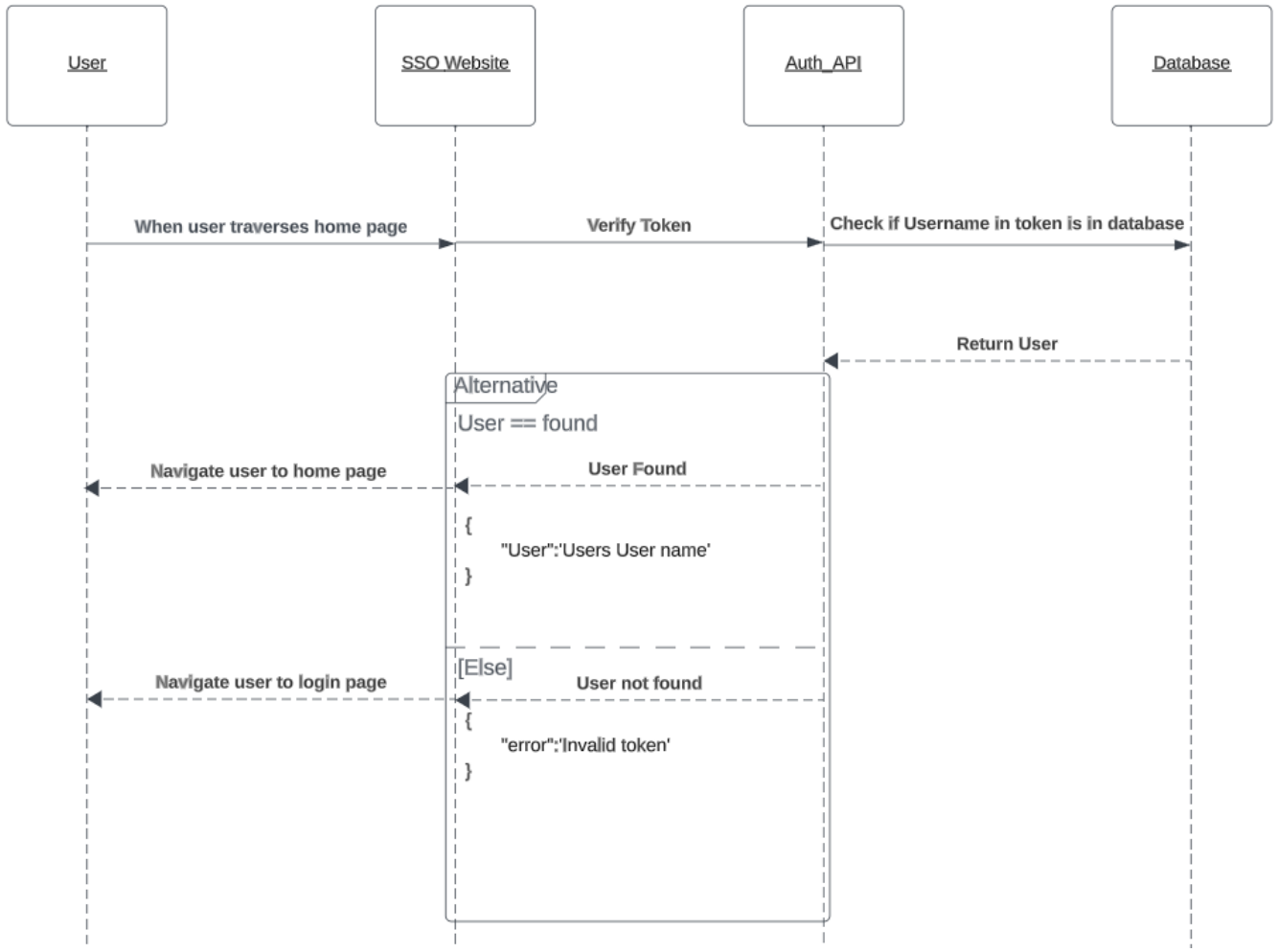

FIGURE 6: HOME PAGE SEQUENCE

In this scenario, the user makes a request over to SSO website to go to Home tab. The SSO website runs a middleware function that verifies the users token before redirecting the user. That function makes a request over to the Auth API that verifies the token, it also asks the database if the user in the token is registered in the database. If verified and registered the

Auth API sends a verified message over to the SSO website with the users username. If not verified of found then the Auth API sends over an invalid token message over to SSO website. If SSO website receives an invalid token message then the user gets sent to login page. If SSO website receives a success message then the user is sent to the home page.

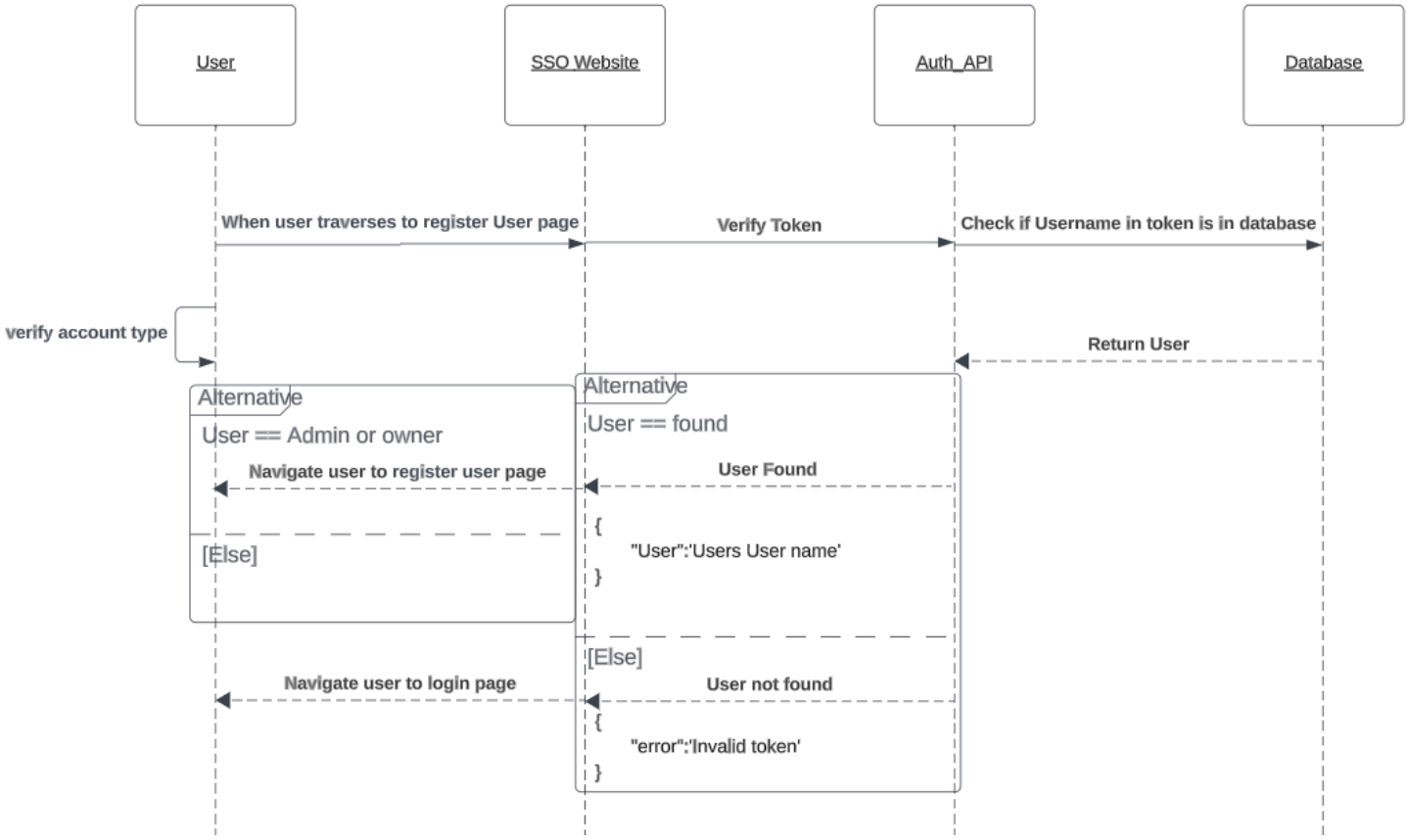

FIGURE 7: REGISTER USER PAGE SEQUENCE

In this scenario, the user makes a request over to SSO website to go to user register tab. The SSO website runs a middleware function that verifies the users token before redirecting the user. That function makes a request over to the Auth API that verifies the token, it also asks the database if the user in the token is registered in the database. If verified and registered the Auth API sends a verified message over to the SSO website with the users username. If not verified of found then the Auth API sends over an invalid token message over to SSO website. Then the SSO website checks to see if user is an Admin or Owner account. If so then the user gets sent to user register page, if not they get sent to the login page. If SSO website receives an invalid token message then they also get sent to login page.

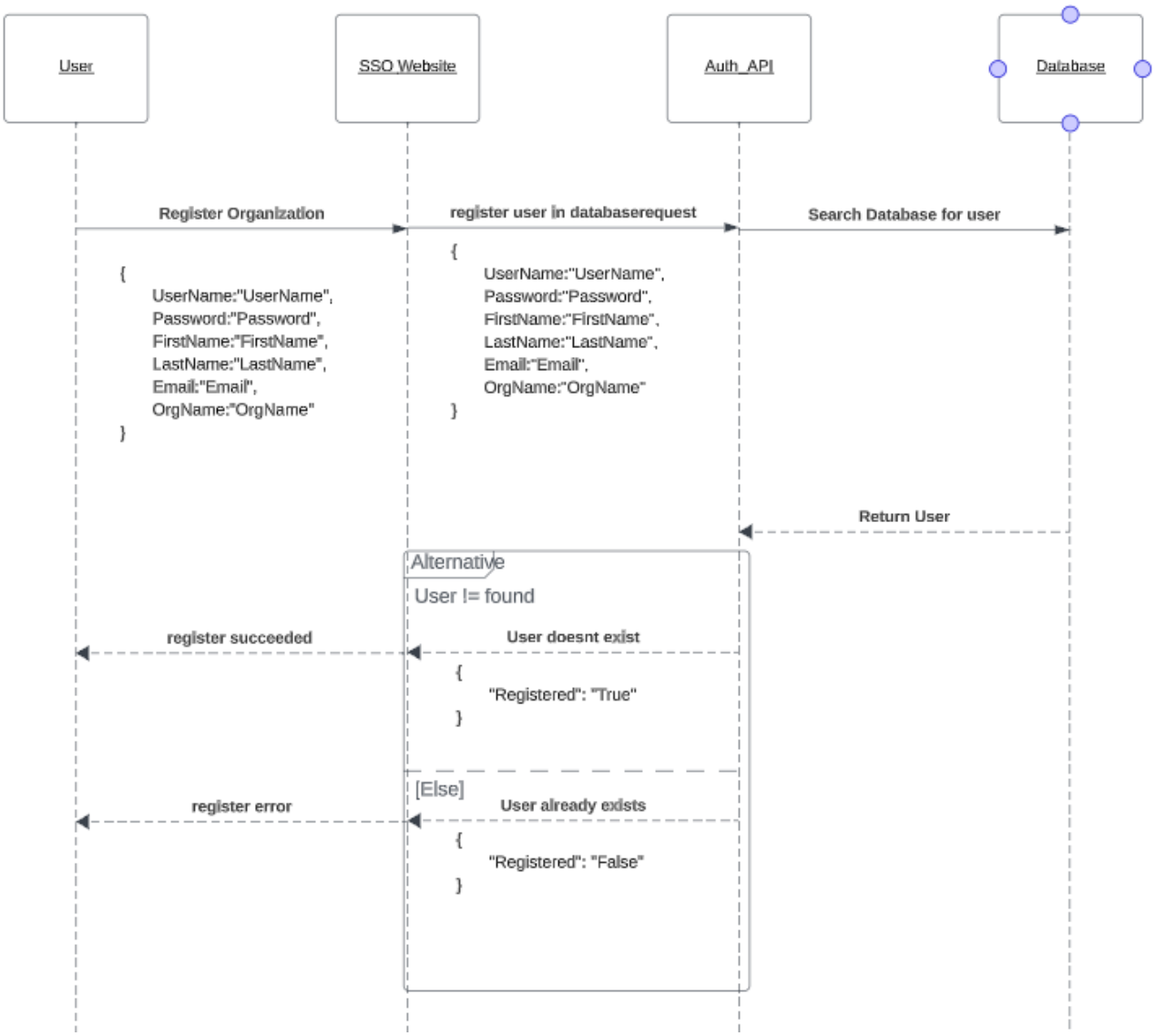

FIGURE 8: ORG REGISTRATION SEQUENCE

In this scenario, the user will make a request from the browser to the SSO website requesting to register an organization. The SSO website will request that the org and user be registered while the name of the organization and the owners user information in the form of JSON. Auth API will search the database for an existing user for redundancy. Database returns the user if any. If none then the Auth API will register the organization and return a success message to the website. If there is an existing user then it will send a failure message over to the SSO website and not register the user and org. The SSO website will redirect over to a success page or return an err based on what message it receives from the Auth API.

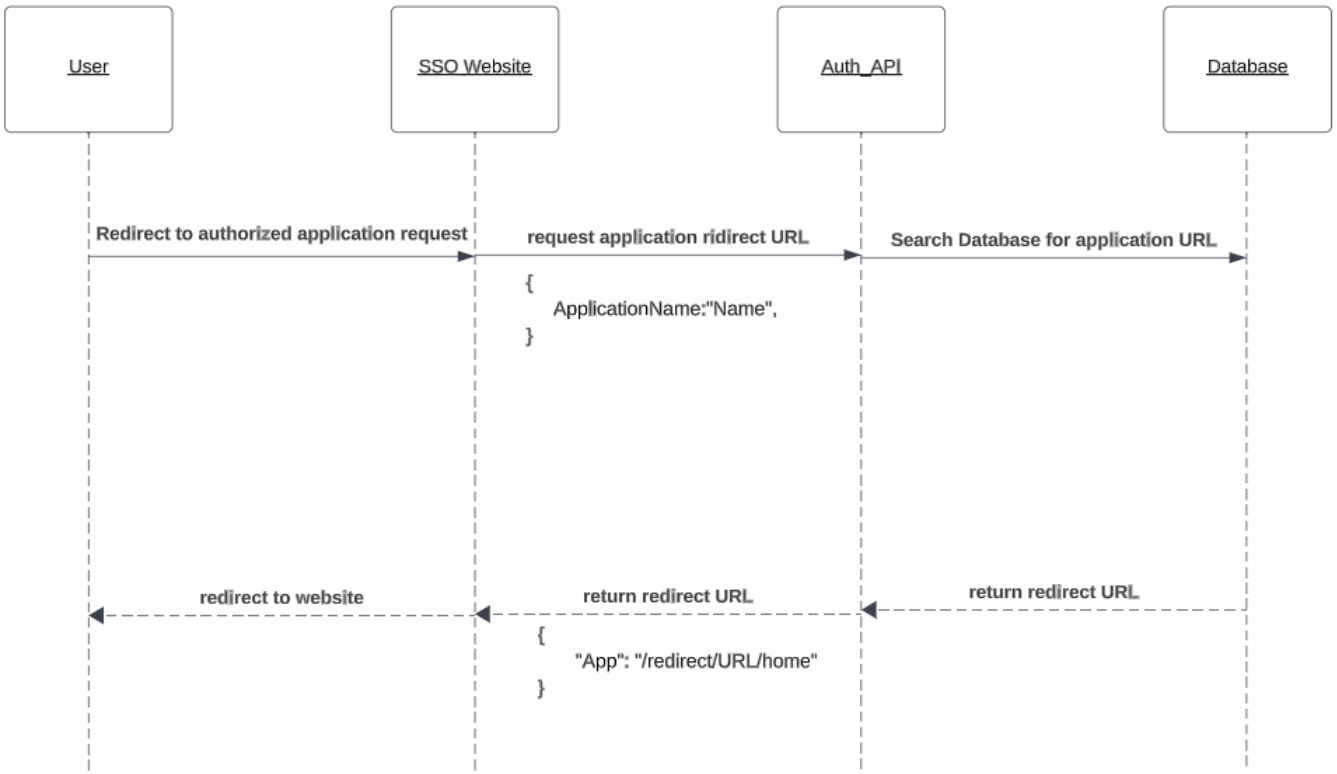

FIGURE 9: REDIRECT TO AUTHORIZED APPLICATION SEQUENCE

In this scenario, the user will make a request from the browser to the SSO website requesting to redirect to another website its authorized to use. The SSO website will request the redirect URL from the Auth API while providing the name of the app it wants to redirect the user to in the form of JSON. Auth API will search the database for the redirect URL that goes with the name of the app. Database returns the redirect URL. The Auth API will give the SSO website the redirect URL in the form of JSON. The SSO website will redirect the user to the requested website.

4.4 Deployment Diagram

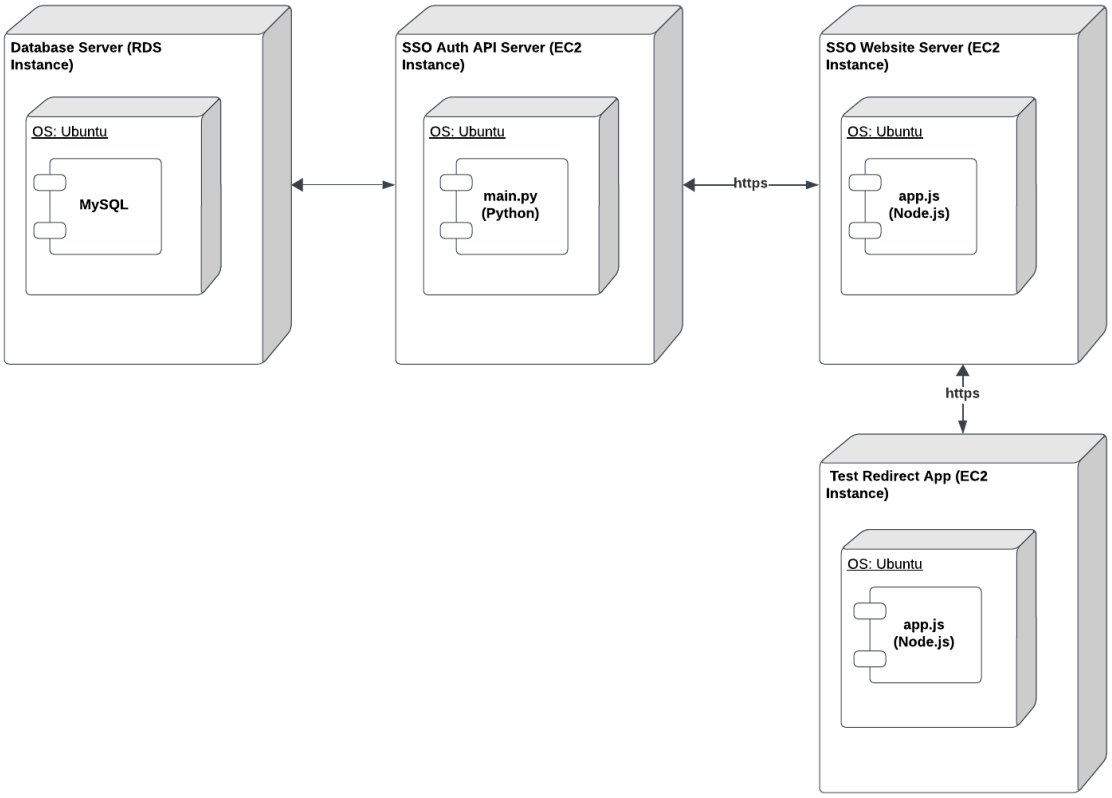

FIGURE 10: DEPLOYMENT DIAGRAM

The figure above shows the deployment of our system and how communication is handled amongst components. The database is hosted on an RDS instance in AWS. RDS is just a service provided by AWS that lets you pick what database you want access to from the cloud. In our case we chose to use MySQL. The database communicates with the Auth API which is hosted on an EC2 instance on AWS. All EC2 instances discussed about are using the t2 micro tier and the operating system is ubuntu. The Auth API communicates with the SSO website using the http protocol, with Json data representation. The SSO website is also hosted on an EC2 instance and it is the one that initiates the http requests over to the Auth API. When prompted, it will also redirect users over to test applications. These test applications are also hosted on AWS on an EC2 instance.

4.4 User Interface design

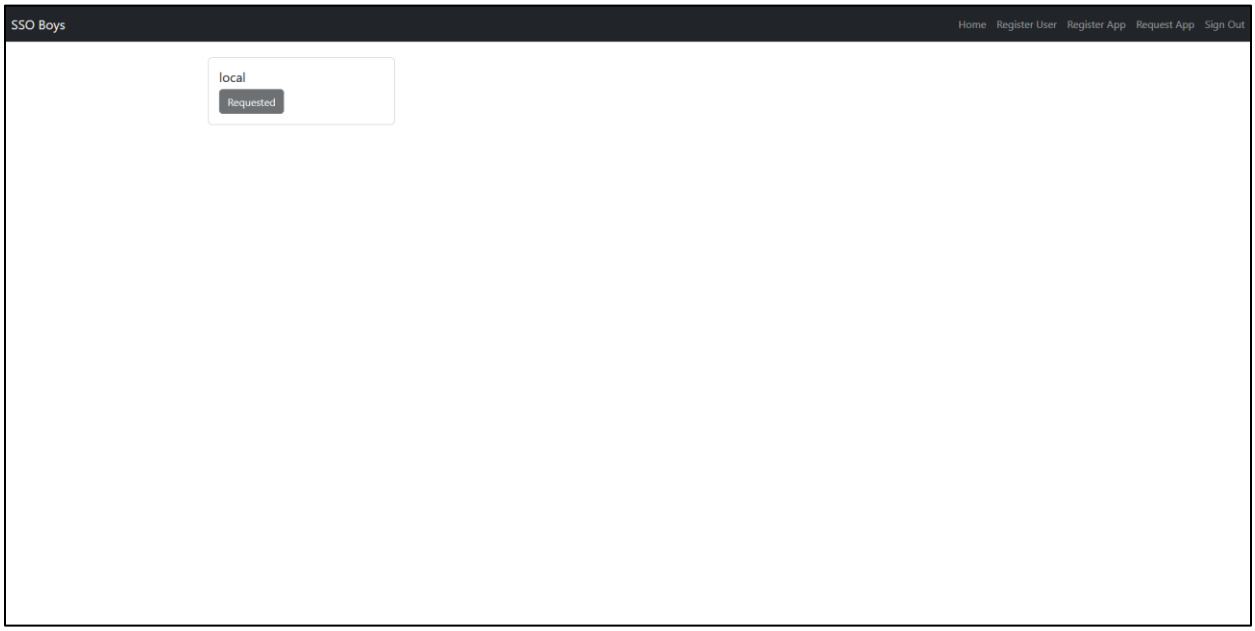

FIGURE 11: REQUESTED APP PAGE

This page represents the request app page. This page has a number of apps on it that are registered with the organization. When the register button is clicked it will request the app to be registered with the user for a support account to accept for the user.

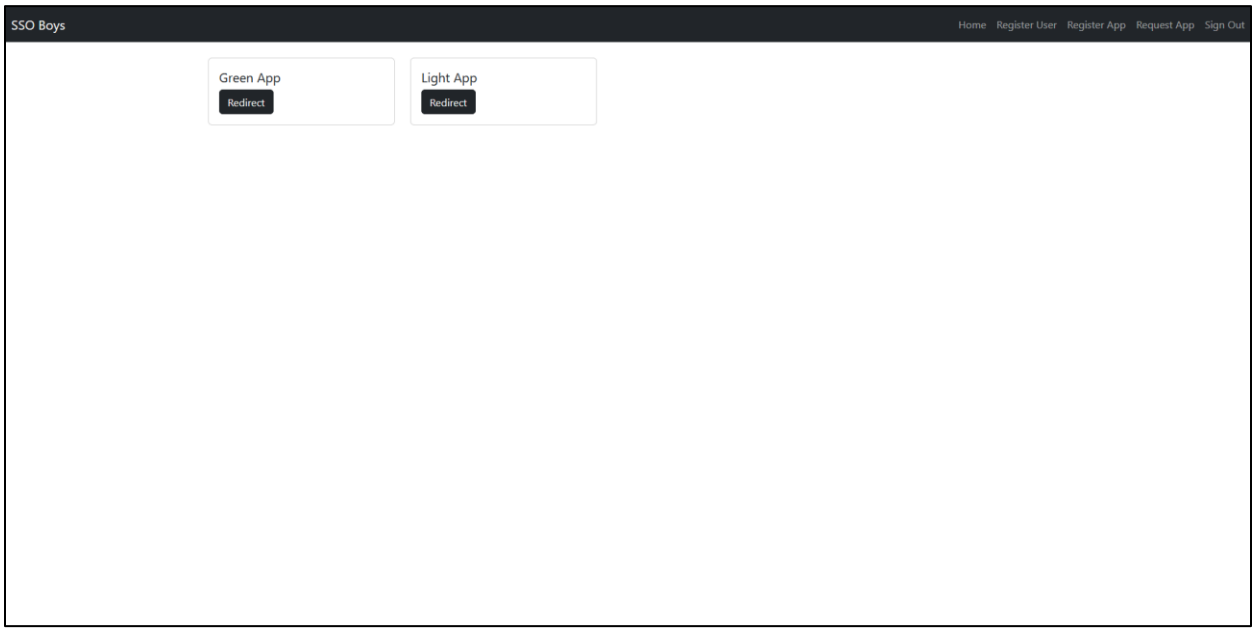

FIGURE 12: DASHBOARD

The dashboard is where users can see all the apps they have access to. When the click on an app they have access to, it will redirect them to that app.

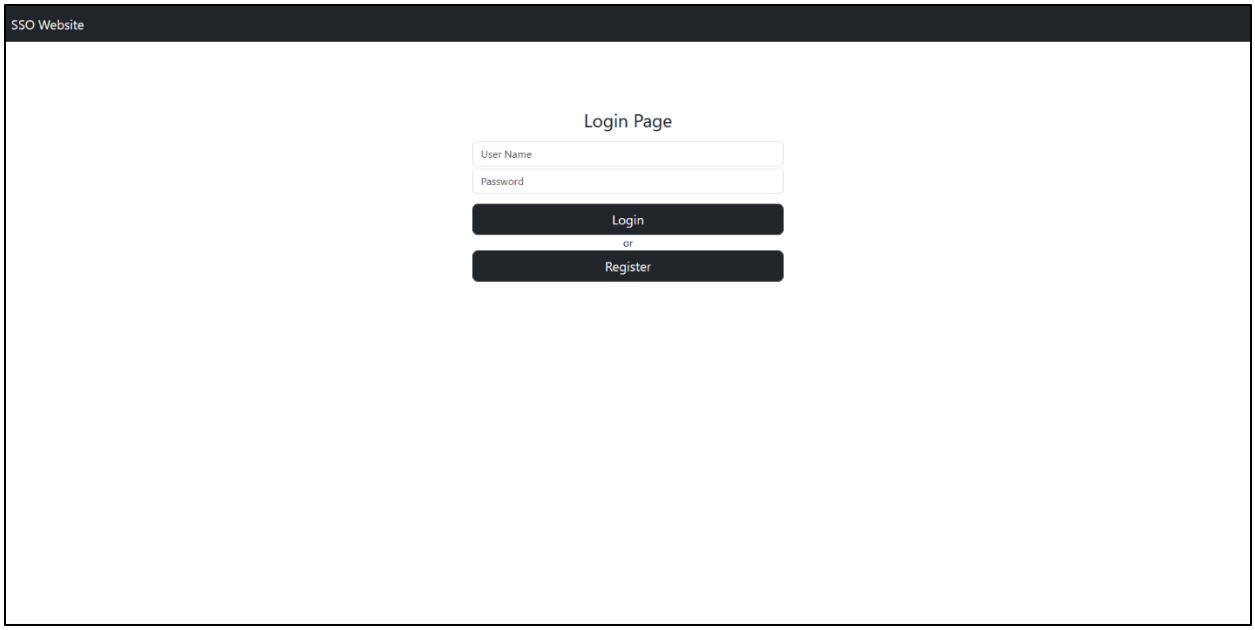

FIGURE 13: LOGIN PAGE

This is the login page, the login page requires the user to put in there login information. If it gets entered in wrong then an error message pops up. If entered in correctly they get redirected to the dashboard.

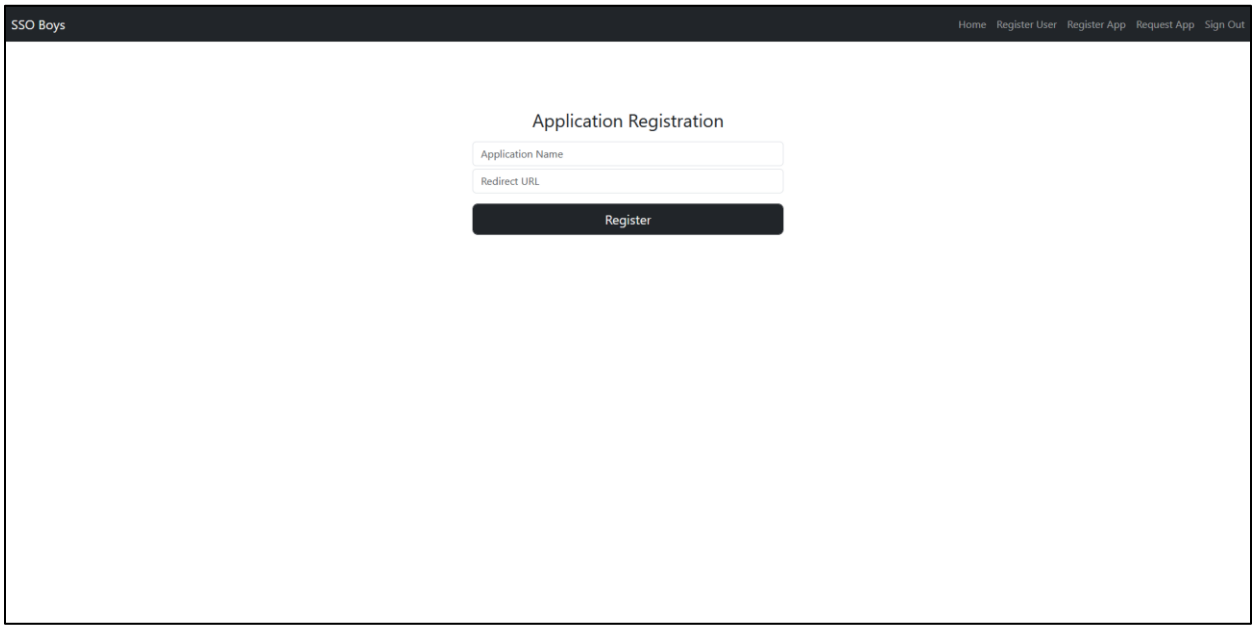

FIGURE 14: APP REGISTRATION

This is the application registration page. When a user registers an app, it will ask the user for the name f the app, and the redirect usl of the app. This is so the website knows where to redirect the user when its clicked on when on the dashboard.

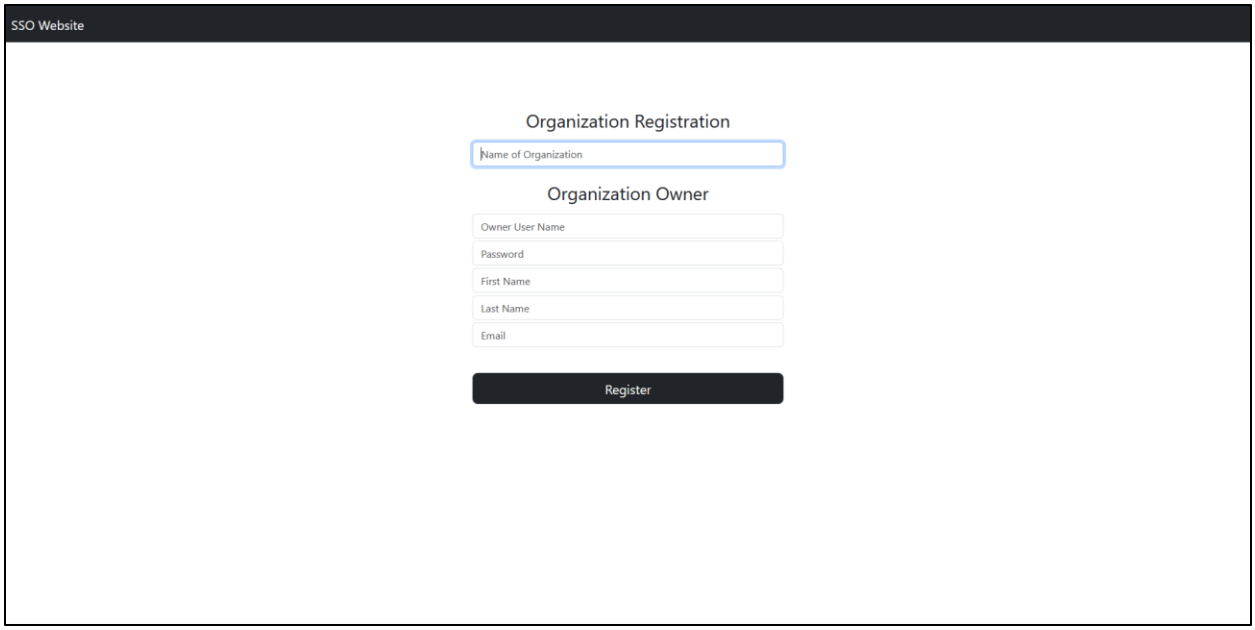

FIGURE 15: ORGANIZATION REGISTRATION

This is the organization register page, this is where the user can register an organization. It requires that the user create an owner account along with the name of the organization.

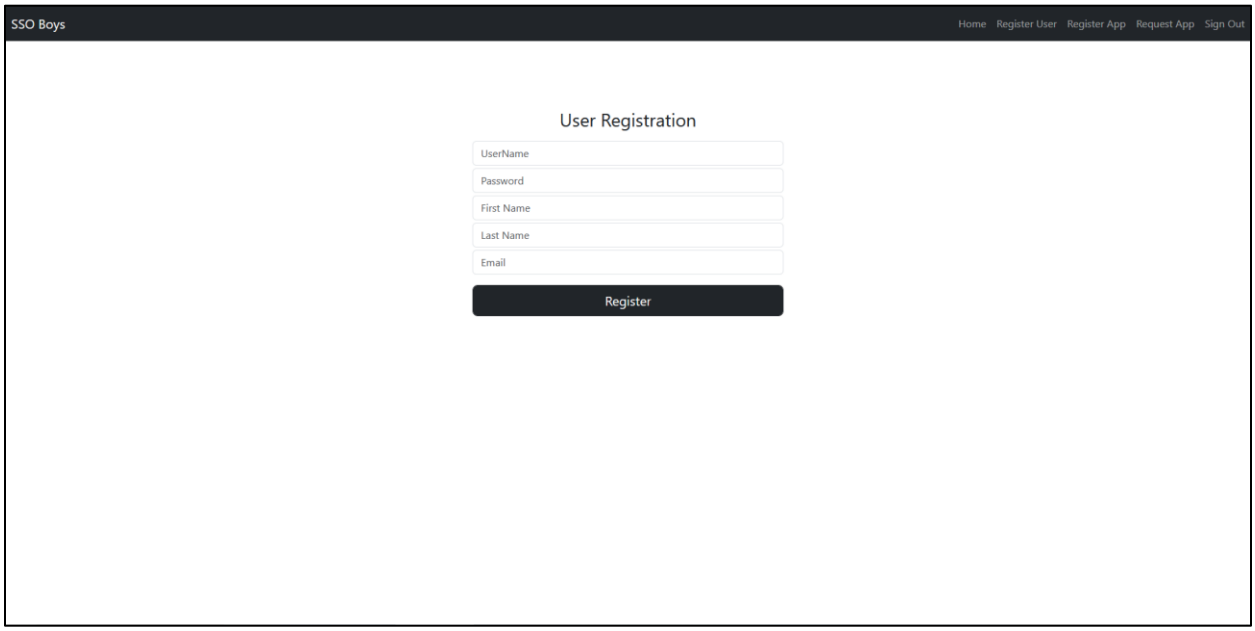

FIGURE 16: USER REGISTRATION

This is where a user gets registered. The user is not the one that gets to register for an account. This page can only be accessed by an admin or an owner. They are the ones that register the employee for the account, and they will give the credentials over to the employee after the user has been created.

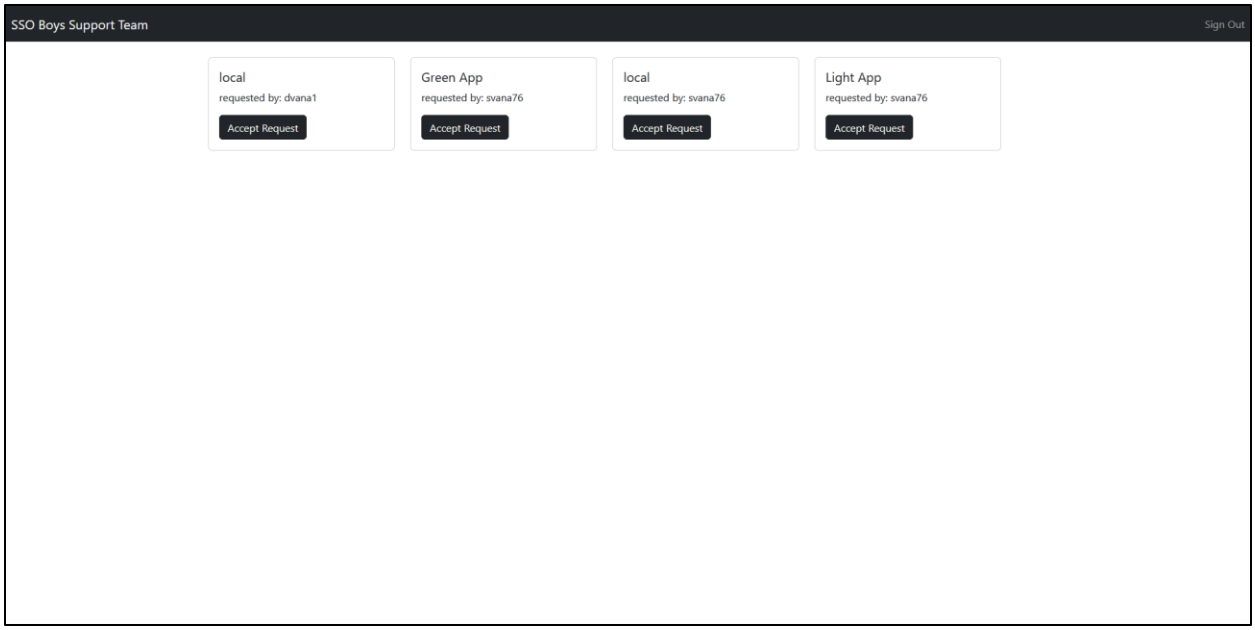

FIGURE 17: SUPPORT DASHBOARD

This is the support dashboard. This is where support accounts get sent when they log in. instead of seeing apps they can redirect to, they instead see apps that other users have requested access to. The support can grant access to users from this dashboard by accepting their requests.

360 SINGLE SIGN-ON (SSO) APPLICATION TESTING

1 Introduction

The testing of the SSO application was conducted to validate its adherence to the specified functional, non-functional, and interface requirements. The primary objective was to ensure the application's security, usability, and performance.

Test Scope and Execution:

2 Testing Types and Additional Information

2.1 Functional Testing:

- All user actions, including login, logout, app access, and app requests, were tested and functioned as expected.
- Different user roles, such as admin and base users, were successfully validated, ensuring appropriate access and capabilities for each role.

2.2 Non-Functional Testing:

- **Security:** The application was rigorously tested against SQL injection, XSS, and token theft. SSL implementation, password hashing, and salting mechanisms were found to be robust and effective.
- **Performance:** The application demonstrated excellent performance under stress, managing multiple simultaneous logins and maintaining responsive load times.

2.3 Interface Testing:

- The UI/UX was found to be intuitive, aligning with modern design principles and offering clear navigation.
- Compatibility tests across various browsers and devices showed consistent performance and no significant issues.

2.4 Test Environment and Data:

- A controlled test environment that closely replicated the production setting was used.
- Realistic test data, including edge cases, was employed, ensuring a thorough examination of the application's capabilities.

2.5 Result Analysis:

- Detailed documentation of all test cases was maintained, with inputs, expected outcomes, and actual outcomes meticulously recorded.
- Analysis of test results did not reveal any patterns of failure or recurring issues.

2.6 Bug Reporting and Retesting:

- Throughout the testing process, a few minor bugs were identified and logged using a bug tracking system.
- These issues were promptly addressed, and subsequent retesting confirmed their resolution without introducing new problems.

3 Final Validation:

- Comprehensive end-to-end testing confirmed that all parts of the application worked together seamlessly.
- User Acceptance Testing (UAT) was conducted with a select group of end-users, who verified the functionality and usability of the application, reporting high satisfaction.

3.1Reporting:

- A final report was compiled, summarizing the successful testing process, findings, and confirming the readiness of the application for deployment.

3.2 Post-Deployment Monitoring:

- Post-launch, monitoring tools have been implemented to continuously assess the application's performance and security.
- A feedback loop for collecting and addressing user feedback is in place, ensuring ongoing improvement and user satisfaction.

The testing of the SSO application was comprehensive and successful. All functional and non-functional requirements were met, and the application demonstrated robust security, high performance, and excellent user experience. With the successful completion of testing, the SSO application is confirmed to be ready for deployment and use in a live environment.

FINAL CONCLUSION

In this document we describe the project, talked about the requirements of the project. We mentioned the security risks this project faces due to the nature of this application. This includes XSS, SQL injection, data integrity, etc... This document showed a flurry of sequence diagrams that show exactly what happens on the backend in specific scenarios, along with a detailed diagram of the database using an entity relationship diagram. To explain the relationship amongst components we first displayed an MVC diagram and a deployment diagram. All UML diagrams took place within the design document. To conclude the report we went over the test plan that described how we combatted errors, and we went over the project plan to show how al team members worked together to build this application.

APPENDIX

PROJECT PLAN

1.0 Project Overview

The SSO app is a web application that will be used to authenticate and authorize users for other applications. The user will first log into the SSO application, then they will see a dashboard with a number of apps they are authorized to use. The user may click on any one of them, which will log them in to the home screen of the app chosen. Users and apps will be under organizations, and organizations will be able to add more applications to the SSO for their users to use. Organizations can be registered within the SSO app. The organization registers users into the organization.

2.0 Project website

https://protect-sso.github.io/SeniorProdWeb/

3.0 Final Deliverables - Specific To Your Project

1. A final report including SRS, SDD, Software test plan
2. The SSO website link
3. GitHub link to SSO website
4. GitHub link to API service
5. Presentation PowerPoint

4.0 Milestone Events (Prototypes, Draft Reports, Code Reviews, etc)

Define a few milestone events for your project – events that make sense for your project. In general, milestones like planning (RAD), Development, Prototype, Research Results, and Final Report.

#1 SRS and SDD drafts - By 9/21/2023

#2 SRS and SDD - By 9/24/2023

#3 Prototype I - By 9/30/2023

#4 STP – By 10/15/2023

#5 Prototype II - By 10/29/2023

#6 Final Report - By 11/15/2023

#7 Christmas – By 12/25/2023

5.0 Meeting Schedule Date/Time

Determined in the kickoff meeting.

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
			Meetings at 6pm		Meetings at 6pm	

6.0 Collaboration and Communication Plan

Communication will be done through discord, text messages, and in person at the Marietta campus library.

7.0 Project Schedule and Task Planning

Check the attached pdf.

8.0 Version Control Plan

GitHub will be used to commit changes to the project. The way we will collab in GitHub is, a developer makes a branch to work in. When they have made their changes, they will commit to their branch and push their branch up to GitHub. After that they will open a Pull Request and another team member will merge it into main.