**Software Design**

**Document**

**for**

**Peggasus Booking**

**Version 1.1.1 approved**

**Prepared by Santiago Del Rio Obando, Antonio Ortega Guererro, Paulo Da Costa, Jason Salazar, Pierce Nance**

**Cal State La, Professor Marianne Abalos**

**12/01/23**

Table of Contents................................................................................................................... pg #1

Revision History..................................................................................................................... pg #3

1. Introduction................................................................................................................ pg #4
   1. Purpose........................................................................................................... pg #4
   2. Document Conventions……………………................................................. pg #4
   3. Intended Audience and Reading Suggestions................................................ pg #4
   4. System Overview........................................................................................... pg #4
2. Design Considerations............................................................................................... pg #7
   1. Assumptions and dependencies..................................................................... pg #7
   2. General Constraints........................................................................................pg #8
   3. Goals and Guidelines.....................................................................................pg #9
   4. Development Methods................................................................................... pg #10
3. Architectural Strategies.............................................................................................. pg 12
4. System Architecture................................................................................................... pg 15
   1. ....................................................................................................................... pg 15
   2. ....................................................................................................................... pg 16
5. Policies and Tactics.................................................................................................... pg #
   1. Specific Products Used.................................................................................. pg 21
   2. Requirements traceability............................................................................... pg 21
   3. Testing the software....................................................................................... pg 21
   4. Engineering trade-offs................................................................................... pg 21
   5. Guidelines and conventions........................................................................... pg 21
   6. Protocols........................................................................................................ pg 21
   7. Maintaining the software............................................................................... pg 21
   8. Interfaces........................................................................................................ pg 21
   9. System's deliverables..................................................................................... pg 21
   10. Abstraction..................................................................................................... pg 21
6. Detailed System Design............................................................................................. pg #22

6.x Name of Module........................................................................................... pg #23

6.x.1 Responsibilities...................................................................................... pg #23

6.x.2 Constraints............................................................................................. pg #23

6.x.3 Composition........................................................................................... pg #23

6.x.4 Uses/Interactions.................................................................................... pg #23

6.x.5 Resources............................................................................................... pg #23

6.x.6 Interface/Exports.................................................................................... pg #23

1. Detailed Lower level Component Design

7.x Name of Class or File..................................................................................... pg #25

7.x.1 Classification.......................................................................................... pg #25

7.x.2 Processing Narrative(PSPEC)................................................................ pg #25

7.x.3 Interface Description.............................................................................. pg #25

7.x.4 Processing Detail.................................................................................... pg #25

7.x.4.1 Design Class Hierarchy................................................................... pg #25

7.x.4.2 Restrictions/Limitations.................................................................. pg #25

7.x.4.3 Performance Issues........................................................................ pg #25

7.x.4.4 Design Constraints......................................................................... pg #25

7.x.4.5 Processing Detail For Each Operation........................................... pg #25

1. User Interface
   1. Overview of User Interface.......................................................................... pg #26
   2. Screen Frameworks or Images..................................................................... pg #27
   3. User Interface Flow Model........................................................................... pg #28
2. Database Design
3. Requirements Validation and Verification................................................................. pg #31
4. Glossary..................................................................................................................... pg #33
5. References................................................................................................................. pg #33

**Revision History**

| Name | Date | Reason For Changes | Version |
| --- | --- | --- | --- |
| Santiago Del Rio Obando, Antonio Ortega Guerrero,  Paulo Da Costa,  Jason Salazar, Pierce Nance | 10/23/23 | Created Documents | 0.0.0 |
| Paulo Da Costa | 11/10/23 | Updated Table of Contents  Overall proofread | 0.0.1 |
| Paulo Da Costa Antonio Ortega  Santiago Del Rio  Jason Salazar | 11/30/23 | Final Review | 0.1.1 |
|  |  |  |  |

**1. Introduction**

**1.1 Purpose**

This document outlines the design and architecture of the Hospital Appointment Booking Website, Peggasus Booking, version 1.0.0. The scope of this SDD encompasses the entire Peggasus Booking system. This includes all major components, modules, and subsystems responsible for user authentication, appointment scheduling, user profiles, doctor profiles, notification systems, and database management.

**1.2 Document Conventions**

No Specific Document Conventions are used at this time.

**1.3 Intended Audience and Reading Suggestions**

* Intended Audience:
  + Developers, Project Managers, Testers, Documentation Writers.
* Reading Suggestions:
  + Developers: Begin with Section 2 (Design Considerations) to understand the assumptions, constraints, and goals that influence the system's design. Proceed to Section 3 (Architectural Strategies) for insights into the system's overall structure. Then, delve into Section 6 (Detailed System Design) for component-specific details.
  + Project Managers: Start with Section 4 (System Architecture) to gain a high-level overview of how the system is organized. Then, review Section 5 (Policies and Tactics) for information on design policies and tactics. Section 9 (User Interface) will provide an understanding of the end-user experience.
  + Testers: Focus on Section 9 (User Interface) for an overview of user interactions and flows. Move on to Section 6 (Detailed System Design) to understand specific components and their responsibilities. Finally, consult Section 10 (Requirements Validation and Verification) for testing methods and validation criteria.
  + Documentation Writers: Begin with Section 4 (System Architecture) to grasp the overall structure of the system. Then, refer to Section 8 (User Interface) for insights into the user interface elements. Section 9 (Requirements Validation and Verification) will provide information on system requirements.
  + End Users: While the document primarily targets technical stakeholders, end users may find value in Section 4 (System Architecture) for a broad understanding of the system's capabilities. They can also refer to Section 8 (User Interface) for insights into user interactions.

**1.4 System Overview**

Peggasus Booking is a comprehensive software system designed to facilitate seamless scheduling and management of appointments between patients and healthcare providers. This system serves as a centralized platform, streamlining the process of booking medical consultations, examinations, and treatments.

**1.4.1 Functionality**

* The primary functionalities of Pegassus Booking include:
* User Authentication and Authorization:
  + The system allows users (both patients and healthcare providers) to create accounts, providing secure authentication mechanisms to ensure data privacy and user identity validation.
* Appointment Scheduling:
  + Users can view the availability of healthcare providers, select preferred time slots, and schedule appointments for specific medical services.
* User Profiles and Preferences:
  + Registered users have access to personalized profiles where they can manage their contact information, view appointment history, and set communication preferences.
* Doctor Profile and Availability Management:
  + Healthcare providers have the ability to create and manage their profiles, specifying their specialties, availability, and clinic details.
* Notifications and Reminders:
  + The system sends automated notifications and reminders to users regarding upcoming appointments, ensuring timely attendance.

**1.4.2 Design Approach**

* + The design approach of Peggasus Booking centers around a Model-View-Controller (MVC) architectural pattern. This pattern provides a clear separation of concerns, enhancing maintainability and scalability.
    - Model: Represents the data and business logic of the system. It encompasses the database schema, data handling, and validation processes.
    - View: Defines the user interface elements and their layout. It focuses on presenting information in an intuitive and user-friendly manner.
    - Controller: Acts as an intermediary between the Model and View. It handles user interactions, processes input data, and updates the Model accordingly.
  + The system employs a RESTful API for seamless communication between the frontend and backend components. This allows for efficient data exchange and supports a diverse range of client platforms.

**1.4.3 Organization**

* The system is organized into distinct modules, each responsible for specific aspects of functionality. These modules include:
  + User Authentication Module: Manages user registration, login, and session management.
  + Appointment Management Module: Handles appointment scheduling, availability management, and notification services.
  + User Profile Module: Enables users to create, view, and update their profiles, including contact information and communication preferences.
  + Doctor Profile Module: Allows healthcare providers to create and manage their profiles, specifying their specialties, availability, and clinic details.
  + Administrative Module: Provides tools and functionalities for system-wide management and oversight. This module includes features for user management as well as system configuration.
* By organizing the system into these modules, we ensure a modular and maintainable architecture, making it easier to extend and enhance functionality in the future.

**2. Design Considerations**

**2.1 Assumptions and Dependencies**

These assumptions or dependencies to the software and its use. These may concern such issues as:

* **Related software or hardware**:
  + **Assumption**: The system assumes that users have access to a standard web browser.
  + **Dependency**: The system may depend on external services for features like email notifications, or payment processing
* **Operating systems**:
  + **Assumption**: The system is assumed to be platform-agnostic and should work on major operating systems, including Windows, macOS, and Linux.
  + **Dependency**: The system may depend on specific server-side operating systems for hosting, such as Linux or Windows Server.
* **End-user characteristics**:
  + **Assumption**: Users are assumed to have basic computer literacy and internet access.
  + **Assumption**: Users are expected to provide accurate and up-to-date personal information for registration and appointment booking.
  + **Assumption**: Doctors and administrative staff are assumed to be trained to use the system efficiently.
  + **Dependency**: The system may require users to have a compatible device with certain screen size or resolution for optimal user experience.
* **Possible and/or probable changes in functionality**:
  + **Assumption**: The system assumes that it will evolve over time and require updates, enhancements, and bug fixes.
  + **Assumption**: Regulatory requirements, such as healthcare compliance, are expected to change over time, and the system should be updated to meet these changes.
  + **Dependency**: The system may depend on third-party APIs for services like geolocation, payment processing, or electronic medical records, and changes in API could impact those functionalities.

**2.2 General Constraints**

This section outlines the various global limitations and constraints that can significantly impact the system’s design and development of the Peggasus Online Appointment Booking System. These include the following:

* **Hardware or software environment**:
  + **Constraint**: The system must run on low-end hardware, as some users may access it from older devices.
  + **Impact**: The software design must optimize resource utilization and minimize memory and processing requirements to ensure smooth performance on a wide range of devices.
* **Security requirements**:
  + **Constraint**: The system must comply with the Health Insurance Portability and Accountability Act (HIPAA) regulations to protect patient health information.
  + **Impact**: The software design must include robust security measures such as encryption, authentication, and access controls to ensure the confidentiality and integrity of patient data..
* **Performance requirements**:
  + **Constraint**: The system must support a large number of concurrent users during peak appointment booking hours.
  + **Impact**: The software design needs to incorporate load balancing, caching, and efficient database queries to meet performance expectations and avoid service disruptions.
* **Standards compliance**:
  + **Constraint**: The system needs to adhere to industry standards for web accessibility to ensure usability for individuals with disabilities.
  + **Impact**: The software design needs to incorporate accessible user interfaces and provide alternative content for screen readers and assistive technologies.
* **Network communications**:
  + **Constraint**: The system needs to be accessible even with limited bandwidth or intermittent network connectivity.
  + **Impact**: The software design needs to employ techniques like progressive web apps and offline data storage to ensure essential functionality remains available when network conditions are suboptimal.
* **Data repository and distribution requirements**:
  + **Constraint**: The system must ensure data consistency and availability, even in the event of data center failures.
  + **Impact**: The software design needs to include data replication and redundancy strategies to maintain data integrity and availability.
* **Verification and validation requirement**s:
  + **Constraint**: Rigorous testing, functional, security, and compliance testing, is required before deployment.
  + **Impact**: The software design needs to incorporate clear interfaces, and logging to facilitate testing and debugging.

**2.3 Goals and Guidelines**

**Goal 1: Efficient Appointment Management**

**Description**: The system aims to provide a user-friendly platform for patients, doctors, and administrative staff to efficiently schedule, manage, and monitor appointments.

**Reason:** This goal is essential to streamline the appointment booking process, making it convenient for all users. It improves overall productivity and reduces manual intervention.

**Goal 2: Improved User Experience**

**Description**: The system focuses on enhancing the experience of patients by allowing them to easily book, reschedule, and cancel appointments online.

**Reason:** A user-friendly interface increases user satisfaction, encourages patient engagement, and reduces potential frustrations related to appointment management.

**Goal 3: Streamlined Doctor Scheduling**

**Description**: The system enables doctors to have better control over their schedules, allowing them to manage appointments and view patient information conveniently.

**Reason:** This goal is crucial for optimizing the workflow of healthcare providers, ensuring that they can efficiently allocate their time and resources.

**Goal 4: Enhanced Administrative Oversight**

**Description**: The system empowers administrative staff with the tools to efficiently manage user accounts, appointments, and system settings.

**Reason:** This is important for maintaining the smooth operation of the system, ensuring that administrative tasks are handled effectively and without complications.

**Goal 5: Integration with Existing Systems**

**Description**: The system aims to seamlessly integrate with the hospital's existing Electronic Medical Record (EMR) system for a cohesive patient management experience.

**Reason:** This integration ensures that relevant patient data is accessible to authorized personnel, providing a comprehensive view of a patient's medical history.

**Goal 6: Security and Compliance with HIPAA**

**Description:** The system places a strong emphasis on security measures to protect sensitive patient data and compliance with healthcare data privacy regulations (HIPAA).

**Reason:** Ensuring data security and compliance is crucial for maintaining patient trust, avoiding legal issues, and safeguarding sensitive information.

**Goal 7: User Authentication and Authorization**

**Description:** The system implements a secure authentication mechanism and role-based access control to ensure that users only have access to appropriate features.

**Reason:** This goal ensures that data is accessed and modified only by authorized individuals, reducing the risk of unauthorized access.

**Goal 8: Cross-Browser Compatibility and Responsiveness**

**Description:** The system is designed to function consistently across various web browsers and devices, including desktops, tablets, and mobile phones.

**Reason:** This ensures that users have a consistent experience regardless of their choice of device or browser, improving accessibility.

**Goal 9: Adherence to Deadline**

**Description:** The software development process is guided by a mandatory delivery date, which must be met by the end of the CS 3337 class.

**Reason:** Meeting the deadline is crucial for project completion and ensures that the system is ready for deployment in a timely manner.

**2.4 Development Methods**

The software design for Peggasus Online Appointment Booking was primarily guided by Agile Development with a specific implementation of the Scrum framework.

* Agile Development emphasizes iterative and incremental progress, allowing for flexibility in responding to changing requirements and user feedback. It promotes collaboration among cross-functional teams and continuous delivery of small, manageable features.
* Scrum is a specific Agile framework that divides the development process into small, time-boxed iterations called "sprints". Each sprint typically lasts two to four weeks and results in a potentially shippable increment of the product. The Scrum approach includes regular sprint planning, daily stand-up meetings, sprint review, and sprint retrospective.

This combination of Agile and Scrum was chosen to facilitate:

* Iterative Progress: Breaking down development into small, manageable chunks allows for rapid iteration and continuous improvement based on feedback.
* Flexibility and Adaptability: Agile methodologies are well-suited for projects with evolving requirements, enabling the team to adjust priorities and features as needed.
* Team Collaboration: Cross-functional teams collaborate closely throughout the development process, ensuring a shared understanding of goals and tasks.
* Client Engagement: Regular sprint reviews provide opportunities for client feedback, ensuring that the product aligns with their expectations.
* Transparency and Accountability: Daily stand-ups and regular retrospectives promote transparency within the team and hold members accountable for their tasks.
* Risk Mitigation: Agile practices help identify and address potential issues early in the development process, reducing the likelihood of major setbacks.

While other development methodologies may have been considered, the Agile and Scrum approach was chosen for its ability to adapt to the dynamic nature of the project, facilitate effective team collaboration, and prioritize client involvement in the development process. For a more detailed description of Agile and Scrum, references to foundational Agile literature and Scrum guides were consulted.

**3. Architectural Strategies**

**3.1 Architectural Pattern**

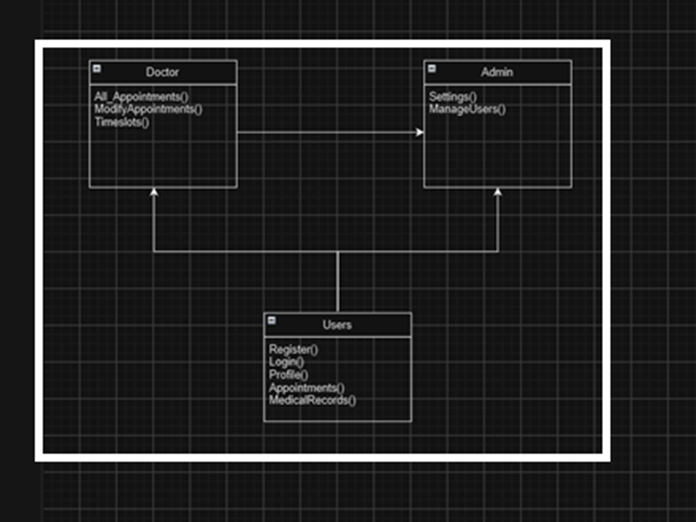
* The chosen architectural pattern for Peggasus Booking is the Model-View-Controller (MVC) pattern.
  + **Model(M):** This component manages data and business logic of the application. It represents the structure of the data, its relationships, and the rules governing how the data can be manipulated.
  + **View(V):** The view is responsible for presenting the data to the user and receiving user input. It encompasses the user interface components, which include pages, forms, and interfaces.
  + **Controller(C):** The controller acts as an intermediary between the model and the view. It receives input from the user, processes it with interactions with the Model and updates the view accordingly.
* **Reasoning:**
  + This pattern offers benefits like clear separation of components, flexibility, support for parallel development, adaptability to changes, and ease of testing. It aligns well with the system's requirements and design goals.

**3.2 Modularization and Component-Based Development**

* **Decision:** The system was organized into distinct components (e.g., User Management, Appointment Booking, Administrative Functions) to encapsulate specific functionalities.
* **Reasoning:** Modularity promotes maintainability, reusability, and allows for parallel development of different system features. It also enhances the system's ability to evolve over time.

**3.3 Products Implemented**

* **Programming Language:** The backend of the system is developed using C#, a versatile and powerful language known for its efficiency and robustness.
* **Web Framework:** ASP.NET Core, a high-performance framework, is used for building modern, scalable, and high-performance web applications.
* **Database:** MySQL is chosen as the database management system to handle data storage, retrieval, and management efficiently.
* **Frontend Technologies:** HTML, CSS, and JavaScript are used for creating the user interface, ensuring a seamless and interactive experience.
* **Framework for User Interface:** Bootstrap is employed to enhance the aesthetics and responsiveness of the frontend.
* **Version Control:** Github is utilized for version control, allowing collaborative development and easy management of code changes.
* **SQL Database**: Pre-made SQL databases from our database classes are used as a guide to create our hospitals database. Following steps from an already made database will help us to refine our own while streamlining the process.
* **Javascript:**  Due to some roadbumps encountered in the website development process, we may switch our coding language to Javascript to ease the website coding process We are also going to change to C# for this reason.
* **MySQL Workbench:** Our current SQL application seems to be a light version that does not integrate well with websites and as such will be required to move to a version that allows for this process but this will have little to no effect on current databases.
* **Mouse(GUI):** Since our project revolves around the use of a website, our primary form of interaction will be a Mouse controlled graphical user interface (GUI)
* **Mobile User Interface (MobileUI)**: While optimizing a mobile version of the site would be a stretch goal, even unoptimized, smart devices should still function with the same interface
* **Hardware**:Our website is created for those who use a desktop/laptop or a smart device. Other devices who have access to the internet will be able to view the website as well, The website will be created and optimized with only those two in mind
* **Software**: As our website will not have its own stand-alone application, we will be relying on internet browsers in order to access our website. We plan to make sure our website’s performance is up to grade on a few popular browsers such as Chrome, Firefox, and Edge.
* **Testing:**Through manual and automatic testing we hope to eliminate any errors that may show up during regular use and design catches to prevent errors from crashing the website and causing data loss.
* **Regular backups:** In the case that an error slips through the cracks, we will implement regular backups to keep both downtimes and data loss to a minimum.
* **Isolation:** We plan to implement database practices such as Isolation which will not allow any changes to the database unless the current query transaction is successful. A failed transaction will leave all data unchanged.
* **Memory management software** Applications like [SSMS](https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16) will be used to control and optimize memory on SQL databases
* **Optimization:** SQL database queries will be optimized to only use what is required to perform the query
* **Settings 0**: SQL has settings for memory allocation that can be kept at 0 which allows for SQL to use what memory is necessary
* **Retention:** In accordance with our SRD, data will be kept indefinitely until the patient or administration deletes account information from the database.
* **Network:** Our data will be kept on an SQL server which can be controlled or edited from the website given the user has the proper permissions to do so.
* Bellow is a generalization chart which shows how each type of user will relate to each other arrow pointing to a box mean they can use the functions they come from

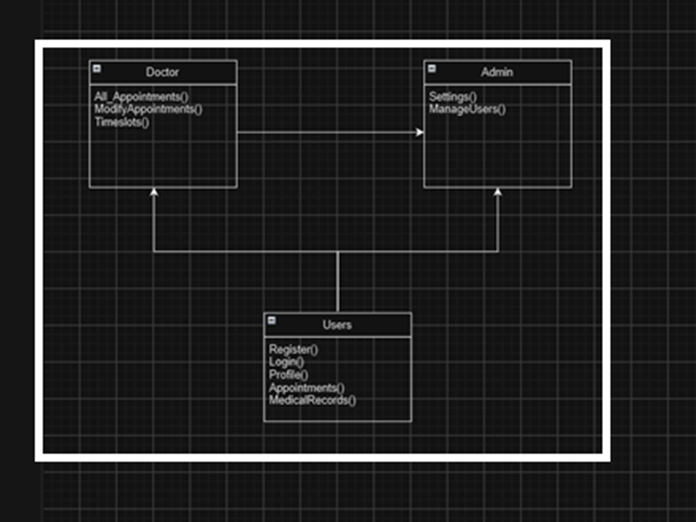


* **Isolation:** The website’s records will be kept updated based on the database and with Isolation,any changes made to the database will only be synced once they go with no errors
* Our project will communicate in three different ways
  + **Database:** This is where our data will be stored and where the majority of interactions will be.
  + **Email/phone:** We plan to have communication with a phone and email in order to notify users of the upcoming appointments.
  + **Maps**:Our project will integrate google maps in to show users where their next appointment is located at.

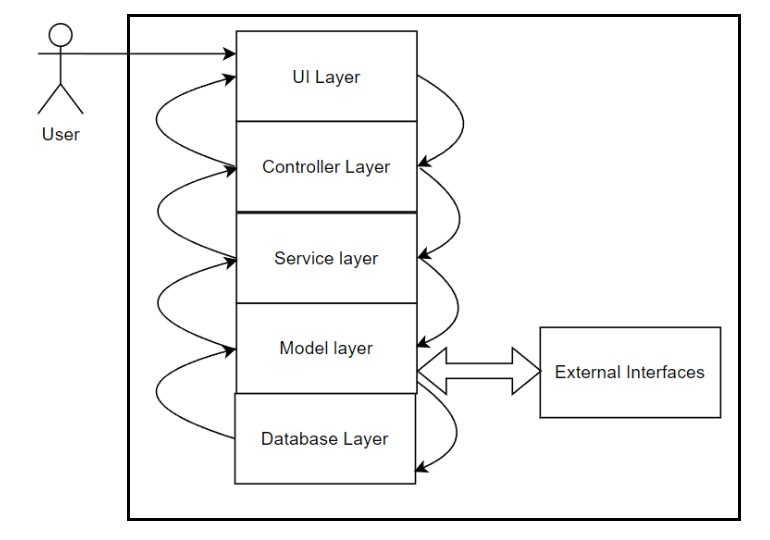
**4. System Architecture**

## 4.1 Logical View

* **User class**: The lowest class in our website will be the basic users. They will have access to features such as registration, login, profile, appointment, and medical records.
* **Doctor class**: Doctor classes will be able to do everything regular users can use but will also be able to access all appointments, modify any appointments and be able to set their time slots.
* **Admin Class**: The Admin Class will have access to everything the previous two classes can access but also have the ability to modify settings on the site and be able to manage users.



## 4.2 Development View



**System Decomposition**

**UI Layer:**

* Description: This layer encompasses the user-facing elements of the system, including web pages, forms, and interactive components.
* Responsibilities: Handles user interactions, presents information in a user-friendly manner.
* Components: HTML templates, CSS stylesheets, client-side JavaScript.

**Controller Layer:**

* Description: Acts as an intermediary between the UI layer and the underlying business logic.
* Responsibilities: Receives input from users, processes requests, and coordinates information flow.
* Components: Controllers in C# (ASP.NET Core).

**Service Layer:**

* Description: Contains the core business logic and application services.
* Responsibilities: Encapsulates system functionalities (e.g., appointment scheduling, user authentication).
* Components: Service classes in C#.

**Model Layer:**

* Description: Represents data structures and entities used within the system.
* Responsibilities: Defines how data is organized, stored, and manipulated (e.g., patients, doctors, appointments).
* Components: Entity classes in C# (using Entity Framework Core).

**Database Layer:**

* Description: Encompasses the database management system (DBMS) for data storage and retrieval.
* Responsibilities: Ensures data integrity, security, and efficient querying.
* Components: MySQL Database, tables for patients, doctors, appointments, etc.

**External Interfaces:**

* Description: Allows the system to interact with external services and resources.
* Responsibilities: Facilitates communication with Google Calendar API, email notifications, EMR systems integration, etc.

**Key Dependencies**

Key dependencies refer to the relationships and reliance between different components or subsystems of the system. In our case:

* The Controller layer depends on the UI layer to receive user input and initiate actions.
* The Service layer depends on the Controller layer to invoke services based on user requests.
* The Model layer relies on the Service layer to perform business logic operations.
* The Database layer is essential for storing and retrieving persistent data, and it's a critical dependency for the Model layer.
* External Interfaces are crucial for functionalities like calendar synchronization and email notifications.

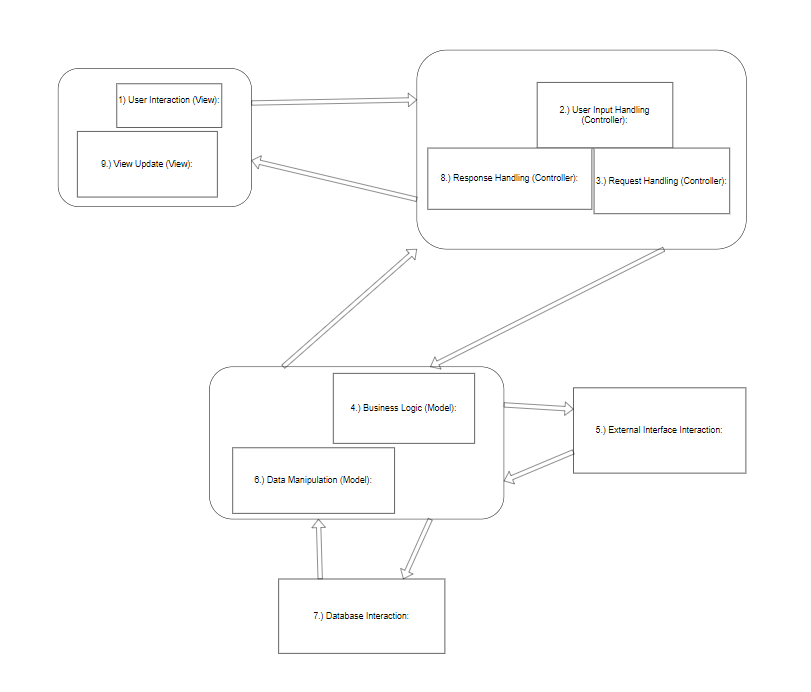
**Rationale**

The chosen decomposition aligns with several design goals and principles:

* Modularity and Separation of Concerns: Each layer has a distinct responsibility, promoting maintainability and ease of development.
* Scalability: By structuring the system in layers, it becomes easier to scale specific components independently, such as adding more servers to the Database layer.
* Flexibility for Future Changes: This architecture allows for easier modifications or updates in the future. For instance, switching to a different database system would primarily impact the Database layer.
* Security and Data Integrity: The separation of layers helps enforce security measures and maintain data integrity. For example, the Database layer can implement access controls.
* Reusability of Components: The use of established frameworks like ASP.NET Core and Entity Framework Core promotes reusability and accelerates development.

This decomposition strategy has been chosen to strike a balance between simplicity, scalability, and maintainability, aligning with the overall goals of the project.

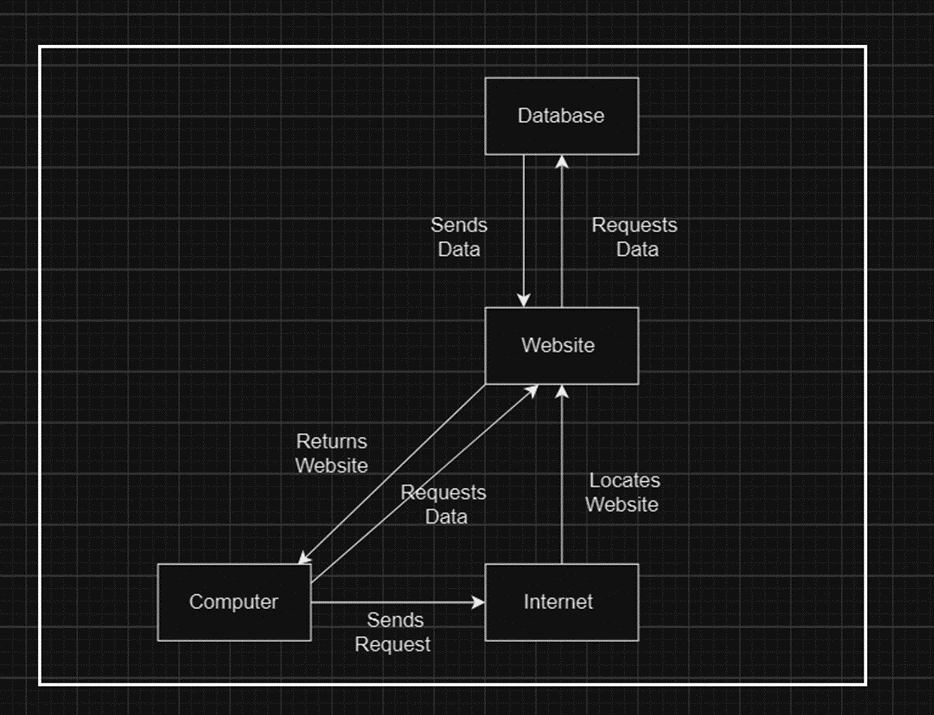
## 4.3 Process View



1. User Interaction (View):
   1. The user interacts with the View as usual, which includes the UI elements.
2. User Input Handling (Controller):
   1. The Controller receives the user input and processes it.
3. Request Handling (Controller):
   1. Based on the user's input, the Controller might need to interact with external services. For example, if a user schedules an appointment, the system might need to interact with Google Calendar API to update the schedule.
4. Business Logic (Model):
   1. The Controller interacts with the Model to perform operations. This is where interactions with external interfaces occur.
   2. The Model is responsible for managing the interactions with external systems or services. For example, it might send a request to the Google Calendar API to create an event.
5. External Interface Interaction
   1. This is where the communication with external services or systems takes place. The Model layer encapsulates the logic for making requests to these interfaces.
   2. For example, the Model would facilitate communication with the Google Calendar API to update schedules.
6. Data Manipulation (Model):
   1. After receiving a response from an external interface, the Model might need to process and manipulate the data before returning it to the Controller.
7. Database Interaction
   1. If necessary, the Model layer may also interact with the database to store any relevant information related to the external interaction.
8. Response Handling (Controller):
   1. The Controller processes the response from the Model, which may include data from the external interface interaction.
9. View Update (View):
   1. The Controller provides data to the View, which updates the UI accordingly, potentially reflecting information obtained from the external interface interaction.

## 4.4 Physical view

1. **Computer:** Computer is booted up which allows the user to interact with the website.
2. I**nternet:**  In order to connect to the website, the user must connect to the internet with their computer
3. **Website:** Once on the internet, the user can connect to the website to access its features. Once connected, the website is displayed on the users device
4. **Request**: When the user uses the website and views data, they make a request in order to see things like appointments and locations.
5. **Database**: All information is stored on the database once clicked on something, the website makes a request to the database in order to access the information.
6. **Approved**: Once the request is approved, the information is sent to the website which then gets sent back to the users device for viewing.



**5. Policies and Tactics**

**5.1 Choice of which specific products used**

MySQL Workbench,

Visual Studio

**5.2 Plans for ensuring requirements traceability**

First step is to make a clear outline of the requirements needed for this software by documenting them in the Software Requirements Document (SRD). With the unique number of identifiers such as requirements 2.1,2.2,2.3, etc. we are be able to trace requirements throughout the project and help us easily cross reference with the SRD document.

**5.3 Plans for testing the software**

Set up test environments that closely mimic the production environment to ensure realistic testing conditions.

**5.4 Engineering Trade-offs**

User Experience vs. Load Times: Optimize web application for fast loading times which can result in a less appealing user interface. We need to find a balance between a visually appealing interface and quick load times.

Data Security vs. Accessibility: Striking a balance between data security and accessibility is crucial. Security measures, like encryption, can put some restrictions to what information users can access.

**5.5 Coding Guidelines and Conventions**

Code Documentation: All code is documented and includes comments to explain the purpose of the code and how it works

Minimize Code: Minimize length of functions and methods to improve the readability and maintainability of code

**5.6 The protocol of one or more subsystems, modules, or subroutines**

* Specify how the system schedules appointments, taking into account the availability of healthcare providers, resources, and appointment types
* Patient and Staff Authentication Protocol
  + Authorization methods to ensure that each account has the proper authorizations
* We will document each element of the protocol with clear and concise comments.
* Clearly explain the purpose, inputs, outputs, and usage of each function from the code.
* Maintain comprehensive documentation for team members who will work with the protocol.

**5.7 The choice of a particular algorithm or programming idiom (or design pattern) to implement portions of the system's functionality**

* Encryption Algorithm
  + We will implement data encryption to account for login information in order to secure patient information

**5.8 Plans for maintaining the software**

* Documentation
  + Clearly and regularly update documentation of all aspects of the project
* Knowledge
  + All members of the projects should be equally knowledgeable to maintain the software
* Bug Tracking
  + Keep track of all bugs and create a system to prioritize bugs in the software from most to least critical
* Backup Recovery
  + We will maintain a backup of all data since we are dealing with critical patient information

**5.9 Interfaces for end-users, software, hardware, and communications**

End-Users

* Usability Policy: Ensure that user interfaces are designed with a focus on user experience and usability
* Tactics
  + User-Centered Design
  + Accessibility Compliance

Communication

* Encryption and Security Policy: Mandated encryption and security measures for data transmitted over communication interfaces.

**5.10 Hierarchical organization of the source code into its physical components (files and directories).**

1. Root Directory
2. Source Code Directory
3. Subdirectories based on functionality of code

**5.11 How to build and/or generate the system's deliverables (how to compile, link, load, etc.**

* Compilation
  + Microsoft Visual C#
* Database and C# linkage
  + MySQL Workbench

**5.12 Describe tactics such as abstracting out a generic DatabaseInterface class, so that changing the database from MySQL to Oracle or PostGreSQL is simply a matter of rewriting the DatabaseInterface class.**

* We will write pseudocode that will help us design classes in the case that we end up having to switch platform

**6. Detailed System Design**

**6.1 Login**

**6.1.1 Responsibilities:** Gives users access to the website upon successfully verifying their credentials.

**6.1.2 Constraints:** Permits login only for accounts whose data is recorded in the database, ensuring valid user access.

**6.1.3 Composition:** An input interface where users provide login information via a keyboard and an input manager to handle user-provided data in the form of an input field.

**6.1.4 Uses/Interactions:** Interacts with the database for user information validation, authenticator for verification, security measures for data safety, and the website interface for successful login display.

**6.1.5 Resources:** Involves website traffic and database interactions. SQL contains safeguards that prevent lockup issues.

**6.1.6 Interface/Export:** Allows access to the entire website post-login, offering various features based on the account type, enabling appointment-making, viewing medical records, and accessing hospital information.

**6.2 Create Account**

**6.2.1 Responsibilities:** Allows user account creation while restricting the creation of admin accounts, allowing only regular user account creation.

**6.2.2 Constraints:** Prevents the creation of admin accounts through this process.

**6.2.3 Composition:** Involves the UI for interaction, database operations to log user information securely, and security measures for data protection.

**6.2.4 Uses/Interactions:** Users interact via the website's 'create account' button, while the database stores user information upon account creation.

**6.2.5 Resources:** Involves database alterations when the account is successfully created. SQL contains safeguards that prevent lockup issues.

**6.2.6 Interface/Export:** Enables account creation and access to various website functionalities, such as appointment-making and accessing hospital information.

**6.3 Database**

**6.3.1 Responsibilities**

Responsible for storing information related to appointments, including patient details, doctor schedules, and available time slots.

**6.3.2 Constraints**

Must comply with proper regulations related to patient data protection

**6.3.3 Composition**

Established tables to store doctor/patient information and appointment data

**6.3.4 User/Interactions**

Supports user authentication so every user only has access to specific data

Interacts with the appointment booking system allowing to create, modify, and cancel appointments

**6.3.5 Resources**

Sufficient storage allocation

**6.3.6 Interface/Export**

Methods to export data are in place for backup purposes

**6.4 Appointment Scheduler**

**6.4.1 Responsibilities:** Enables users to create and manage their appointments within the system.

**6.4.2 Constraints:** Allows appointment creation only in available slots; higher-ranking accounts may have exceptions to this constraint.

**6.4.3 Composition:** Involves A UI appointment selection and cancellation alongside a calendar for visualizing appointments.

**6.4.4 Uses/Interactions:** The database is updated based on user actions (creating or canceling appointments), and the website dynamically adjusts available slots based on user choices.

**6.4.5 Resources:** SQL contains safeguards that prevent lockup issues.

**6.4.6 Interface/Export:** Allows users to manage appointments, relying on the website and database for functionality.

**6.5 Medical Records**

**6.5.1 Responsibilities**

Designed to store and display patient health information to appropriate users

**6.5.2 Constraints**

Follow and comply with relevant healthcare regulations(HIPPA)

Insuring the accuracy and integrity of data information

**6.5.3 Composition**

Appointment history

Patient information

Lab results

**6.5.4 User/Interactions**

Authorized personnel are only allowed to update and maintain

Patients are able to view their own medical records

**6.5.5 Resources**

Secure database to store and view medical records

**6.5.6 Interface/Export**

UI interface to view medical records

**6.6 Admin Controls**

**6.6.1 Responsibilities**

Modify/Delete accounts

Manage access levels for accounts

Control system settings such as appointment slots, availabilities, and work hours

**6.6.2 Constraints**

Admins controls does not include access to view critical patient information

Secure authorization

**6.6.3 Composition**

UI interaction with the admin controls

Database - Create new time slots to store in database and manage account level authorizations

**6.6.4 User/Interactions**

Patients rely on admin to maintain appointments

System configurations can be modified

**6.6.5 Resources**

Relies on UI to manage system settings

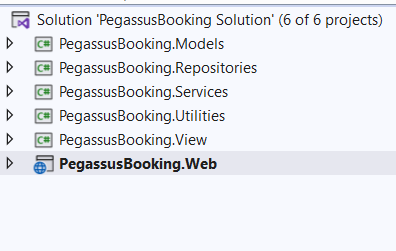
Database management settings

**6.6.6 Interface/Export**

Admin dashboard for appointment modification access and account management access

**7. Detailed Lower level Component Design**

Our Project is broken into 6 sub project files that each handle and assist with other parts of the website. These project files are seen below



**7.1 Models Layer**

**7.1.1: Description**

The PegassusBooking.Models project is tasked with data migration to and from the database. It’s comprised of several .cs files each in charge of one table from the MySql database

**7.1.1.1 Classification: Model Class (User.cs)**

* This class is specifically designed to manage user-related data and facilitate interactions with the MySQL database.

**7.1.1.2 Processing Narrative (PSPEC)**

* The User.cs file within the model layer serves as an intermediary between the application and the MySQL database, specifically managing user-related data. It follows a Plain Old CLR Object (POCO) approach, encapsulating data access logic and ensuring seamless communication with the database.
* It's in charge of the login and user classification aspect of the website.
  + User Classifications are as follow
    - Patient
    - Doctor
    - Admin
    - Nurse

**7.1.1.3 Interface Description**

* The User.cs interface is simply pulling ‘User’ information that’s requested from the database and sending that information back should anything be updated, deleted or added. It creates an entity with properties corresponding to the fields of the User table, allowing seamless mapping between the application and the database.

**7.1.1.4 Processing Detail**

* The processing details for the User.cs file within the model layer are outlined below:

**7.1.1.4.1 Design Class Hierarchy**

* The User.cs file is part of the model layer, specifically designed to manage User-related data.
* It inherits from the Asp.net.EntityFrameworks “DbContext” class which ensures a consistent approach across different entities.

**7.1.1.4.2 Restrictions/Limitations**

* No specific restrictions or limitations.

**7.1.1.4.3 Performance Issues**

* Optimized for efficient database interactions using Entity Framework.

**7.1.1.4.4 Design Constraints**

* Follows the conventions and patterns of Entity Framework for data access.

**7.1.1.4.5 Processing Detail For Each Operation**

* The model layer subclasses follow the POCO get and set approach to get and store data from the database whenever it is called by another layer.

**7.1.2.1 Classification: Model Class (Patient.cs)**

* This class falls under the category of a model class within the model layer. It is specifically designed to manage patient-related data and facilitate interactions with the MySQL database.

**7.1.2.2 Processing Narrative (PSPEC)**

* The Patient.cs file within the model layer serves as an intermediary between the application and the MySQL database, specifically managing patient-related data. It follows a Plain Old CLR Object (POCO) approach, encapsulating data access logic and ensuring seamless communication with the database.

**7.1.2.3 Interface Description**

* The Patient.cs interface is simply pulling ‘Patient’ information that’s requested from the database and sending that information back should anything be updated, deleted or added. It creates an entity with properties corresponding to the fields of the patient table, allowing seamless mapping between the application and the database.

**7.1.2.4 Processing Detail**

* The processing details for the Patient.cs file within the model layer are outlined below:

**7.1.2.4.1 Design Class Hierarchy**

* The Patient.cs file is part of the model layer, specifically designed to manage patient-related data.
* It inherits from the Asp.net.EntityFrameworks “DbContext” class which ensures a consistent approach across different entities.

**7.1.2.4.2 Restrictions/Limitations**

* No specific restrictions or limitations.

**7.1.2.4.3 Performance Issues**

* Optimized for efficient database interactions using Entity Framework.

**7.1.2.4.4 Design Constraints**

* Follows the conventions and patterns of Entity Framework for data access.

**7.1.2.4.5 Processing Detail For Each Operation**

* The model layer subclasses follow the POCO get and set approach to get and store data from the database whenever it is called by another layer.

**7.1.3.1 Classification: Model Class (Doctor.cs)**

* This class falls under the category of a model class within the model layer. It is specifically designed to manage doctor-related data and facilitate interactions with the MySQL database.

**7.1.3.2 Processing Narrative (PSPEC)**

* The Doctor.cs file within the model layer serves as an intermediary between the application and the MySQL database, specifically managing doctor-related data. It follows a Plain Old CLR Object (POCO) approach, encapsulating data access logic and ensuring seamless communication with the database.

**7.1.3.3 Interface Description**

* The Doctor.cs interface is simply pulling ‘Doctor’ information that’s requested from the database and sending that information back should anything be updated, deleted or added. It creates an entity with properties corresponding to the fields of the doctor table, allowing seamless mapping between the application and the database.

**7.1.3.4 Processing Detail**

* The processing details for the Doctor.cs file within the model layer are outlined below:

**7.1.3.4.1 Design Class Hierarchy**

* The Doctor.cs file is part of the model layer, specifically designed to manage patient-related data.
* It inherits from the Asp.net.EntityFrameworks “DbContext” class which ensures a consistent approach across different entities.

**7.1.3.4.2 Restrictions/Limitations**

* No specific restrictions or limitations.

**7.1.3.4.3 Performance Issues**

* Optimized for efficient database interactions using Entity Framework.

**7.1.3.4.4 Design Constraints**

* Follows the conventions and patterns of Entity Framework for data access.

**7.1.3.4.5 Processing Detail For Each Operation**

* The model layer subclasses follow the POCO get and set approach to get and store data from the database whenever it is called by another layer.

**7.1.4.1 Classification: Model Class (Admin.cs)**

* This class falls under the category of a model class within the model layer. It is specifically designed to manage doctor-related data and facilitate interactions with the MySQL database.

**7.1.4.2 Processing Narrative (PSPEC)**

* The Admin.cs file within the model layer serves as an intermediary between the application and the MySQL database, specifically managing admin-related data. It follows a Plain Old CLR Object (POCO) approach, encapsulating data access logic and ensuring seamless communication with the database.

**7.1.4.3 Interface Description**

* The Admin.cs interface is simply pulling ‘Admin’ information that’s requested from the database and sending that information back should anything be updated, deleted or added. It creates an entity with properties corresponding to fields of the admin table, allowing seamless mapping between the application and the database.

**7.1.4.4 Processing Detail**

* The processing details for the Admin.cs file within the model layer are outlined below:

**7.1.4.4.1 Design Class Hierarchy**

* The Admin.cs file is part of the model layer, specifically designed to manage patient-related data.
* It inherits from the Asp.net.EntityFrameworks “DbContext” class which ensures a consistent approach across different entities.

**7.1.4.4.2 Restrictions/Limitations**

* No specific restrictions or limitations.

**7.1.4.4.3 Performance Issues**

* Optimized for efficient database interactions using Entity Framework.

**7.1.4.4.4 Design Constraints**

* Follows the conventions and patterns of Entity Framework for data access.

**7.1.4.4.5 Processing Detail For Each Operation**

* The model layer subclasses follow the POCO get and set approach to get and store data from the database whenever it is called by another layer.

**7.1.5.1 Classification: Model Class (Hospitals.cs)**

* This class falls under the category of a model class within the model layer. It is specifically designed to manage hospital-related data and facilitate interactions with the MySQL database.

**7.1.5.2 Processing Narrative (PSPEC)**

* The Hospital.cs file within the model layer serves as an intermediary between the application and the MySQL database, specifically managing hospital-related data. It follows a Plain Old CLR Object (POCO) approach, encapsulating data access logic and ensuring seamless communication with the database.

**7.1.5.3 Interface Description**

* The Hospitals.cs interface is simply pulling ‘Hospital’ information that’s requested from the database and sending that information back should anything be updated, deleted or added. It creates an entity with properties corresponding to the fields of the hospital table, allowing seamless mapping between the application and the database.

**7.1.5.4 Processing Detail**

* The processing details for the Hospitals.cs file within the model layer are outlined below:

**7.1.5.4.1 Design Class Hierarchy**

* The Hospitals.cs file is part of the model layer, specifically designed to manage patient-related data.
* It inherits from the Asp.net.EntityFrameworks “DbContext” class which ensures a consistent approach across different entities.

**7.1.5.4.2 Restrictions/Limitations**

* No specific restrictions or limitations.

**7.1.5.4.3 Performance Issues**

* Optimized for efficient database interactions using Entity Framework.

**7.1.5.4.4 Design Constraints**

* Follows the conventions and patterns of Entity Framework for data access.

**7.1.5.4.5 Processing Detail For Each Operation**

* The model layer subclasses follow the POCO get and set approach to get and store data from the database whenever it is called by another layer.

**7.1.5.1 Classification: Model Class (Appointments.cs)**

* This class falls under the category of a model class within the model layer. It is specifically designed to manage appointment-related data and facilitate interactions with the MySQL database.

**7.1.5.2 Processing Narrative (PSPEC)**

* The Appointment.cs file within the model layer serves as an intermediary between the application and the MySQL database, specifically managing appointment-related data. It follows a Plain Old CLR Object (POCO) approach, encapsulating data access logic and ensuring seamless communication with the database.

**7.1.5.3 Interface Description**

* The Appointment.cs interface is simply pulling ‘Appointment’ information that’s requested from the database and sending that information back should anything be updated, deleted or added. It creates an entity with properties corresponding to the fields of the Appointment table, allowing seamless mapping between the application and the database.

**7.1.5.4 Processing Detail**

* The processing details for the Appointment.cs file within the model layer are outlined below:

**7.1.5.4.1 Design Class Hierarchy**

* The Appointment.cs file is part of the model layer, specifically designed to manage patient-related data.
* It inherits from the Asp.net.EntityFrameworks “DbContext” class which ensures a consistent approach across different entities.

**7.1.5.4.2 Restrictions/Limitations**

* No specific restrictions or limitations.

**7.1.5.4.3 Performance Issues**

* Optimized for efficient database interactions using Entity Framework.

**7.1.5.4.4 Design Constraints**

* Follows the conventions and patterns of Entity Framework for data access.

**7.1.5.4.5 Processing Detail For Each Operation**

* The model layer subclasses follow the POCO get and set approach to get and store data from the database whenever it is called by another layer.

## 7.2 Repositories/Database Layer

**Description:**

* The PegassusBooking.Repositories serves as the repository layer in the ASP.Net Core MVC Architecture. It is responsible for handling data access logic and communication with the underlying database.
* It is split into 3 main parts:

oGenericRepository

oUnitOfWork

oApplicationDbContext.cs

#### 7.2.1.1 Classification: GenericRepository

· The GenericRepository.cs file, serves as a generic repository implementation in the ASP.NET Core MVC framework. It provides a flexible and reusable way to interact with the different entities in the database

#### 7.2.1.2 Processing Narrative (PSPEC)

· The GenericRepository.cs file streamlines the data access by offering a generic approach for CRUD operations on various enitities. It utilizes Entity Framework Core for database interactions.

#### 7.2.1.3 Interface Description

· The IGenericRepository.cs file defines the generic methods for the CRUD operations, ensuring consistency across all the applications enities.

#### 7.2.1.4 Processing Detail

##### 7.2.1.4.1 Design Class Hierarchy

· GenericRepository<TEntity>: This Class implements the IGenericRepository.cs interface and provides concrete implementations for generic CRUD operations. It uses the Entity Framework Core’s DbContext for data access and the Models layer in order to facilitate smooth data access operations.

##### 7.2.1.4.2 Restrictions/Limitations

· The GenericRepository focuses on common data access operations so anything more complex would require another repository with unique logic.

##### 7.2.1.4.3 Performance Issues

· Utilizes asynchronous programming to maintain responsiveness.

· Implements efficient queries to minimize database interactions.

##### 7.2.1.4.4 Design Constraints

· Follows the repository pattern, abstracting away database-specific details.

· Promotes reusability by handling various entities through generic operations built into the framework.

##### 7.2.1.4.5 Processing Detail for each operation

· **Add and AddAsync:** adds a new entity to the database set.

· **Delete and DeleteAsync:** deletes an entity from the database set.

· **Add and AddAsync:** adds a new entity to the database set.

· **Dispose and DisposeAsync:** disposes the memory allocation for the database set.

· **GetById and GetByIdAsync:** retrieves a specific entity from the database set using a specific identifier.

· **Update and UpdateAsync:** Modifies an existing entity in the database.

### 7.2.2.1 Classification: UnitofWork

· The UnitofWork.cs file serves as a unit of work implementation of the ASP.NET Core MVC framework. It coordinates transactions across multiple repositories, ensuring data consistency.

#### 7.2.2.2 Processing Narrative (PSPEC)

· The UnitofWork.cs file consolidates and manages transactions for the repository layer within a single operation. It helps maintain integrity and consistency across the application.

#### 7.2.2.3 Interface Description

· The IUnitofWork interface defines methods for saving changes and accessing different repositories.

#### 7.2.2.4 Processing Detail

##### 7.2.2.4.1 Design Class Hierarchy

· UnitofWork.cs implements the IUnitofWork interface and manages transactions across the repository layer. It holds instances of the repository and ensures a consistent context for the database operations.

**7.2.2.4.2 Restrictions/Limitations**

· The UnitofWork.cs file focuses on coordinating transactions and may require adjustments for specific scenarios, such as distributed transactions.

**7.2.2.4.3 Performance Issues**

· Utilizes dependency injection to manage repository instances efficiently.

· Implements asynchronous saving of changes to maintain responsiveness.

**7.2.2.4.4 Design Constraints**

· Adheres to the unit of work pattern, providing a higher-level abstraction for managing transactions.

· Promotes modular and testable code by abstracting away database interactions.

**7.2.2.4.5 Processing Detail for each operation**

· **SaveChangesAsync:**  Persists changes made in the repositories to the underlying database.

### 7.2.3.1 Classification: ApplicationDbContext.cs

· The ApplicationDbContext.cs file serves as the DbContext for the ASP.NET Core MVC application. It manages the database connection and entity relationships. It’s the foundation for database interaction in the website.

#### 7.2.3.2 Processing Narrative (PSPEC)

· The ApplicationDbContext.cs file provides a link between the application and the database. It defines the entity models and their relationships, facilitating database operations.

#### 7.2.3.3 Interface Description

· The DbContext Class is extended to create ApplicationDbContext.cs It includes ‘Dbset’ properties for each entity in the application.

#### 7.2.3.4 Processing Detail

##### 7.2.3.4.1 Design Class Hierarchy

· ApplicationDbContext inherits from DbContext and represents the main entry point for database interaction in our website.

**7.2.3.4.2 Restrictions/Limitations**

· Requires proper configuration of the DbContextOptions during instantiation.

**7.2.3.4.3 Performance Issues**

· Utilizes lazy loading for Entity relationships to optimize performance.

· Implements dependency injection for managing the database connection.

**7.2.3.4.4 Design Constraints**

· Adheres to the Entity Framework Core conventions for defining entities and their relationships.

· Can be extended to include additional configurations such as seed data or custom mappings.

**7.2.3.4.5 Processing Detail for each operation**

· **Constructor:** Takes DbContextOptions as parameter allowing configuration during Application Start up.

· **DbSet Properties:**  Persists changes made in the repositories to the underlying database.

## 7.3 Services Layer

**Description:**

· The PegassusBooking.Services represents the services layer in the ASP.NET Core Mvc application. The service layer contains application-specific logic, handling communication between the Web Layer and the repository layer.

## 7.4 Utilities/External Interfaces Layer

**Description:**

· The PegassusBooking.Utilities layer serves as the utility layer in the ASP.Net Core MVC Architecture. It is responsible for providing utility functions catering to various common tasks within the website.

#### 7.4.1.1 Classification: EmailSender.cs

· The EmailSender.cs file is a class that will send emails for the notification service of the website

#### 7.4.1.2 Processing Narrative (PSPEC)

· The EmailSender class is responsible for sending emails in the ASP.NET Core MVC framework. It implements the IEmailSender interface from the Microsoft.AspNetCore.Identity.UI.Services library, providing email functionality for user authentication, password recovery, and other notification needs.

#### 7.4.1.3 Interface Description

· **The IEmailSender Interface:**

o SendEmailAsync(string email, string subject, string htmlMessage): Sends an email asynchronously with the specified email address, subject, and HTML message.

#### 7.4.1.4 Processing Detail

##### 7.4.1.4.1 Design Class Hierarchy

· Parent Class: None

· Child Class: None

##### 7.4.1.4.2 Restrictions/Limitations

· The class is dependent on external email server configurations.

##### 7.4.1.4.3 Performance Issues

· Email sending performance is subject to the responsiveness of the configured email server

##### 7.4.1.4.4 Design Constraints

· Configuration parameters, such as SMTP server details, need to be appropriately set in the application settings.

##### 7.4.1.4.5 Processing Detail for each operation

· **SendEmailAsync**:

o Initiates the asynchronous process of sending an email using the configured email server.

· Parameters:

o email: The recipient's email address.

o subject: The subject of the email.

o htmlMessage: The HTML content of the email.

· This operation handles the actual sending of the email through the configured email server.

## 7.5 View/UI Layer

**Description:**

· The PegassusBooking.View layer in the ASP.NET Core MVC framework is responsible for presenting the user interface and rendering dynamic content based on data received from the Controller. It includes components for defining the layout, structure, and appearance of web pages.

#### 7.5.1.1 Classification

· This is an overarching Layer project containing the html, css, and razor pages of the project.

#### 7.5.1.2 Processing Narrative (PSPEC)

· The view layer interacts with the controllers within the Web layer of the projects and updates the view accordingly based on the information pulled from the controllers.

#### 7.5.1.3 Interface Description

· The View Layer interfaces with the Controller to receive data for presentation.

· Utilizes Razor syntax to embed C# code into HTML for dynamic content rendering.

· Supports the use of HTML helpers for generating standard HTML elements and forms.

· Interacts with the Model to display data retrieved from the underlying data sources.

#### 7.5.1.4 Processing Detail

##### 7.5.1.4.1 Design Class Hierarchy

· **Components:**

o Razor Views: CSHTML files containing a mix of HTML and C# code.

o HTML Helpers: Pre-built C# methods for generating HTML elements.

o Layout Pages: Define the overall structure shared across multiple views.

o View Components: Reusable components for rendering complex UI elements.

##### 7.5.1.4.2 Restrictions/Limitations

· Views focus on presentation logic and avoid complex business logic.

· Dependency on Razor syntax for embedding C# code within HTML.

##### 7.5.1.4.3 Performance Issues

· Rendering performance may be affected by the complexity of views and the amount of data processed.

##### 7.5.1.4.4 Design Constraints

· Views are designed to be stateless and rely on the Controller for data.

##### 7.5.1.4.5 Processing Detail for each operation

· **Render Views:**

o Views render HTML dynamically based on data received from the Controller.

o Use of Razor syntax enables the embedding of C# code for dynamic content.

· **Interact with Model:**

o Views interact with the Model to display data to users.

o Binding data received from the Controller to the UI elements.

· The View Layer plays a crucial role in presenting a user-friendly interface and ensuring a seamless user experience within the ASP.NET Core MVC application.

## 7.6 Web/Control Layer

**Description:**

· The PegassusBooking.Web file in the ASP.NET Core MVC framework serves as the entry point for handling HTTP requests and managing the overall application flow. It orchestrates the interaction between the user interface (View Layer) and the underlying business logic.

#### 7.6.1.1 Classification

· This is an overarching Layer project that contains the https requests, the websites appsettings, the controllers, bootstrap helper files and scaffolding Areas.

#### 7.6.1.2 Processing Narrative (PSPEC)

· The Web layer interacts with the model layer and the view layer through its controller classes, as well as the utilities, repositories and service layers to better facilitate these requests and inputs.

#### 7.6.1.3 Interface Description

· HTTP Requests Handling:

o Manages incoming HTTP requests from clients.

o Routes requests to the appropriate Controller based on defined routes.

· Response Generation:

o Generates HTTP responses for clients.

o Utilizes Views from the View Layer for rendering HTML content.

· Middleware Processing:

o Engages with middleware components for various tasks such as authentication, logging, and error handling.

#### 7.6.1.4 Processing Detail

##### 7.6.1.4.1 Design Class Hierarchy

· **Components:**

o Startup Class: Configures services and middleware for the application.

o Controller Classes: Handle specific HTTP requests, invoke services, and provide data to Views.

o Routing: Defines URL patterns and routes requests to the appropriate controllers.

##### 7.6.1.4.2 Restrictions/Limitations

Dependency on ASP.NET Core MVC framework conventions for routing and request handling.

##### 7.6.1.4.3 Performance Issues

· Performance considerations include request processing time and resource utilization.

##### 7.6.1.4.4 Design Constraints

· Adheres to the MVC architectural pattern, separating concerns between the Model, View, and Controller.

##### 7.6.1.4.5 Processing Detail for each operation

· **Request Handling:**

o Routes incoming HTTP requests to the appropriate Controller based on defined routes.

o Invokes Controller actions to process user input and retrieve data**.**

· **Response Generation:**

o Utilizes Views from the View Layer to generate HTML responses.

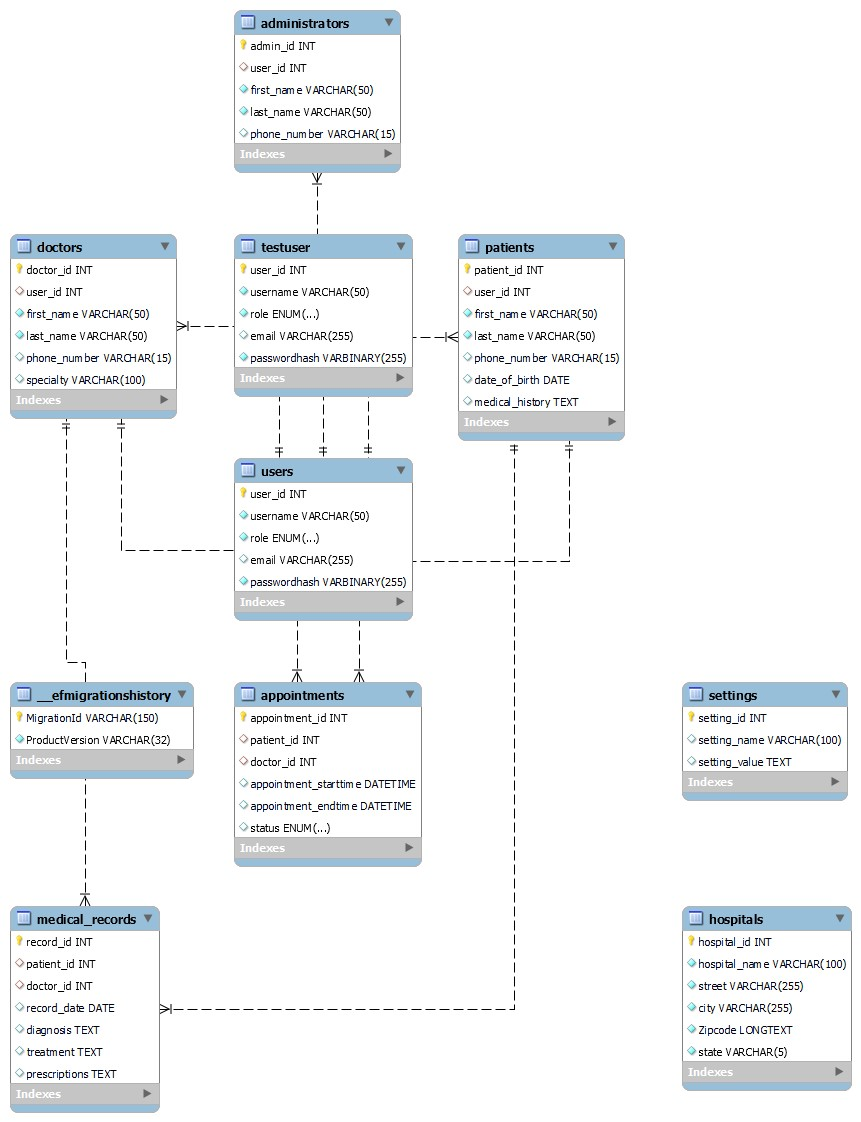
o Manages the flow of data between the Controller and the View.

· **Middleware Interaction:**

o Engages with middleware components configured in the Startup class.

o Middleware handles cross-cutting concerns such as authentication and logging.

**8. Database Design**

****

**8.1 Database Schema**

Pegasus Booking relies on a MySQL database to store and manage critical information related to users, appointments, medical records, and system configurations. The database schema is thoughtfully designed to ensure data integrity, support complex relationships, and facilitate efficient retrieval of information.

The database schema encompasses several key tables, each serving a specific purpose:

**8.1.1 Users Table**

**Purpose:** The users table is responsible for managing user authentication and roles within the system. It stores essential user information required for login and access control.

**Fields:**

* user\_id **(Primary Key)**: A unique identifier for each user.
* username: The username used for authentication.
* password: Encrypted password for secure access.
* role: The role of the user (patient, doctor, admin).

**8.1.2 Patients Table**

**Purpose:**  The patients table stores detailed information about registered patients. It includes personal details and medical history, enabling efficient patient management.

**Fields:**

* patient\_id (Primary Key): A unique identifier for each patient.
* user\_id (Unique): References the corresponding user.
* first\_name: First name of the patient.
* last\_name: Last name of the patient.
* email (Unique): Email address for communication.
* phone\_number: Contact number of the patient.
* date\_of\_birth: Date of birth for age verification.
* medical\_history: Detailed medical history and relevant information.
* insurance\_number: Patients insurance number

**8.1.3 Doctors Table**

**Purpose:** The doctors table contains information about registered doctors. It includes details about their specialties, allowing for proper assignment of appointments.

**Fields:**

* doctor\_id (Primary Key): A unique identifier for each doctor.
* user\_id (Unique): References the corresponding user.
* first\_name: First name of the doctor.
* last\_name: Last name of the doctor.
* email (Unique): Email address for communication.
* phone\_number: Contact number of the doctor.
* specialty: Area of medical specialization.

**8.1.4 Administrators Table**

**Purpose:** The administrators table holds information about system administrators. It includes contact details and serves as a record of authorized system administrators.

**Fields:**

* admin\_id (Primary Key): A unique identifier for each administrator.
* user\_id (Unique): References the corresponding user.
* first\_name: First name of the administrator.
* last\_name: Last name of the administrator.
* email (Unique): Email address for communication.
* phone\_number: Contact number of the administrator.

**8.1.5 Appointments Table**

**Purpose:** The appointments table manages scheduled appointments between patients and doctors. It records essential details about appointments, including start and end times.

**Fields:**

* appointment\_id (Primary Key): A unique identifier for each appointment.
* patient\_id (Nullable): References the patient for the appointment.
* doctor\_id: References the doctor associated with the appointment.
* appointment\_starttime: Date and time of the appointment start.
* appointment\_endtime: Date and time of the appointment end.
* status: Status of the appointment (scheduled, completed, canceled).

**8.1.6 Settings Table**

**Purpose:** The settings table stores system configuration settings that can be customized to tailor the system behavior according to specific requirements.

**Fields:**

* setting\_id (Primary Key): A unique identifier for each setting.
* setting\_name (Unique): Identifier for a specific configuration setting.
* setting\_value: Value associated with the configuration key.

**8.1.7 Medical Records Table**

**Purpose:** The medical\_records table records medical information about patients, including diagnoses, treatments, and prescriptions provided by doctors.

**Fields:**

* record\_id (Primary Key): A unique identifier for each medical record.
* patient\_id: References the patient associated with the record.
* doctor\_id: References the doctor associated with the record.
* record\_date: Date of the record entry.
* diagnosis: Diagnosis provided by the doctor.
* treatment: Treatment prescribed by the doctor.
* prescriptions: Medications prescribed by the doctor.

**8.2 Indexing and Optimization**

* To enhance query performance, appropriate indexes are applied to critical fields such as user IDs, appointment dates, and doctor IDs. This means that these fields are sorted and structured in a way that makes it faster to retrieve specific pieces of information. For example, by indexing user IDs, the database can quickly locate and retrieve data associated with a specific user. Additionally, appointment dates and doctor IDs are indexed to facilitate efficient retrieval of appointment records and doctor-specific information.
* Normalization techniques are also employed to minimize data redundancy and ensure data consistency. This involves organizing the database in a way that eliminates unnecessary duplication of information. For instance, rather than storing the same information in multiple places, it is organized in a logical manner, reducing storage requirements and enhancing data integrity.

**8.3 Data Security and Access Control**

* Access control measures are implemented to restrict unauthorized access to sensitive data. Role-based access control (RBAC) is utilized to ensure that users only have access to data and functionalities relevant to their role within the system. This means that different users (such as patients, doctors, and administrators) have specific permissions based on their roles. For instance, doctors may have access to patient records and appointment scheduling, while patients have access to their own medical history and appointment details.

**9. User Interface**

**9.1 Overview of User Interface**

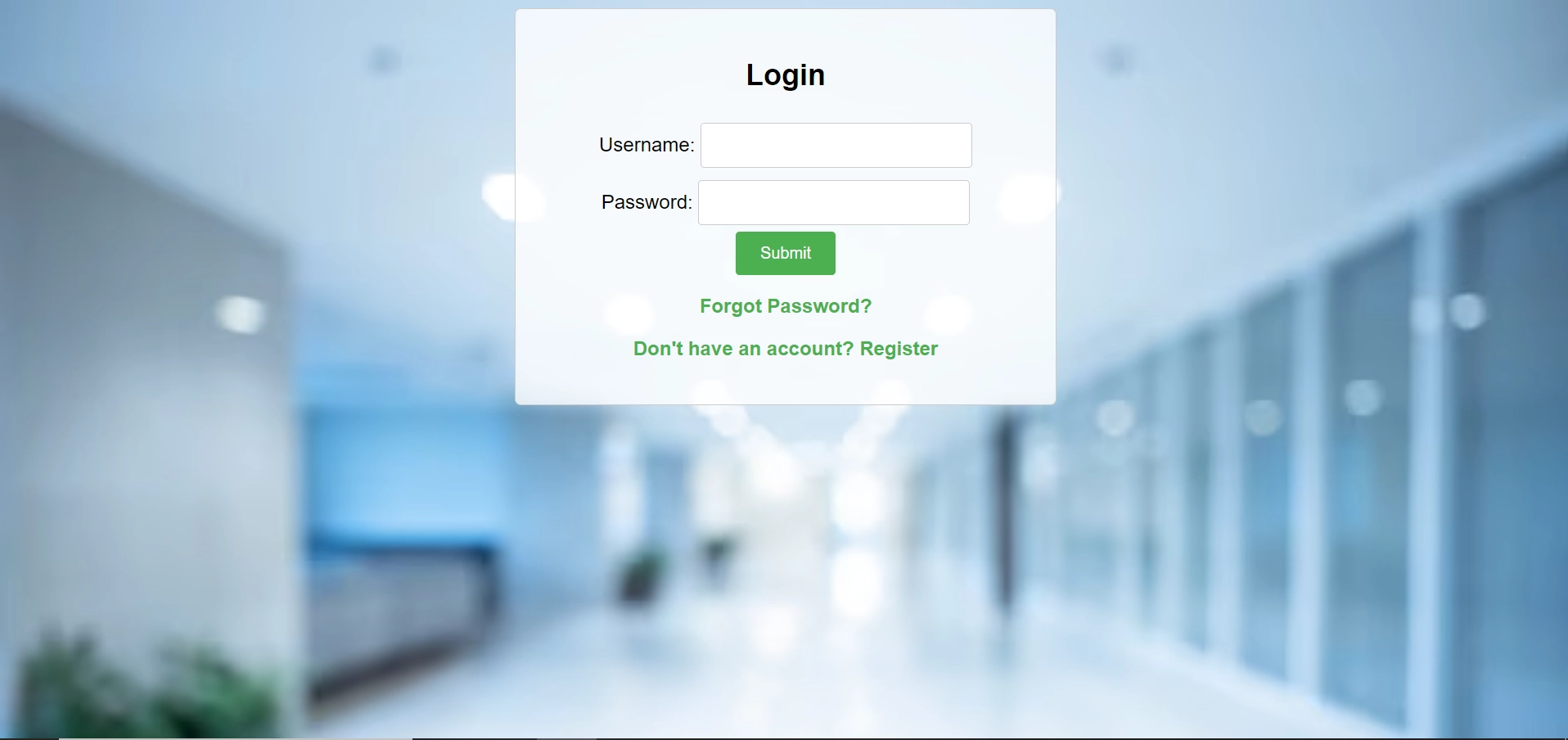
The website UI for Peggasus Booking consists of typical appointment booking features. These features include means to book an appointment via registration with name and personal information, canceling an appointment, and rescheduling it. Along with the scheduling there are features to view account information and information of doctors that are scheduled with a patient.

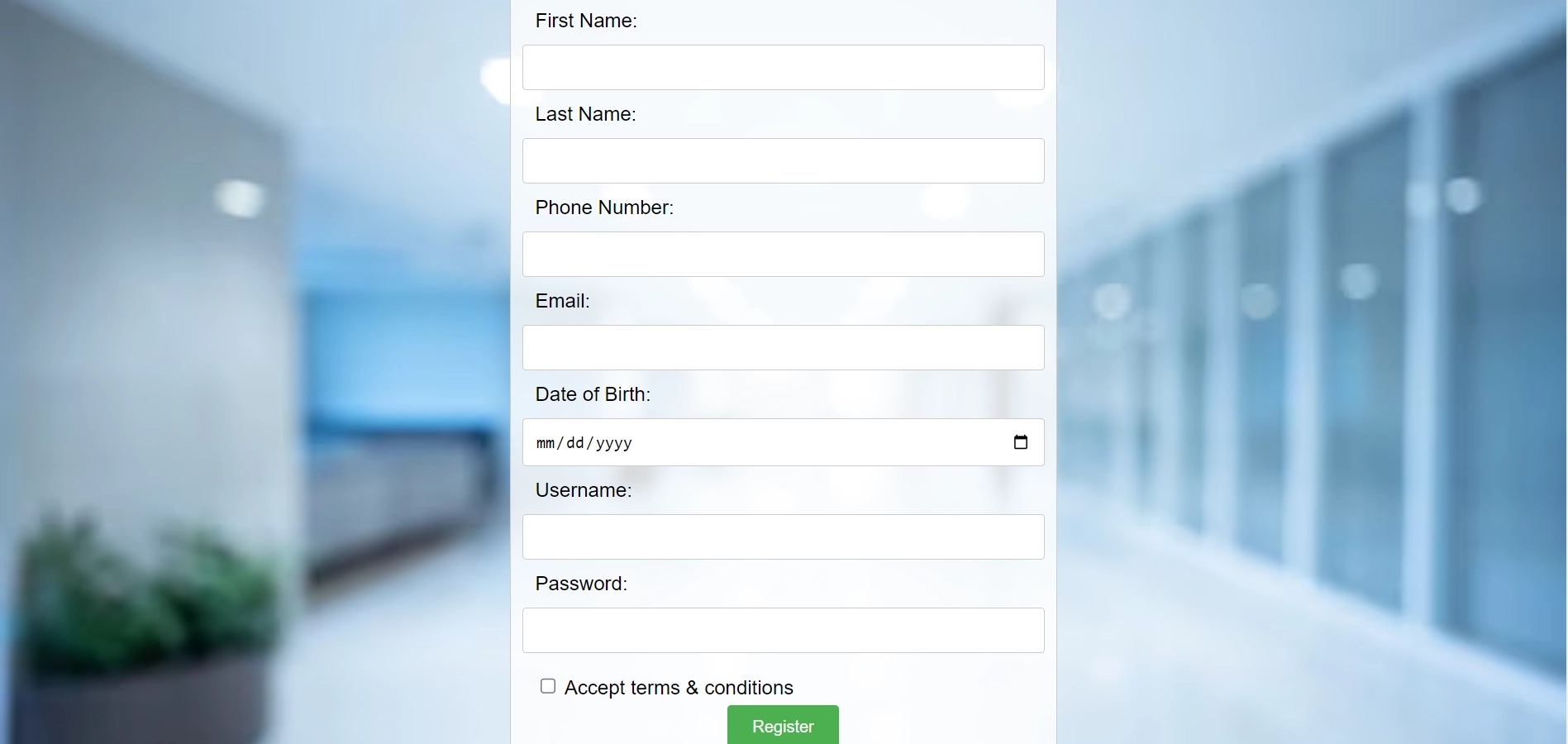
**9.2 UX Standards**

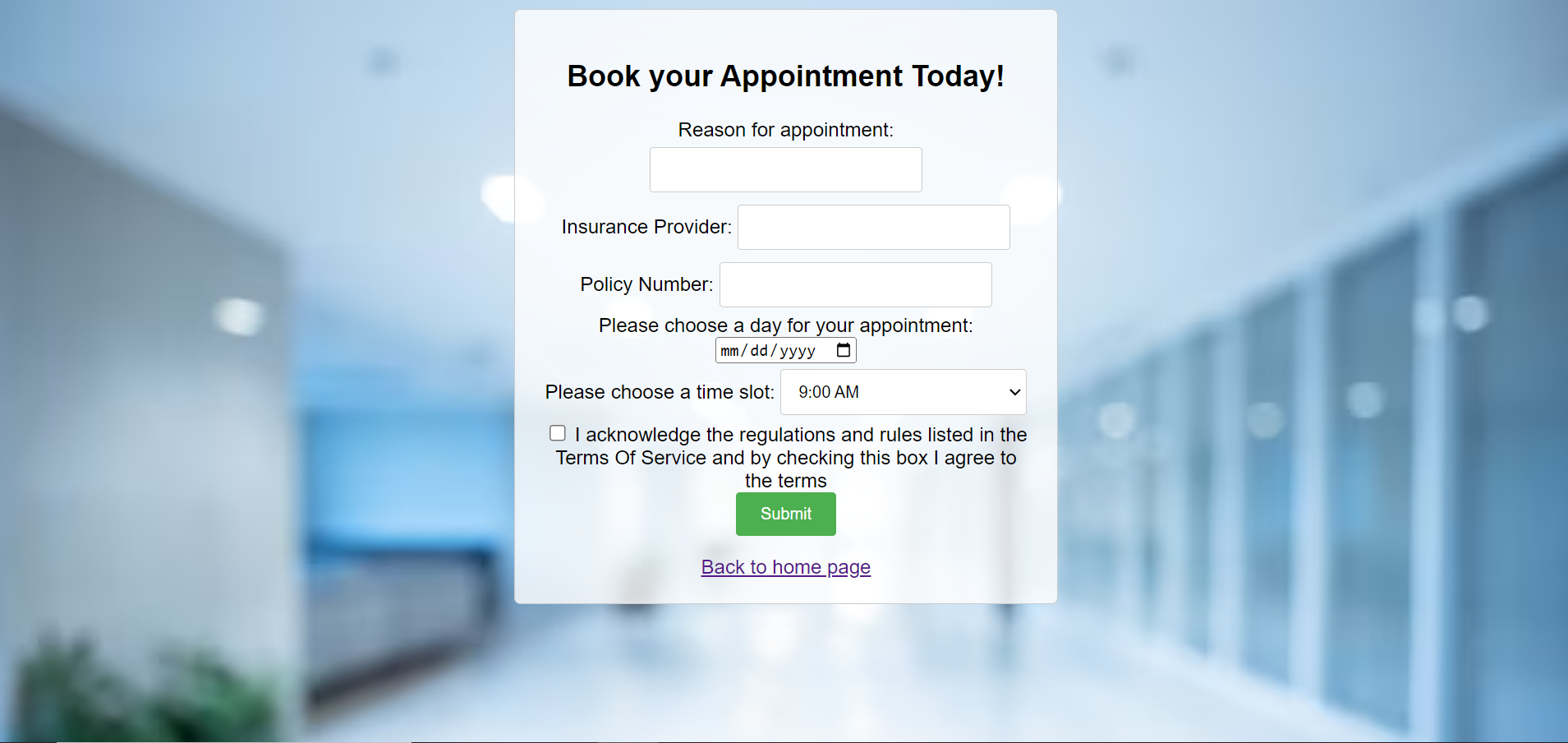
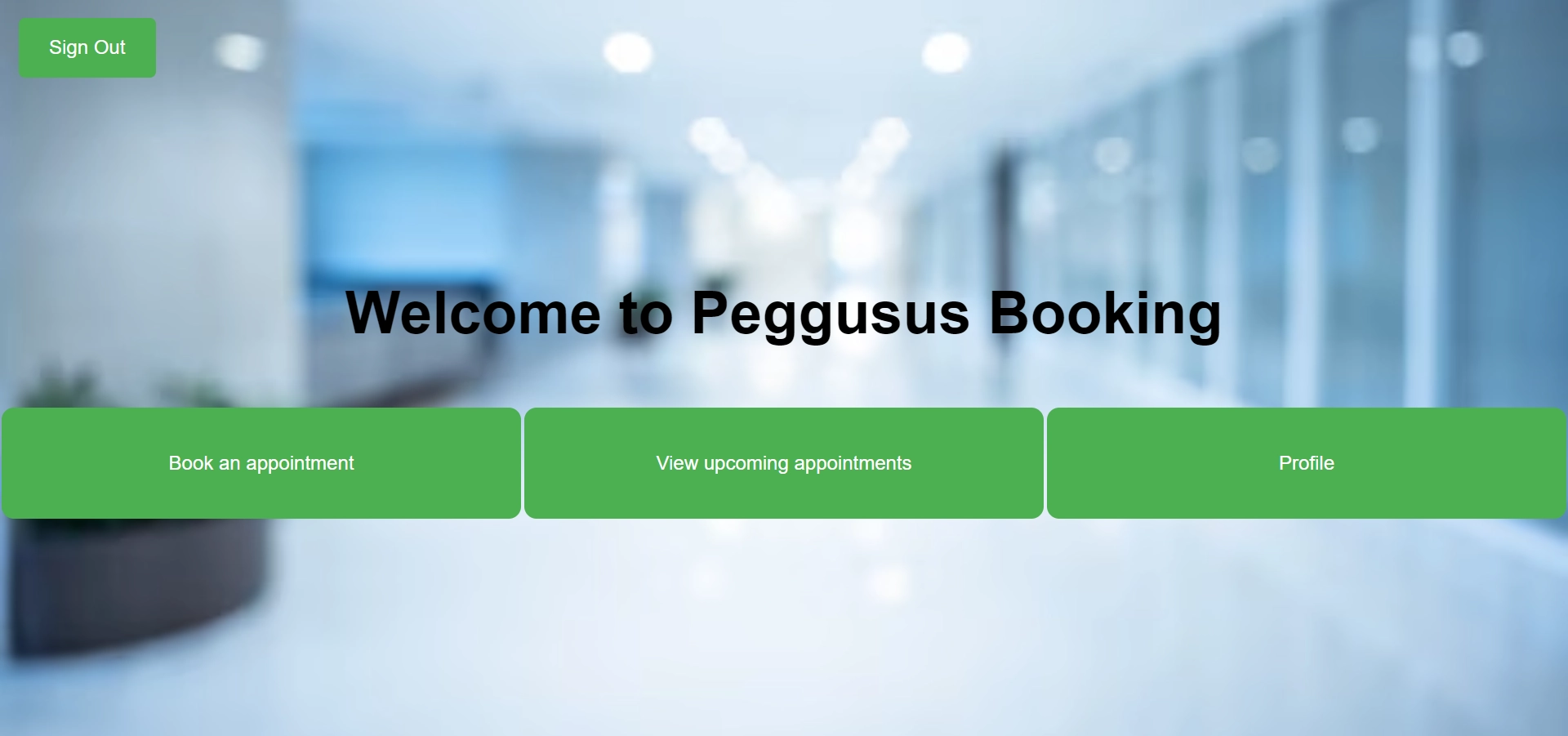
The format is suited for both desktop and mobile systems. Text and information must be suited to be adjusted to both platforms and be viewable at any resolution. Information given to the website will be saved internally, however this means that online information, excluding set appointment times and dates, won't be able to be viewed offline.

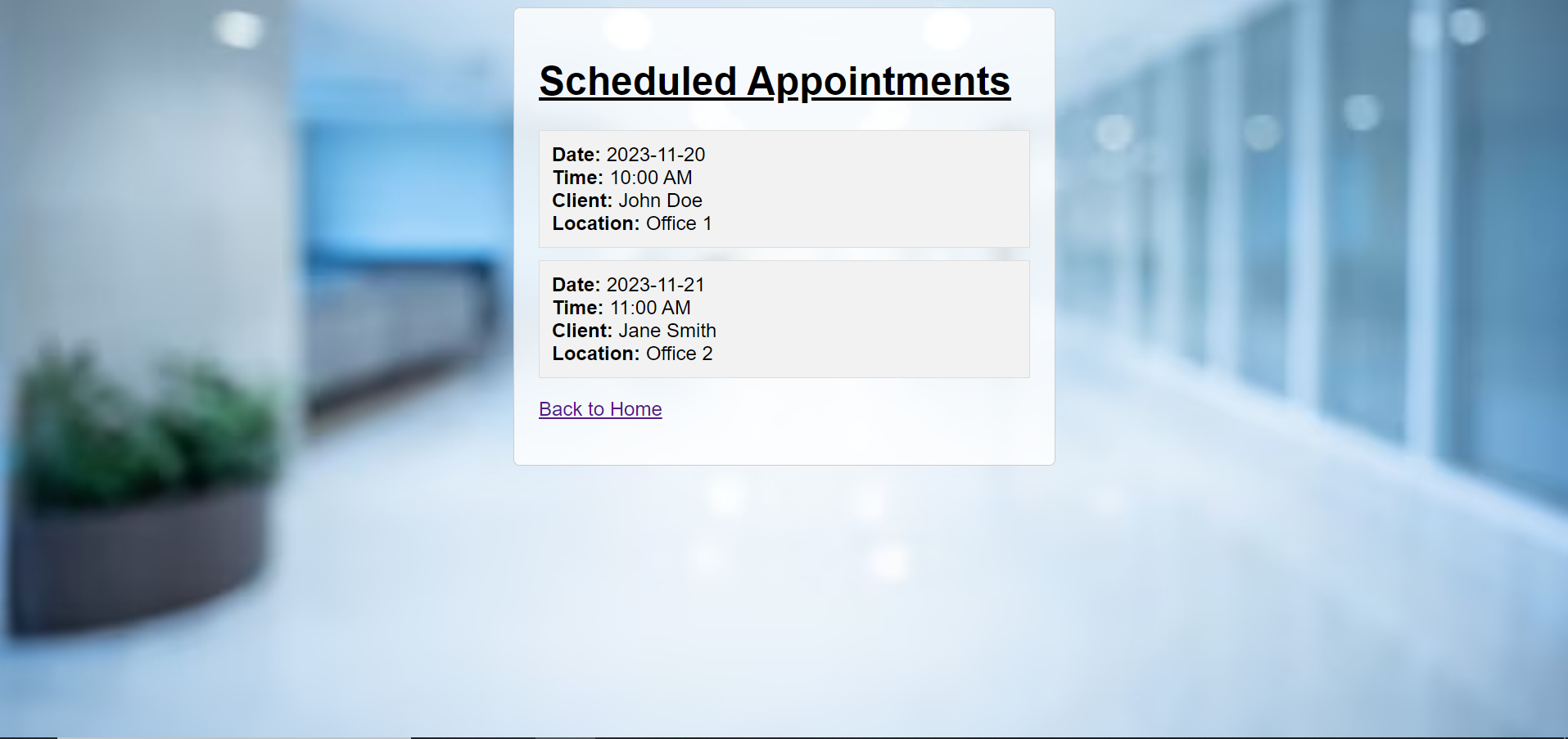
Pegassus Booking is ADA compliant by following WCAG Guidelines 2.1 AA level.

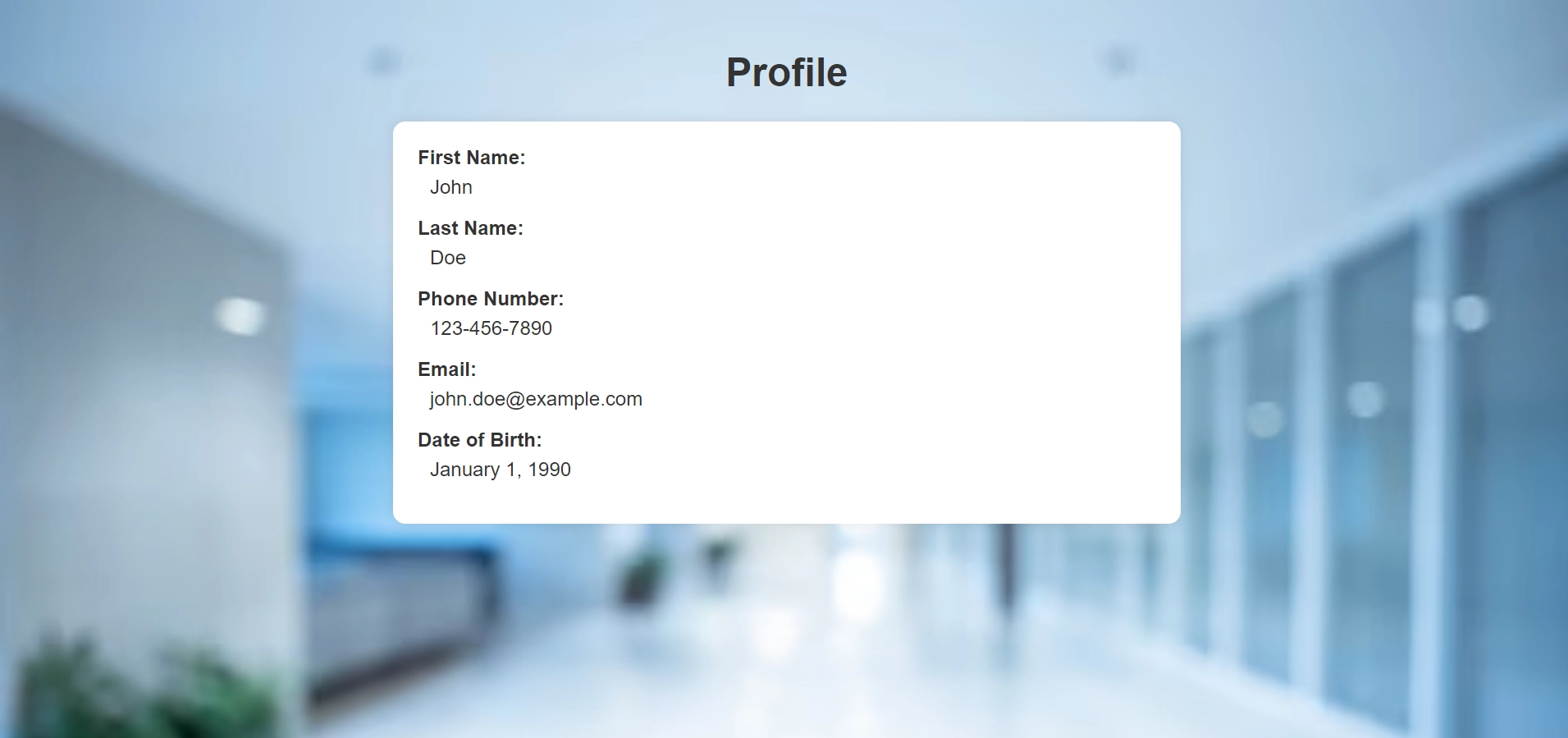
**9.2 Screen Frameworks or Images**





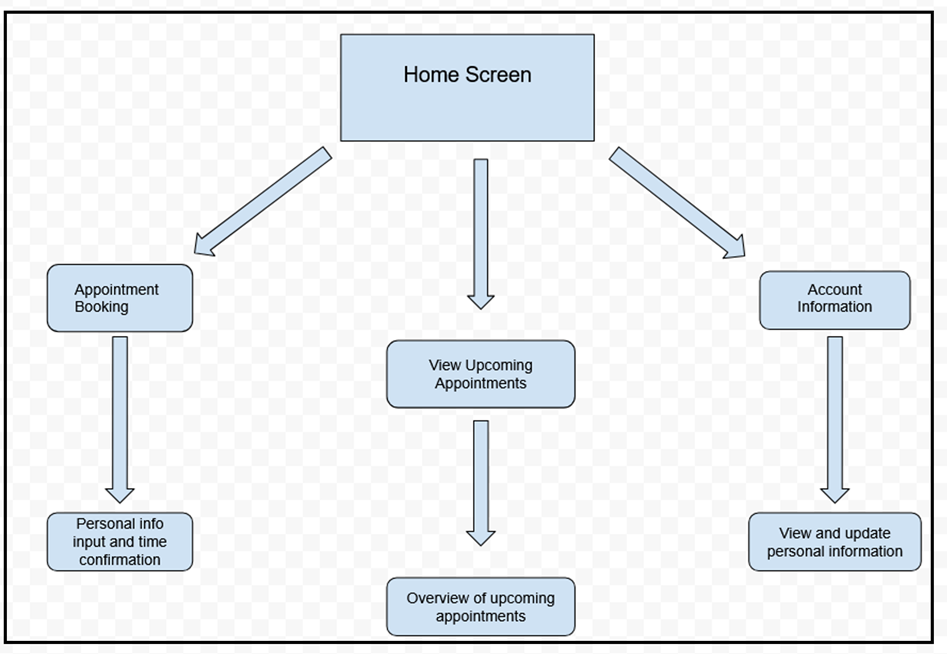






**9.3 User Interface Flow Model**

The following diagram is a rough draft UI flow chart for patients, where they have access to seven screens. Home Screen is just the home page for viewing Appointment Booking, View Upcoming Appointments, and Account Information. Appointment Booking will allow Users to schedule an appointment with account information and time scheduling options. View Upcoming will allow Users to view upcoming appointments and even cancel or reschedule appointments. Account Information will allow Users to edit personal information like Name, DOB, SSN, Insurance Information, Health Issues, etc.



**10. Requirements Validation and Verification**

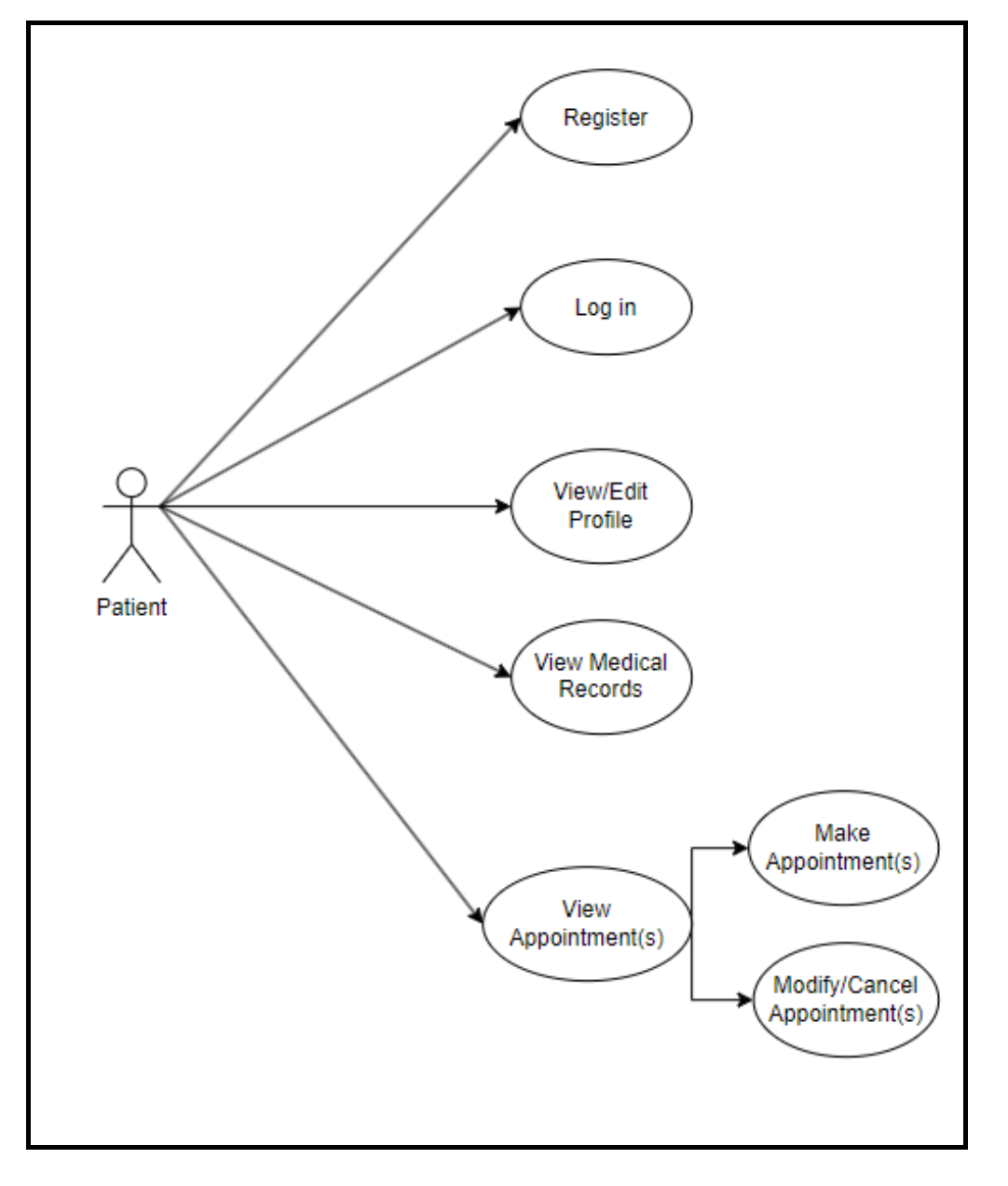
| **Requirement ID** | **Requirement Description** | **Component Modules** | **UI Elements / Low Level Components** | **Testing Method** |
| --- | --- | --- | --- | --- |
| REQ-001 | User should be able to log in | Authentication Module | Login Screen, Authentication API | Manual Testing: Enter valid credentials and verify successful login |
| REQ-002 | User should be able to register | Authentication Module, User Management | Registration Form, User Database | ManualTesting: Fill registration form with valid data and verify user creation |
| REQ-003 | Patient can view available appointments | Patient Module/ Appointment Management Module | Appointment Dashboard/ Database | Manual Integration Testing:  Create sample appointments, login as a patient, verify available appointments are displayed |
| REQ-004 | Patients can make appointments | Patient Module/ Appointment Management Module | Appointment Dashboard/ Database | Manual Integration Testing:  Log in as a Patient, Verify you can make an appointment from samples |
| REQ-005 | Patients can view their appointments | Patient Module/ Appointment Management Module | Appointment Dashboard/ Database | Manual Testing:  Login as Patient,  Make an Appointment,  Verify that Patient can view their appointment |
| REQ-006 | Patients can cancel their appointments | Patient Module/ Appointment Management Module | Appointment Dashboard/ Database | Manual Testing:  Login as Patient,  Select patients Appointment and cancel it,  Verify that appointment was canceled and that the timeslot became available again |
| REQ-007 | Patients can view their medical data | Patient Module/ Appointment Management Module | Appointment Dashboard/ Database | Manual Testing:  Login as Patient,  Select patients  check records |
| REQ-008 | Doctors can set appointment times | Doctor Module/ Appointment Time Management Module | Appointment time  Dashboard/ Database | Manual Testing:  Login as Doctor,  Select calendar  set times |
| REQ-009 | Doctors Can Modify Appointments | Doctor Module/ Appointment Management Module | Appointment Dashboard/ Database | Manual Testing:  Login as Doctor,  Select patient  Alter appointment |
| REQ-010 | Doctors can see all appointments | Doctor Module/ Appointment Management Module | Appointment Dashboard/ Database | Manual Testing:  Login as Doctor,  Select Calendar  View upcoming appointments |
| REQ-011 | Doctor inherit Patient functions | Doctor Module/ Function  Management Module | Patient functions | Manual Testing:  Login as Doctor Use patient functions |
| REQ-012 | Admin can create Admin accounts | Admin Module/  User Management  Module | Appointment Dashboard/ Database | Manual Testing: Login as Admin  Create Admin Account |
| REQ-013 | Admin can create Doctor  accounts | Admin Module/  User Management  Module | Appointment Dashboard/ Database | Manual Testing: Login as Admin  Create Doctor Account |
| REQ-014 | Admin Can change Appointments | Admin Module/  User Management  Modul | Appointment Dashboard/ Database | Manual Testing: Login as Admin  Select patient  Edit Appointments |
| REQ-014 | Admin inherit Patient functions | Admin Module/ Function  Management Module | Patient functions | Manual Testing:  Login as Admin Use patient functions |
| REQ-015 | Admin inherit Doctor functions | Admin Module/ Function  Management Module | Doctor functions | Manual Testing:  Login as Admin Use Doctor functions |

**11. Glossary**

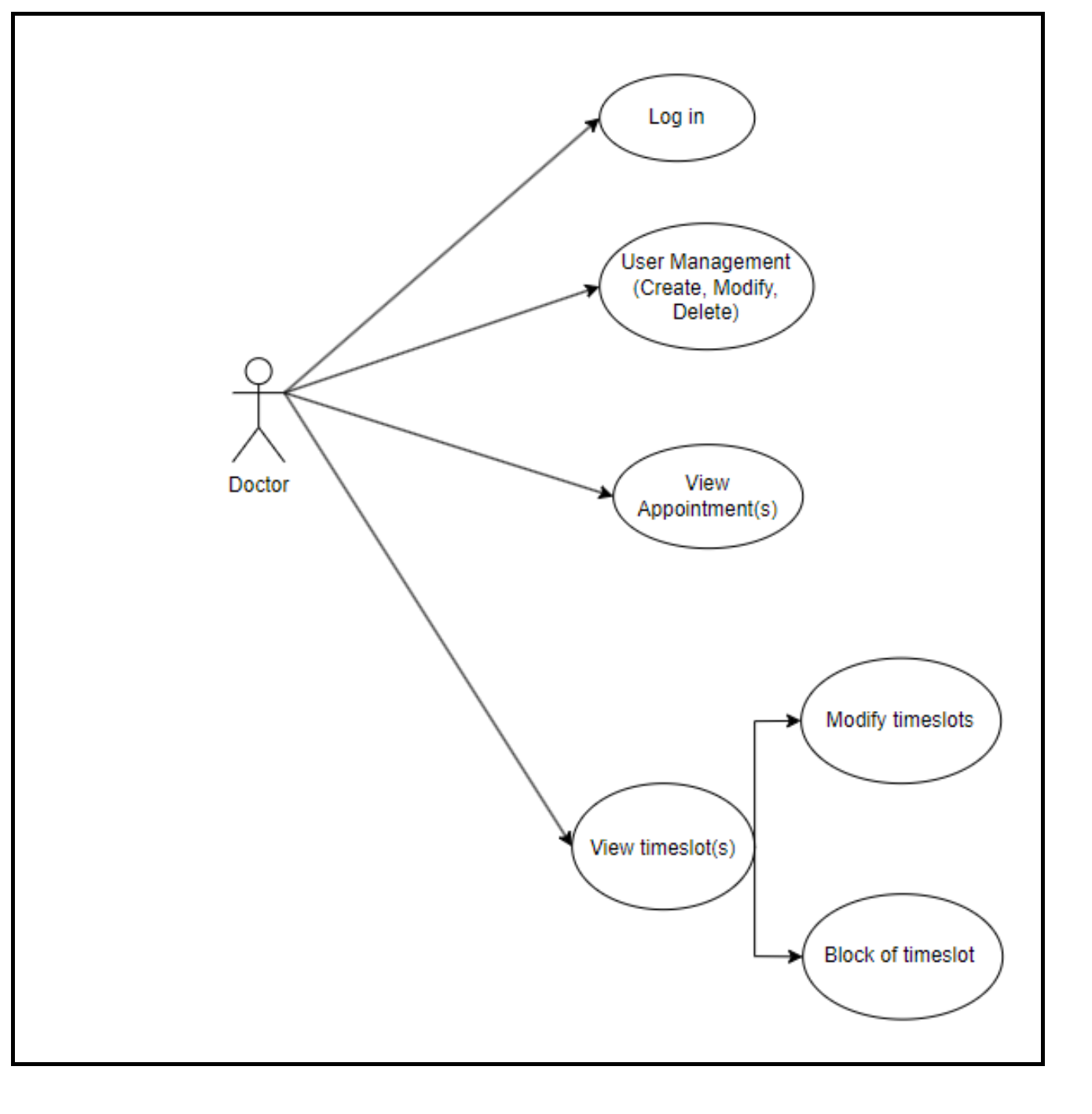
ADA - Americans with Disabilities Act

HIPAA - Health Insurance Portability and Accountability Act

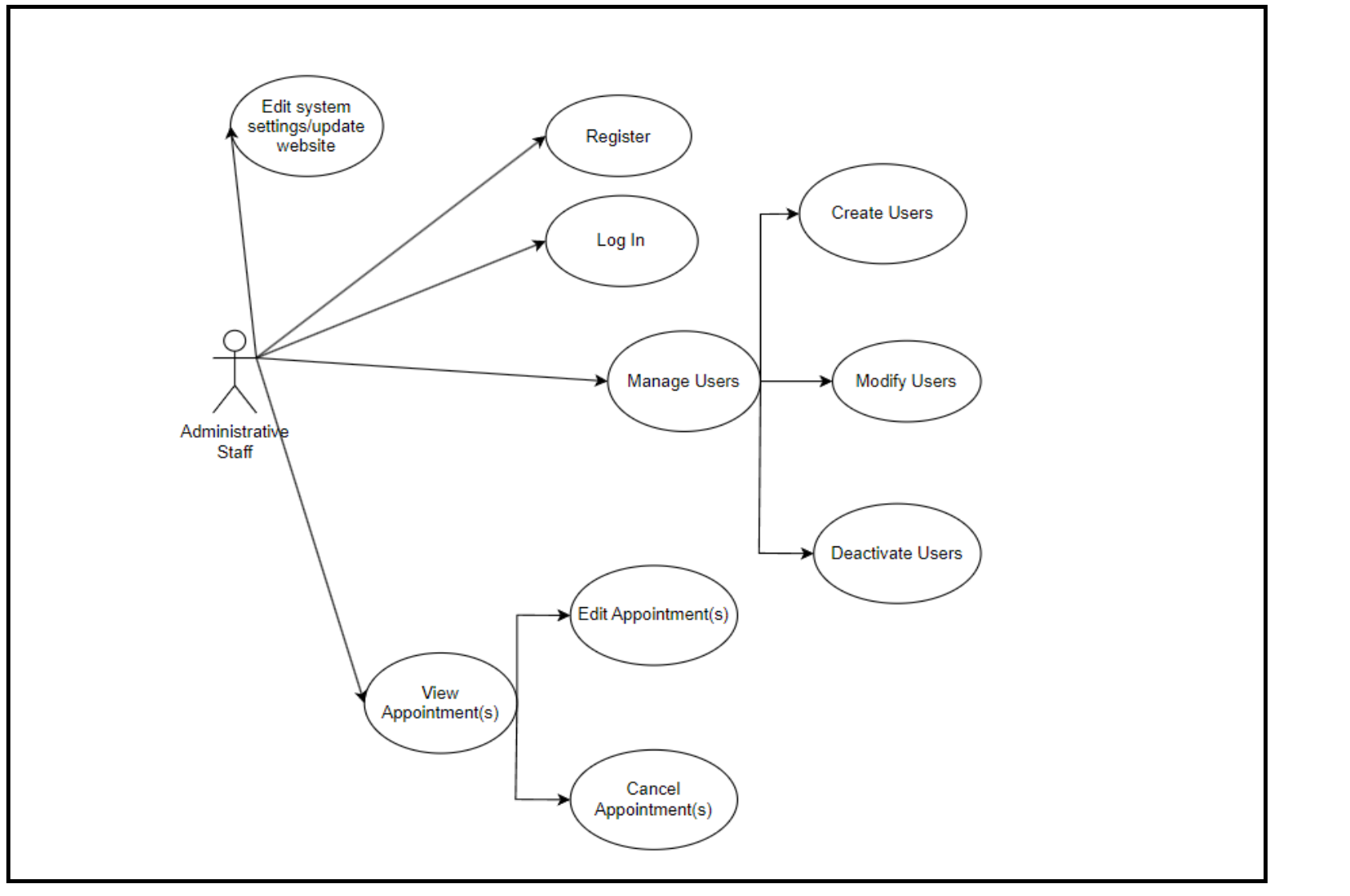
Patient Use-Case Diagram



Doctor Use-Case Diagram



Admin Use Case Diagram

****

**12. References**

Brad Appleton <brad@bradapp.net>  <http://www.bradapp.net>

<https://www.cs.purdue.edu/homes/cs307/ExampleDocs/DesignTemplate_Fall08.doc>

[The Scrum Events | Scrum Alliance](https://resources.scrumalliance.org/Article/scrum-events)

[Sprint Planning Meeting: Why It's Critical for Agile Success (scrumalliance.org)](https://resources.scrumalliance.org/Article/sprint-planning-meeting)

[What Is the Daily Scrum? | A Guide to the Daily Event | Scrum Alliance](https://resources.scrumalliance.org/Article/the-daily-scrum)

[What Is a Sprint Review? Goals and Tips for Running This Scrum Event (scrumalliance.org)](https://resources.scrumalliance.org/Article/sprint-review)

[The Sprint Retrospective - What It Is & Tips for Making the Most of Your Meeting (scrumalliance.org)](https://resources.scrumalliance.org/Article/sprint-retrospective)

[What Is Product Backlog Refinement in Scrum? | Scrum Alliance](https://resources.scrumalliance.org/Article/product-backlog-refinement)