# **HealthPaladin, Project Asvins**

# **Software Design**

## **CSCI-P465/565 (Software Engineering I)**

## **Project Team**

### **Margaret** **Shubham** **Thomas Foley** **Sumanth Konjeti Venkata Viswanath Chittilla**

## 1. Introduction

This section introduces the design approach to the software system.

### 1.1 System Description

Provide a description of the system, and the problem that is being solved by developing the system (may be copied from the Project Plan or Software Specification if those documents are accurate).

|  |
| --- |
| Often for a patient to get proper healthcare, they will have to login through multiple platforms and call multiple doctors just to get one appointment set up. Even after all that, the doctor they chose may not take their insurance, which starts the process again. HealthPaladin which is a part of Project Asvins, is a web platform hub for all things healthcare. The goal of HealthPaladin is to organize resources for patients, doctors, and insurance agencies so that they can access everything they need right in one place. Patients will be able to find doctors that fit their needs and contact them right through their browser, they will also be able to manage their insurance and easily see to do tasks for upcoming appointments. Doctors will be able to access their patient's medical history and give feedback. Insurance agencies will be able to view their patient’s information and communicate deals and information to patients. |

### 1.2 Design Evolution

This section is intended to document the rationale behind the selected design solution.

#### 1.2.1 Design Issues

Given the real-world scenarios our platform addresses, simplicity is at its core. The user interface is meticulously designed to be intuitive, ensuring that users can effortlessly navigate the platform, even during the most critical healthcare interactions. With HealthPaladin, complexity is reduced to a minimum, making healthcare management a hassle-free experience for everyone involved.

#### 1.2.2 Candidate Design Solutions

Initially, our plan was to create the front-end web pages using Figma, and then translate the design into React code. We envisioned developing an application that would be accessible on both desktop and mobile devices, allowing users to log in and manage their appointments and insurance plans. However, with the ever-expanding range of screen sizes and devices, limiting the application to just desktop and mobile compatibility wouldn't provide the best user experience. Therefore, our goal is to ensure that it works seamlessly across various devices, making it device-agnostic for the sake of user convenience and usability.

#### 1.2.3 Design Solution Rationale

The chosen design solution involves creating a mobile-friendly web application with a straightforward user interface that adheres to a design language familiar to most users. This approach ensures that all users can easily access their insurance tracking and doctor appointments using their smartphones from any convenient location. Additionally, our web application will adapt to screens of various sizes, presenting all relevant information at a glance, making it a one-stop destination for managing insurance and appointments.

Our team possesses expertise in web application development using the MERN stack, so this approach aligns well with our skill set. Furthermore, it enables our application to be accessible on any device equipped with a web browser, eliminating any constraints related to operating systems.

### 1.3 Design Approach

#### 1.3.1 Methods

#### Our project, which is a Patient and Insurance management portal, is being developed through a systematic process of discussing project requirements and identifying the target demographics. These discussions play a pivotal role in shaping our design solution. Each conversation helps us eliminate ideas that, although appealing initially, had underlying issues. Gradually, we refine our design by eliminating suboptimal concepts, resulting in the best production ready design.

#### In terms of technical architecture, Object-Oriented Design serves as the foundation. We organized relevant entities such as Doctors, Patients, and Insurance Providers into their respective Objects, clarifying how data interactions would occur. Object-Oriented design ensured that all elements received equal attention, leaving no important aspect overlooked during the system's overall design.

#### 1.3.2 Standards

#### In crafting the user experience, we've taken great care in selecting color palettes and font families. These seemingly small details play a significant role in enhancing user-friendliness. Our carefully chosen colors promote readability, reduce eye strain, and create a visually pleasing environment. Likewise, the thoughtfully selected font families ensure that information is presented in a clear and easily digestible manner. These design choices contribute to a platform that not only simplifies healthcare management but also offers a visually appealing and comfortable user experience.

We followed some established principles of software design so as to not reinvent the wheel and to avoid common design pitfalls. Tunnel vision was one such pitfall we wary of, and managed to avoid by considering the overall picture of how our application works in the context of not just end users, but also fitness professionals and their ease of use in hosting/providing content, which in turns helps the end users have a rich, seamless experience. We also ensured that our design is flexible enough to accommodate any changes in the future, as it's always possible to have a greater understanding of requirements further down the line which may lead to design changes.

#### 1.3.3 Tools

In our project, we utilized JIRA to allocate responsibilities across various project modules. JIRA provides a versatile platform where tasks/modules are represented as cards, and these cards are categorized into three main sections based on their current status: To-Do, In Progress, and Done. We conducted discussions about these modules using Microsoft Teams calls and organized their respective cards into timeline sections accordingly. To create our Technical Diagrams, flow diagrams, and User Interfaces, we relied on Figma as our preferred tool, ensuring comprehensive and visually appealing project documentation.

## 2. System Architecture

### **2.1 System Design**

A screenshot of a computer

Description automatically generated

At a broad level, our system operates using the MERN stack, which is an acronym for MongoDB, Express, React, and Node. As depicted in the diagram, when looking at the system from a top-level perspective, it can be categorized into two main components: the Client and the Server. The web user interface will be generated using the React framework, a widely-used open-source JavaScript library renowned for constructing adaptable and component-based user interfaces. This front-end will receive routing and server-side functionality from our NodeJS and Express Backend. Access to data will rely on information stored within the MongoDB database and will be facilitated by the backend as needed.

### **2.2 External Interfaces**

Our system will interface with various external components. To ensure security, users will access our website through the encrypted HTTPS protocol. The web interface of our site will be generated by different browser rendering engines, such as Blink for Chromium-based browsers like Chrome, Edge, Brave, Vivaldi, Gecko for Firefox, and Webkit for Safari and other iOS browsers.

The React frontend will be responsible for transmitting HTML, CSS layouts, and JavaScript rendering logic. Any user-input data or data intended for display will pass through secure NodeJS APIs, which also handle routing for the React frontend. This routing mechanism ensures that the front end understands the necessary routes for specific user actions. In cases where users attempt actions without appropriate permissions, the routing logic will inform the frontend, enabling it to display relevant information on the user interface.

All application data will be stored in MongoDB, a NoSQL database that integrates seamlessly with the NodeJS and Express backend. Any data received from users and transmitted from the front end through the mentioned APIs will be stored in this database. This data will be exchanged in JSON format throughout all layers. MongoDB utilizes BSON format for data storage, which closely resembles the JSON format used by the backend, minimizing the risk of errors during format conversion.

The backend will retrieve relevant data from the database as needed, for tasks such as user authentication, and subsequently employ it to send APIs and routing instructions to the frontend.

## **3. Component Design**

### **3.1 Signup Component**

* **Component Name**Signup
* **Component Description**A user creates an account to sign-up for the HealthPaladin platform. A username, email and password is required to be entered. Client-side verification is done for checking correct formatting of email and password. This data is sent to backend where it’s verified whether an account.
* **Responsible Development Team Member**React Frontend: Margaret, Shubham  
  NodeJS Backend: Viswa, Sumanth, Margaret  
  MongoDB: Thomas, Shubham, Sumanth
* **Component Diagram**Graphically depict the design of the component in terms of interfaces with other components and external interfaces. Also, consider including a diagram depicting the internal operations and/or class relationships in the component.
* **Component User Interface**User interface will contain fields for Username, Email Address, Password and Password confirmation. User should also select their Profile (Doctor/ Patient/ Insurance Provider). User can also login via their existing Google/Facebook accounts
* **Component Objects**Signup Component Objects will be the same as the fields in the User Interface as this component, i.e., the username, email address, password, and the user profile.
* **Component Interfaces (internal and external)**Signup component will call the ‘Signup’ NodeJS API, and pass component objects data. API will check whether the database has an existing username with the same given on Signup and will create a new account for given credentials if not.
* **Component Error Handling**Error handling will be done to validate the existence of username in database, as if account exists with that username already, then a new account cannot be created with the same username.

### **3.2 Login Component**

* **Component Name**Login
* **Component Description**A user creates an account to sign-up for the HealthPaladin platform. A username, email and password are required to be entered. Client-side verification is done for checking correct formatting of email and password. This data is sent to backend where it’s verified whether an account.
* **Responsible Development Team Member**React Frontend: Sumanth, Shubham  
  NodeJS Backend: Viswa, Margaret  
  MongoDB: Viswa, Shubham, Sumanth
* **Component Diagram**Graphically depict the design of the component in terms of interfaces with other components and external interfaces. Also, consider including a diagram depicting the internal operations and/or class relationships in the component.
* **Component User Interface**User interface will contain fields for Email Address, Password. User can also login via their existing Google/Facebook accounts
* **Component Objects**Signup Component Objects will be the same as the fields in the User Interface as this component, i.e. email address and password
* **Component Interfaces (internal and external)**Login component will call the ‘Login’ NodeJS API, and pass component objects data. API will check whether the database has an existing username with the same given on Login, and will allow the user to pass through the login.
* **Component Error Handling**Error handling will be done to validate the existence of username in database, as if account exists with that username already.

**3.3 Search Component**

* **Component Name:** Search
* **Component Description** A user can search for doctors or insurance providers from the search bar on the home page. Users can search with doctor/insurance provider name by entering into the search box. This text is sent to backend to return all doctors/insurance providers that match with the text.
* **Responsible Development Team Member**
* React Frontend: Shubham, Sumanth  
  NodeJS Backend: Viswa, Margaret  
  MongoDB: Viswa, Shubham, Thomas
* **Component Diagram**Graphically depict the design of the component in terms of interfaces with other components and external interfaces. Also, consider including a diagram depicting the internal operations and/or class relationships in the component.
* **Component User Interface** User interface will contain fields for search text and search button.
* **Component Objects**Search Component Objects will be the same as the fields in the User Interface as this component.
* **Component Interfaces (internal and external)**Search component will call the ‘Search’ NodeJS API, and pass component objects data. API will check whether the database and returns the list of matching users.
* **Component Error Handling**Error handling will be done to validate the existence of searched username in database.

**3.4 Dashboard Component**

* **Component Name**Dashboard
* **Component Description**The Doctor has a separate dashboard view where they can see all his patients. The patient has a separate dashboard where they can check all the appointments with their respective dotors, and see their insurance plans. The Insurance Provider has a different dashboard where they can check on the patient insurance. A patient can chat with the insurance provider and the doctor.
* **Responsible Development Team Member**React Frontend: Margaret, Sumanth  
  NodeJS Backend: Viswa, Shubham  
  MongoDB: Viswa, Shubham, Thomas
* **Component Diagram**Graphically depict the design of the component in terms of interfaces with other components and external interfaces. Also, consider including a diagram depicting the internal operations and/or class relationships in the component.
* **Component User Interface and component objects**

**Doctor Dashboard:**

User Interface: The Doctor's dashboard provides access to information and tools relevant to their medical practice. It typically includes features like:

A list of all assigned patients.

Access to patient medical records and histories.

Appointment scheduling and management.

Communication tools to chat with patients and insurance providers.

User Screens:

Patient List: Displays the list of all patients under the doctor's care, along with basic patient information.

Patient Details: Provides detailed patient profiles.

Appointment Scheduler: Allows the doctor to schedule, reschedule, or cancel appointments with patients.

Chat Interface: Enables real-time communication with patients and insurance providers.

User Notifications/Messages: Doctors may receive notifications for upcoming appointments, messages from patients, or updates on insurance approvals.

**Patient Dashboard:**

User Interface: The Patient's dashboard offers a view of their medical information and appointments. It typically includes features like:

A list of upcoming and past appointments.

Access to their medical records.

Information about their insurance plans.

Communication tools to chat with their doctor and insurance provider.

User Screens:

Appointments: Displays a list of upcoming and past appointments, including date, time, and doctor details.

Medical Records: Provides access to their medical history, lab results, and treatment plans.

Insurance Details: Shows information about their insurance coverage, including policy details and claims.

Chat Interface: Allows patients to communicate with their doctor and insurance provider.

User Notifications/Messages: Patients may receive appointment reminders, messages from healthcare providers, and updates on insurance coverage.

**Insurance Provider Dashboard:**

User Interface: The Insurance Provider's dashboard focuses on managing patient insurance-related tasks. It typically includes features like:

A list of patients with active insurance policies.

Claims processing and approval.

Communication tools to chat with patients and doctors.

User Screens:

Patient List: Displays a list of patients with active insurance policies, including policy details.

Claims Processing: Provides tools to process and approve insurance claims submitted by patients or healthcare providers.

Chat Interface: Allows insurance providers to communicate with patients and doctors regarding insurance-related inquiries.

User Notifications/Messages: Insurance providers may receive notifications for new insurance claims, messages from patients or doctors, and updates on policy status.

The chat functionality is a common feature in all three user interfaces, enabling real-time communication between Doctors, Patients, and Insurance Providers. This chat feature can be used for inquiries, consultations, and sharing important information.

The user interfaces are designed to cater to the specific needs and roles of each user type, ensuring efficient management of patient data, appointments, insurance-related tasks, and communication within the healthcare system. Notifications and messages play a crucial role in keeping users informed about important events and updates.

**Component Interfaces (internal and external)**The React frontend will be responsible for transmitting HTML, CSS layouts, and JavaScript rendering logic. Any user-input data or data intended for display will pass through secure NodeJS APIs, which also handle routing for the React frontend. This routing mechanism ensures that the front end understands the necessary routes for specific user actions. In cases where users attempt actions without appropriate permissions, the routing logic will inform the frontend, enabling it to display relevant information on the user interface.

All application data will be stored in MongoDB, a NoSQL database that integrates seamlessly with the NodeJS and Express backend. Any data received from users and transmitted from the front end through the mentioned APIs will be stored in this database. This data will be exchanged in JSON format throughout all layers. MongoDB utilizes BSON format for data storage, which closely resembles the JSON format used by the backend, minimizing the risk of errors during format conversion.

* **Component Error Handling**Describe the steps taken in the design to incorporate fault tolerance, data corruption prevention, and incorrect operation avoidance. Please organize these into error cases.

**3.5 Chat Component**

* **Component Name** Chat
* **Component Description** Each user should have chat feature available to them to begin or continue conversations with other parties. Patient->Doctor, Patient->InsuranceProvider. The messages should be redirected to appropriate receivers.
* **Responsible Development Team Member**
* React Frontend: Margaret, Sumanth, Thomas  
  NodeJS Backend: Viswa, Shubham  
  MongoDB: Viswa, Shubham, Thomas
* **Component User Interface** The user interface of chat will be a window with past messages and textbox for users to type in message before sending it.
* **Component Objects** The Message object will have

Actual messageSenderReceiverSent timeReceived time

* **Component Error Handling** Proper Error handling is done.

**Example:** Message Not sent

## **Revision History**

|  |  |  |
| --- | --- | --- |
| **Revision** | **Date** | **Change Description** |
| Sprint 1 | 10-02-2023 | Set up the basic front end and backend. Set up Git repo, JIRA and designed all the documents. |
| Sprint 2 | 10-16-2023 | Setup of Login/Signup pages and backend for this. Setup all the backend models for Patient, Doctor and Insurance Provider. Connect front-end with backend. |
| Sprint 3 | 10-18-2023 | Dashboards for Patient, Doctor and the InsuranceProvider. Core task – 2 setup. |
| Sprint 4 | 11-13-2023 | Dashboards for Patient, Doctor and the InsuranceProvider. Core task – 3 setup. |
| Sprint 5 | 12-03-2023 | Appointment scheduling, recommendations, covid news details, Chat feature. Core task 4 and 5. |