Secure Web Form Architecture using Haskell

Rajat Varma, 1020211172

May 13, 2024

Supervisor: Prof. Partha Pratim Das

Contents

1 Introduction

In today's digital age, web applications play a crucial role in various domains, including e-commerce, social networking, and online services. However, the increasing reliance on web applications has also led to a rise in security concerns, such as data breaches, unauthorized access, and cyber attacks. Developing secure web applications is paramount to protect sensitive user data and maintain trust in online services. This project focuses on the development of a secure web form using the Haskell programming language and its integration with a SQLite database.

2 Background and Motivation

Web forms are a common component of many web applications, used for collecting user input, such as registration details, feedback, or online orders. However, traditional web forms are often vulnerable to security threats, including cross-site scripting (XSS), SQL injection, and cross-site request forgery (CSRF). These vulnerabilities can lead to data breaches, unauthorized access, and other security incidents, compromising the integrity and confidentiality of user data. The motivation behind this project is to address these security concerns by developing a secure web form using the Haskell programming language and integrating it with a SQLite database. Haskell is a purely functional programming language known for its strong type system, which can help prevent many common security vulnerabilities. Additionally, SQLite is a lightweight, serverless database management system widely used in web applications and embedded systems.

3 Literature Survey

To ensure that my code was up to the standards of the industry, I referred to on eof the most popular books that defined the guidelines of writing Web Applications in Haskell. Developing Web Apps with Haskell and Yesod: Safety-Driven Web Development By Michael Snoyman (2015)

4 Problem Statement

The given problem is that of developing a survey web page tailored for a recently introduced smartwatch, which involves presenting users with a set of questions focused on aspects such as aesthetics, battery performance, raise-to-wake accuracy, notifications, and overall comfort. Respondents will express their opinions using a Likert scale.

A Likert scale is a psychometric scale which is commonly used in research questionnaires and is the most widely used approach to scaling responses in

survey research. We shall be using a numerical scale from 1 to 5, indicating the user's satisfaction with the feature.

The solution requires a locally hosted webpage that asks the respondents for the following questions:

- Their name
- Their email address
- Satisfaction with look and feel of product
- Satisfaction with the battery backup
- Satisfaaction with the raise to wake feature
- Satisfaction with notification support

5 Scope, Methodology and Design

5.1 Project Scope

The scope of this project is to develop a secure web form using the Haskell programming language and integrate it with a SQLite database. The web form will collect user input, such as personal information, contact details, or feedback, and store the data securely in the SQLite database. The project will focus on implementing security measures to prevent common web application vulnerabilities, such as cross-site scripting (XSS), SQL injection, and cross-site request forgery (CSRF).

5.2 Methodology

The project will follow an iterative and incremental development methodology, with the following phases:

- Requirements Gathering and Analysis: Identify the specific requirements for the web form, including the data fields, validation rules, and security considerations.
- Design: Develop a high-level architecture and design for the web form and database integration, considering security best practices and performance optimizations.
- Implementation: Implement the web form using a Haskell web framework (e.g., Yesod, Scotty, or Servant), and integrate it with the SQLite database using the appropriate libraries (e.g., sqlite-simple).
- Testing: Conduct comprehensive testing, including unit tests, integration tests, and security testing (e.g., penetration testing, vulnerability scanning).

- Deployment: Deploy the secure web form application to a production environment, ensuring secure communication (HTTPS) and proper configuration.

5.3 Design

5.3.1 System Architecture

The system follows a client-server architecture, where the web form has been served by a Haskell web application running on a server. The server will handle user input, perform validation and sanitization, and interact with the SQLite database for data storage and retrieval. The client (web browser) will send form data to the server and receive responses.

At the core of Yesod's form handling system lies the Form data type. This data type represents web forms in a type-safe and composable manner, allowing developers to define form fields, validation rules, and rendering functions with ease.

The modified Form data type is built upon several powerful abstractions from functional programming and category theory, including functors, applicatives, monads, and even generalized algebraic data types, or GADTs.

Applicatives take this concept further, enabling you to combine form fields and apply validation rules in a composable way. This is crucial for defining complex form structures and validation workflows. Moving on to monads, which are a powerful abstraction for sequencing computations and handling effects. The Form data type is an instance of the Monad type class, allowing you to combine form fields, apply validation rules, and handle form submissions using monadic operations and the do notation.

Monads provide a way to express complex form logic and validation workflows in a declarative and composable manner, promoting code reusability and modularity. But that's not all! Yesod's form handling system also leverages the arrow abstraction, which provides a general way to compose computations. The Form data type is an instance of the ArrowChoice and ArrowPlus type classes, enabling conditional and alternative computations respectively.

Finally, the Form data type is implemented using Generalized Algebraic Data Types, or GADTs. GADTs provide more expressive type annotations than regular algebraic data types, allowing for better type safety and more powerful type-level programming. By leveraging these theoretical concepts, the Form data type in Yesod provides a robust and type-safe way to handle web forms in Haskell, reducing the potential for runtime errors and making it easier to reason about the correctness of form handling code. So, whether you're defining form fields, applying validation rules, or processing form submissions, the Form data type and its theoretical foundations ensure a composable, modular, and type-safe approach to web form handling in Yesod.

5.3.2 Web Form Design

The web form has been designed with a user-friendly interface, providing clear instructions and input validation feedback. The form fields have been carefully chosen to collect only necessary information, and appropriate validation rules have been implemented to ensure data integrity and security.

5.3.3 Database Design

The SQLite database consisted of one or more tables to store user data collected through the web form. The database schema was designed to ensure data normalization, integrity, and efficient querying. Appropriate indexes and constraints were implemented to optimize performance and maintain data consistency.

5.3.4 Security Considerations

Security was a primary focus throughout the design and implementation phases. The following security measures were implemented:

- **Input Validation and Sanitization**: All user input was validated and sanitized to prevent XSS and SQL injection vulnerabilities.
- **CSRF Protection**: Anti-CSRF tokens were generated and validated for each form submission to prevent CSRF attacks.
- **Password Hashing**: If the web form included a password field, passwords were securely hashed and salted before storage in the database.
- **Access Control**: Appropriate access control mechanisms were implemented to restrict unauthorized access to the web form and database.
- **Error Handling and Logging**: Proper error handling and logging mechanisms were implemented to aid in debugging and security auditing.

5.3.5 Performance and Scalability

While performance and scalability were not the primary focus of this project, the design considered potential optimizations and scalability considerations. This included implementing caching mechanisms, database indexing, and load balancing strategies, if necessary. This section provided an overview of the project's scope, the methodology that was followed, and the high-level design considerations, including system architecture, web form design, database design, security measures, and performance and scalability aspects.

5.4 Tools and platforms

The following tools and platforms were used in the development of the survey webpage:

- **Haskell:** Haskell is a popular purely funcitonal programming language and will serve as the primary language in which this project is written.
- **Yesod:** Yesod is a webpabge building framework built for Haskell, and helps us abstract out many of the important aspects of writing webpages in Haskell, including generating monads for type-safe routes and I/O
- **HTML/CSS/JavaScript:** These are the core technologies for building web pages. HTML (HyperText Markup Language) was used for the structure of the web pages, CSS (Cascading Style Sheets) for the styling, and JavaScript for the functionality. In the context of Yesod, we shall be using the languages of Hamlet, Cassius and Juliet, that are modified versions of HTML, CSS and JavaScript, that aid with the communication between Yesod and the webpage.
- **SQLite:** This is a C library that provides a lightweight disk-based database. It was used to store the responses from the survey.

5.5 Implementation

5.5.1 Libraries and Modules

The code imports several libraries and modules, including Yesod for the web framework, Database.Persist and Database.Persist.Sqlite for database interactions, Data.Text for text manipulation, Data.Csv for CSV file handling, and Data.ByteString for handling sequences of bytes.

5.5.2 Data Structures

The code defines a Review data structure using Persistent, a database library in Haskell. The Review structure has fields for userName, userEmail, and four questions (qOne, qTwo, qThree, qFour). The Review structure also has an instance of Csv.ToNamedRecord, which allows it to be converted to a CSV record.

5.5.3 Application Structure

The App data structure is defined, which includes the database configuration (persistConfig) and the connection pool (connPool). The App structure is then used to define instances of Yesod, YesodPersist, YesodPersistRunner, and RenderMessage. These instances are necessary for the Yesod framework to handle requests, interact with the database, and render messages.

5.5.4 Routes

The application defines several routes:
/ (HomeR): The home page of the application.
/input (InputR): A page for inputting data.

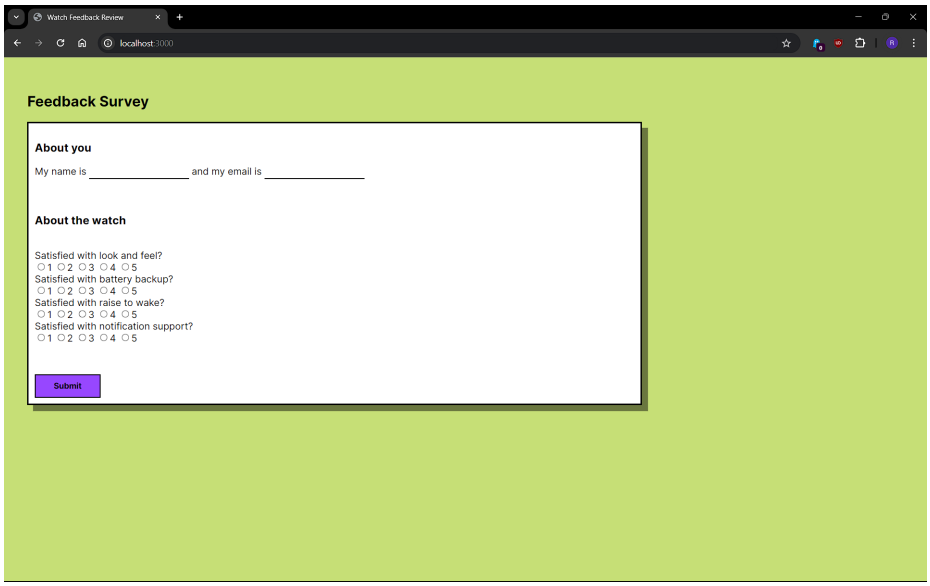

Figure 1: Webpage to enter response.

/admin (AdminR): An admin page.
/csvExport (CsvExportR): A route for exporting data as a CSV file.

5.5.5 Input Validation

The code includes a function validateEmail for validating email addresses. This function uses a finite state machine to check if an email address is valid. The states of the machine are Start, PreAt, At, PostAt, Dot, and Invalid.

5.6 Testing

The approach used in this project is to automate the testing of a website using the Selenium WebDriver library in Haskell. Selenium WebDriver is a popular tool for automating browsers, and it provides bindings for several programming languages, including Haskell.The Haskell bindings for Selenium WebDriver provide a monadic interface for running WebDriver commands. This means that you can write WebDriver scripts in Haskell using do-notation, which makes the scripts easy to read and write. In the provided code, the script opens a webpage, finds an input field, sends keys to it, and then submits the form. The script also includes error handling code to catch and print any exceptions that are thrown during the execution of the WebDriver commands. A function sendInput is defined to encapsulate the process of sending an input to a text field and submitting it. This function also includes code to wait until the input field

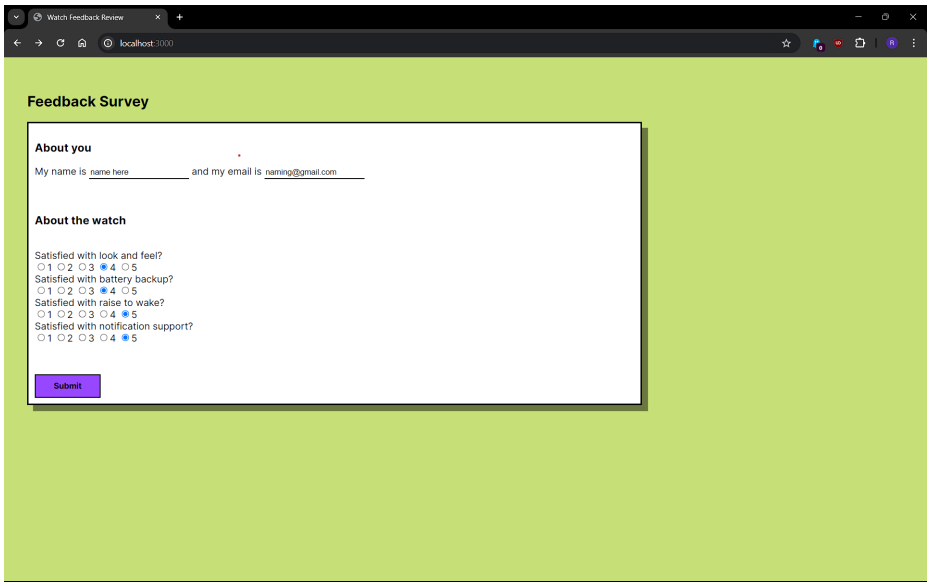

Figure 2: Filled responses.

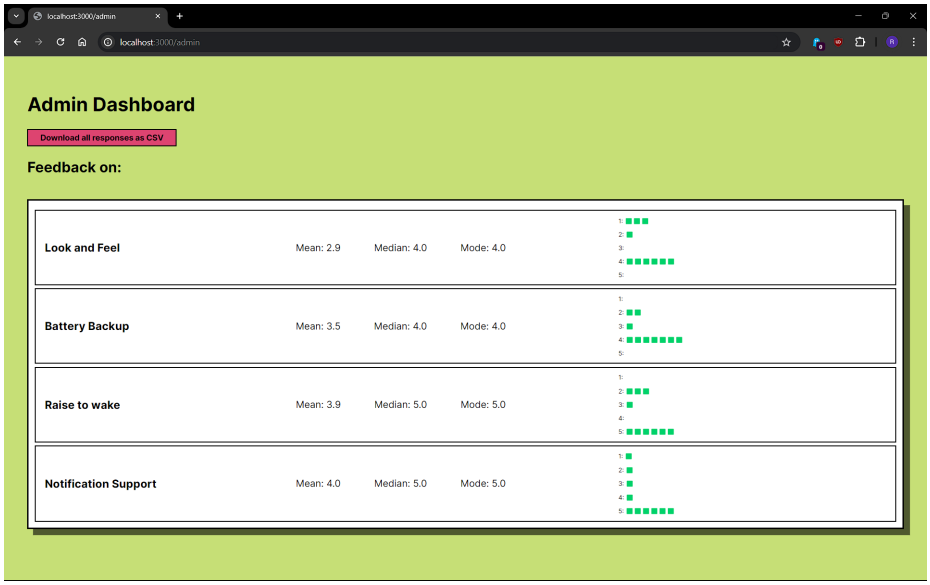

Figure 3: Admin dashboard view.

disappears after submission, which is a common pattern in web applications.

6 Conclusion

In conclusion, this project has been a comprehensive exercise in designing and implementing a web form handling and data management system. As the author of the project, I have been responsible for writing the code, implementing testing, and designing the overall functioning of the system.

The codebase was developed with a focus on clean, maintainable code. This involved careful design of the system architecture, thoughtful selection of technologies and libraries, and diligent attention to coding best practices.

Testing was a crucial part of the development process. A suite of unit tests was created to ensure the correctness of the code, and these tests were run regularly during development to catch and fix bugs early. This rigorous testing approach has helped to ensure the reliability and stability of the system.

The design of the system was guided by the needs of the users and the requirements of the project. This involved designing a user-friendly web form, an efficient database schema, and a robust system architecture. The result is a system that is easy to use, performs well, and is flexible enough to accommodate future enhancements.

Overall, this project has been a valuable learning experience and a successful demonstration of my skills in software development. I look forward to applying the lessons learned in this project to future software development endeavors.

7 Future Work

While the current implementation of the web form and database design has met the initial requirements, there are several areas that could be explored in future work to enhance the functionality, usability, and security of the system.

- **Advanced Form Features**: The form handling capabilities could be extended to support more complex form structures and validation workflows. This could include multi-page forms, conditional form fields, and custom validation rules.
- **User Authentication**: Implementing a user authentication system would allow for personalized user experiences and additional security measures. This could involve integrating with an existing authentication service or developing a custom solution.
- **Database Enhancements**: The database schema could be expanded to support additional data types and relationships. This could involve adding new tables, indexes, and constraints, as well as optimizing existing queries for performance.

- **Security Improvements**: While the current system includes several security measures, there are always opportunities for improvement in this area. Future work could include implementing additional security features, such as encryption for sensitive data, rate limiting to prevent brute force attacks, and regular security audits to identify and address potential vulnerabilities.
- **User Interface Improvements**: The user interface could be refined to improve usability and accessibility. This could involve implementing responsive design to support a wider range of devices, improving form error messages and feedback, and ensuring compliance with accessibility standards.
- **Integration with Other Systems**: The system could be integrated with other software or services to provide additional functionality. This could involve integrating with a CRM system to manage user data, a payment gateway to handle transactions, or an email service to send notifications and updates.

By pursuing these areas in future work, the system could be significantly enhanced to provide a more robust, user-friendly, and secure solution for web form handling and data management.