RECOVER SMART

Your Care, Your Guide, Recover Smart By Your Side

Team Members

Nadagamuwage Sachith Akalanka Perera
Maheshi Randika Senaratne,Udawattage
Mohamed Nazeer Mohamed Ruzaik
Bhargavi Vachhani
Gampalawaduge Anton Shirantha Fonseka
Punam Dahal
Dona, Niranjala Sandamali Gunaratna Jayatilaka

MGMT-6134-(31-32)-25W

Submitted to

Mona Abou Taka
Vinnie Moraes

Date: 17/04/2025

Acknowledgements

We would like to extend our sincere gratitude to everyone who has supported us throughout the development of the *Recover Smart* project. This journey has been both challenging and rewarding, and we are deeply thankful to those who have guided, encouraged, and inspired us along the way.
First and foremost, we are incredibly grateful to our families for their continuous love, understanding, and encouragement. Your support has been our foundation, giving us the strength to stay focused and committed throughout this project.
We are especially thankful to our mentors, **Professor Mona Abou Taka** whose guidance, feedback, and encouragement have been essential in shaping our work. Your dedication and belief in our potential motivated us to strive for excellence, and your insights helped us overcome many challenges along the way.
We also want to express our appreciation to **Professor Vinnie Moraes** for his valuable input on providing feedback and guidance. Your attention to detail and thoughtful suggestions were instrumental in improving the quality of our work.
We extend our thanks to all our peers and classmates for their feedback, collaboration, and shared enthusiasm. Your contributions, ideas, and encouragement made this experience even more meaningful.
Finally, we are grateful to our institution for providing the tools, resources, and environment that allowed us to bring *Recover Smart* to life. Thank you to everyone who played a role, big or small, in making this project possible. We deeply appreciate your support and encouragement throughout this journey.

RECOVER SMART
YOUR CARE, YOUR GUIDE,
RECOVER SMART BY YOUR SIDE

Table of Contents

Abstract

Recover Smart application is a cross-platform healthcare application developed to address the challenges patients face during post-hospital recovery. The application provides personalized checklists, reminders, and real-time communication between patients and healthcare providers. It ensures the proper medication adherence, timely wound care, and recovery monitoring. This application also provides the feature which enables the patients to upload wound images and receive doctor feedback remotely. The app reduces hospital readmissions and improves recovery outcomes. Recover Smart application is developed using Flutter for mobile and web, Django for the backend, and hosted on Oracle Cloud. The application ensures a secure and scalable environment while complying with data privacy laws such as PIPEDA and PHIPA. This report outlines the system architecture, design principles, technologies, and the deployment process, presenting a comprehensive solution to support patient centered care in a digital age.

Introduction

When patients leave the hospital, their recovery journey is just beginning. But too often, that journey becomes confusing and stressful. They're expected to follow a list of instructions—take medications on time, care for wounds, attend follow-up appointments but many don't have the support they need to do all of that correctly. This can lead to complications, delayed healing, or unnecessary trips back to the hospital.

We created Recover Smart to resolve this issue. It's a simple, easy to use the application. It helps patients stay on top of their recovery with daily checklists, reminders, and are able to send updates like photos of wounds—directly to their doctor. The goal is to make recovery less overwhelming and more guided, so patients don't feel like they're doing it alone.

Doctors and healthcare teams benefit too. Instead of waiting for the next appointment to check in, they can review patient updates remotely and give feedback or make decisions right away. This saves time and allows them to focus on the patients who need immediate attention.

From a technical perspective, Recover Smart is built using Flutter, which lets us run it on both mobile and web platforms using the same code. The backend is built with Django and hosted on Oracle Cloud, with MySQL handling the database. We've also implemented secure login using JWT and made sure our setup aligns with healthcare data privacy standards.

This report walks through how we designed and built Recover Smart, how each part works, and how it all comes together to support better, smarter recovery for patients and care teams alike.

Scope of Work

The Recover Smart project focuses on designing and developing a digital health application that improves the post-hospital recovery experience for patients, doctors, and healthcare administrators. The scope includes building a user-friendly mobile application, setting up a secure backend infrastructure, and ensuring smooth communication through checklists, image uploads, and notifications.
The work is divided into the following core areas:

1. **Mobile App Development**
   The application will be developed using Flutter to ensure a consistent experience across both Android and IOS platforms. The app will include features such as personalized recovery checklists, appointment reminders, wound image uploads, and notification alerts.

2. **Backend System Implementation**
   A robust backend using Django will manage user data, recovery progress, media uploads, authentication, and communication. The backend will be hosted on Oracle Cloud using a virtual machine running Ubuntu OS.

3. **Database Setup**
   A MySQL database will be used to securely store all user-related information, including medical checklists, wound photo references, doctor feedback, and administrative controls.

4. **Authentication and Security**
   Secure user authentication will be implemented using JWT tokens. The system will also follow data protection regulations such as PIPEDA and PHIPA to safeguard patient information.

5. **Email Notification Integration**
   An SMTP email server will be configured to send alerts and reminders to users for upcoming tasks, verification, and doctor feedback.

6. **Testing, Feedback, and Iteration**
   The system will go through continuous testing and refinement. A prototype will be shared with stakeholders for feedback, and changes will be made based on their input to ensure an intuitive and efficient user experience.

Deliverables

The following deliverables will be produced as part of the Recover Smart project:

1. **Functional Mobile Application**
   A working version of the application will be developed for Smart phone. It will have complete recovery management features including checklists, notifications, and image uploads.

2. **Backend Infrastructure**
   A secure and scalable backend will be hosted on Oracle Cloud, using Django and MySQL.

3. **User Authentication System**
   JWT-based user authentication system will be implemented in application. This authenticated the user login and managed the session for patients, doctors, and administrators.

4. **SMTP Email Notification Service**
   A functioning email service will be integrated with the application to handle verification, password recovery, and update notifications.

5. **Prototype and Feedback Iterations**
   A prototype of the application will be presented to users and mentors for feedback, along with modifications based on review rounds.

6. **Testing Reports**
   Testing document will cover all functional tests, bug tracking, performance checks, and user acceptance tests.

7. **Project Documentation**
   Project documentation will include system architecture, deployment guide, user manuals, and future recommendations.

8. **Final Report and Presentation**
   Final report will cover all phases of development, design decisions, and implementation strategies, followed by a formal project presentation.

Key Stakeholders

The success of the *Recover Smart* app depends on collaboration and input from several key individuals and groups:

- **Patients (End Users)**
  Patients are an individual who are recovering from surgery or illness and who use the app to follow doctor's recommended care routines and stay connected with healthcare providers.

- **Doctors**
  Doctors are medical professionals who monitor patient progress, review uploaded images, and provide guidance through the app.

- **Healthcare Administrators**
  Administrators are personnel responsible for managing patient records, assigning doctors, and maintaining overall operational flow.

- **Development Team**
  The core project group responsible for building, testing, and deploying the system, including frontend and backend developers.

- **Mentors**
  Academic advisors and subject matter experts providing feedback, technical support, and guidance throughout the project.
  - *Mona Abou Taka*
  - *Vinnie Moraes*

Team Roles

The team comprises members with diverse roles and specializations, each contributing their expertise to the project. Their combined skills ensure a well-rounded approach to achieving project goals efficiently. The role and specialty of the team member for the project are depicted in the following table.

Table 1: Team member's roles

Team Member	**Role / Specialty**
NadagamuwageSachith Akalanka Perera	System architect, Database Specialist
Mohamed Nazeer Mohamed Ruzaik	Teach Lead Front-end & Backend Specialist
Gampalawaduge Anton Shirantha Fonseka	Software Engineer Cloud Specialist
Bhargavi Vachhani	QA Engineer R&D member
Maheshi Randika Senaratne, Udawattage	Senior QA Engineer Networking Engineer
Thanippuli Arachchige Dona Niranjala Sandamali Gunaratna Jayatilaka	Project Manager Business Analyst UI/UX Engineer
Punam Dahal	Technical Writer Documentation Research Specialist

Project Planning

Planning played a key role in the smooth and timely execution of the *Recover Smart* project. Our team approached the project with a structured development process, dividing work into phases and assigning responsibilities to ensure every task was completed efficiently and on time.

Project Timeline

We began by identifying the key phases of the project based on the functional requirements and deliverables. These phases included research and requirement gathering, system design, frontend and backend development, database integration, testing, and deployment. Each phase was assigned an approximate duration based on complexity and team availability.
The overall project was spread across several weeks, aligning with our academic deadlines. We ensured that adequate buffer time was included for feedback, iterations, and troubleshooting.
To manage our progress effectively, we adopted Agile-inspired practices by holding regular team check-ins, adapting the timeline as needed, and delivering work in small, reviewable segments.

Milestones Summary

To keep the project organized and on track, we divided tasks among team members based on individual strengths and areas of interest. The work was structured into weekly sprints using Trello, with tasks clearly labeled, prioritized, and assigned.

Table 2: Milestone Summary

Event	Description	Start date	Due date	Status
Milestone 1	Commencement	2025-01-06	2025-01-31	Completed
Milestone 2	Evaluation	2025-02-01	2025-02-28	Completed
Milestone 3	Design	2025-02-22	2025-03-19	Completed
Milestone 4	Development	2025-03-20	2025-04-09	Completed

Each milestone served as a checkpoint to assess our progress and ensure we were meeting expectations. Mentor feedback and user testing were integrated into this process to validate functionality and make improvements where necessary.

Gantt Chart

The Gantt chart below visually represents our project schedule, showing each major task along with its start and end dates. It outlines dependencies between tasks, parallel development efforts, and critical milestones. This helped us track progress at a glance and stay aligned with our timeline.

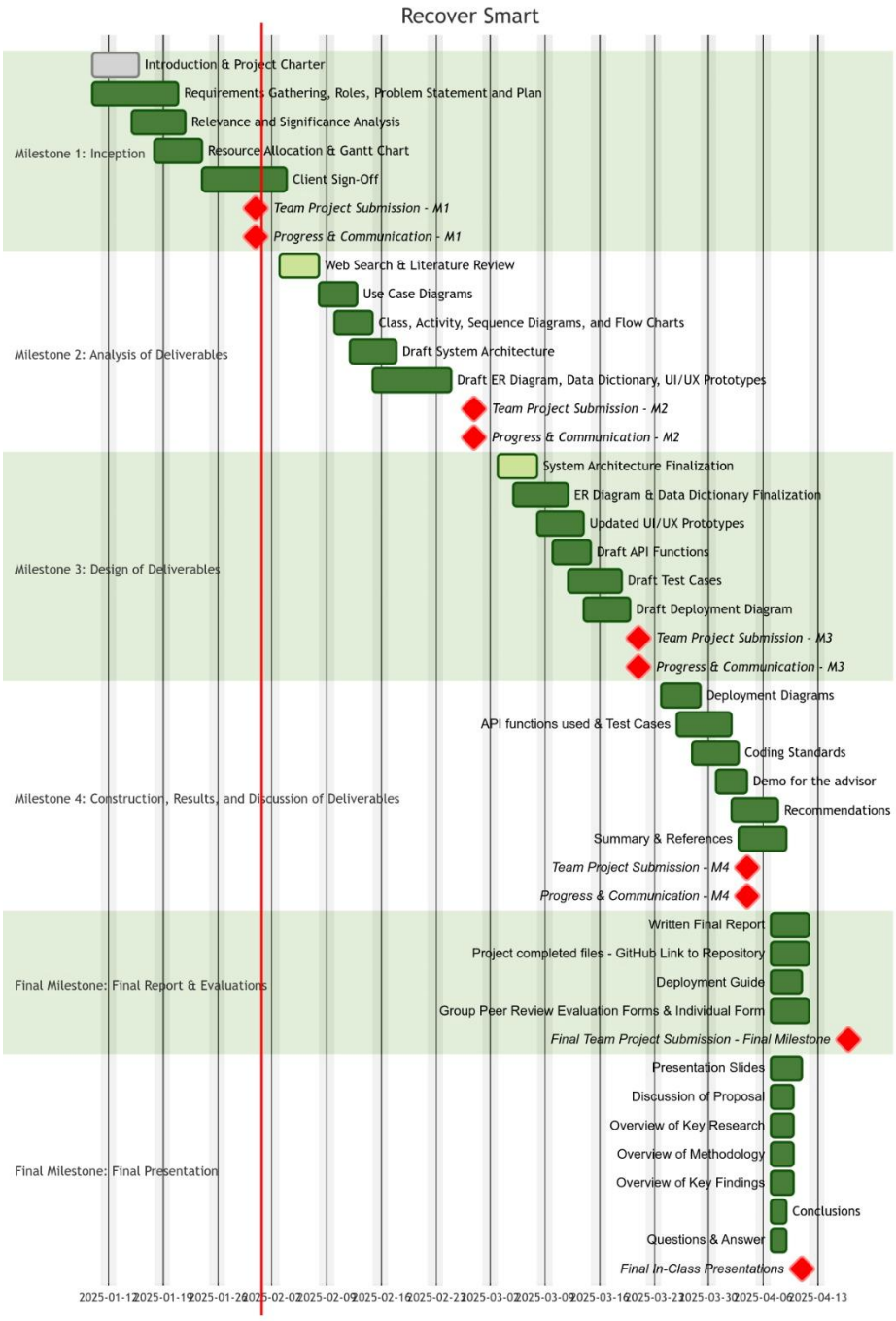


Figure 1: Gant Chart

Problem Statement

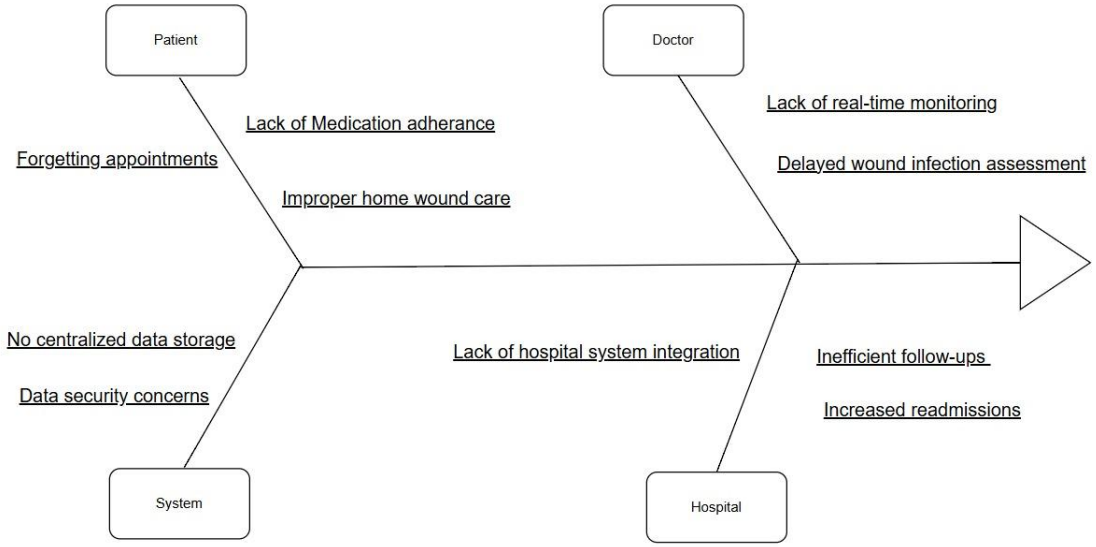


Figure 2: Fishbone architecture for the problem statement

Recovering after surgery is a critical phase in a patient's healthcare journey, yet it often presents significant challenges when the patient is discharged and expected to manage their own care at home. Many patients struggle with maintaining a structured recovery routine—common issues include forgetting medications, skipping doctor-recommended exercises, and mishandling wound care. These lapses in post-operative self-care can lead to complications such as infections, delayed healing, and avoidable hospital readmissions.

From the hospital and system side, several obstacles hinder effective post-discharge care. Doctors and nurses often lack the tools to monitor patients in real time, leading to delays in identifying wound infections or deterioration in recovery status. Additionally, hospitals may suffer from inefficient follow-up systems and a lack of centralized, interoperable data platforms to coordinate care beyond the facility. Concerns around data security further compound these challenges.

The root causes of these issues, as represented in the fishbone (Ishikawa) diagram, are multifaceted:

- **Patient-side challenges**: poor medication adherence, missed appointments, and inadequate wound care.
- **Doctor-side issues**: absence of real-time monitoring and delay in assessing recovery conditions.
- **Hospital-side inefficiencies**: weak follow-up mechanisms and high rates of preventable readmissions.
- **System-level gaps**: lack of integration across healthcare systems and data security concerns.

To bridge these gaps, a cross-platform mobile application named **"Recover Smart"** is proposed, developed using Flutter for both Android and iOS. The goal is to enhance patient engagement and

allow healthcare providers to extend their oversight beyond the hospital walls. The app will serve as a smart recovery assistant, offering:

- Personalized **reminders** for medication intake, wound dressing, and physical therapy exercises.
- A secure platform for patients to **upload wound images** and other recovery data.
- A dashboard for **doctors to monitor progress**, provide feedback, and make early interventions when needed.
- Seamless **communication between patients and care teams**, minimizing miscommunication and maximizing recovery efficiency.

By integrating these features, Recover Smart addresses the identified problems holistically. It empowers patients to take control of their recovery with confidence while equipping healthcare providers with the necessary tools for proactive care. Ultimately, this reduces hospital readmissions, improves recovery outcomes, and creates a connected ecosystem that benefits all stakeholders involved in the post-discharge recovery process.

Requirement Gathering

Functional Requirement

The functional requirements of the Recover Smart app define how the system should work to meet patients', doctors', and administrators' needs.

1. **User Management**
    a. Admin will have the right to:
        i. Create, edit, and delete patient profiles.
        ii. Manage doctor-patient assignments.
        iii. Manage user access and authentication.

    b. Patients will be able to:
        i. Create an account and log in securely.
        ii. Change personal information
        iii. Reset Password

    c. Doctors will be allowed to:
        i. Create an account and log in securely.
        ii. View assigned patient profiles.

2. **Patient Care and Recovery Monitoring**
    a. Checklist Management
        i. Admin can create a personalized recovery checklist for each patient and manage it.
        ii. The patient can mark when a task is completed.
        iii. The system automatically sends reminders of pending tasks.

    b. Medication Management
        i. Doctors can write medication requirements for a patient.
        ii. Admin uploads the medication requirement in the application in the patient's profile.
        iii. Reminders for taking medicines are sent to the patient.
        iv. In case a patient misses any dose, the system notifies the doctor.

3. **Wound Monitoring and Remote Consultation**
    a. Image Upload and Review
        i. The patient will be able to upload images of the wound.
        ii. The image is received and reviewed by a doctor via the application.

iii. If necessary, a doctor may request a better image or schedule a physical visit.

b. Doctor Recommendations
   i. Doctors can provide feedback on wound healing.
   ii. The patients need to be notified of the various recommendations made by doctors.

4. **Notifications and Alerts**
   The system should support push notifications regarding:
   - Reminders for medications
   - Confirm and remind appointments (confirmations)
   - Checklist for recoveries
   - Updates and doctor recommendations

5. **Compliance and Security**
   a. Security of Data
      i. Data of users at rest or during transit is always encrypted.
      ii. Role-based Access: Admin, Patients, and Doctors
      iii. Any login should request two-factor authentication (2FA)

   b. Privacy regulations
      i. Patient data management should follow PIPEDA & PHIPA compliance.
      ii. Deletion or Correction of data: Patients can request data deletion or corrections.

6. **Reports and Analytics**
   a. Admins
      i. Patient recovery progress and engagement tracked
      ii. Reports regarding missed medications, incomplete tasks, & patient compliance
   b. Doctors
      i. Trends in patient recoveries can be viewed
      ii. The adherence of the patients to the given medical guidance by the doctor.

Non-Functional Requirement

1. **Performance and Scalability**
   - **Performance:**

The application must offer smooth and fast performance to ensure user satisfaction. Critical user actions—such as viewing checklists, uploading wound images, and accessing doctor recommendations—should complete within 2 seconds on a standard 4G connection.

- **Scalability:**
  The system should be capable of scaling horizontally to accommodate future growth. It must handle a high volume of concurrent users (e.g., up to 10,000 users during peak hours) without affecting responsiveness. Cloud-based auto-scaling mechanisms may be employed to support sudden increases in usage.

2. **Security**
   - **Data Protection:**
     All sensitive patient data, such as health records, images, and medication details must be encrypted both at rest (AES-256) and during transmission (TLS 1.3).

   - **Authentication and Authorization:**
     Role-based access control (RBAC) will ensure that only authorized users can access specific data and features. Two-factor authentication (2FA) is required for all roles (admin, doctor, patient).

   - **Regulatory Compliance:**
     The system must comply with PIPEDA (Personal Information Protection and Electronic Documents Act) and PHIPA (Personal Health Information Protection Act) to ensure proper handling of personal health information.

3. **Usability**
   - **User-Centric Design:**
     The interface should be designed for ease of use by patients of varying technical ability, including elderly users.

   - **Accessibility:**
     The app must meet WCAG 2.1 Level AA standards and AODA requirements to support users with disabilities.

   - **Guidance and Help:**
     Built-in tutorials, FAQs, tooltips, and error messages should guide users throughout the application to reduce dependency on external support.

4. **Reliability and Availability**
   - **System Uptime:**
     The app should maintain an uptime of 99.9%, with minimal disruptions except during scheduled maintenance windows.
   - **Fault Tolerance:**
     Real-time monitoring, server failover mechanisms, and regular data backups must be in place to ensure continuous availability and quick recovery from system failures.

5. **Maintainability and Extensibility**
   - Clean Architecture:
     The app should follow modular design principles (e.g., MVC architecture) to support long-term maintenance and future enhancements.

   - **API-Driven:**
     Backend services should use RESTful APIs for integration and modularity.

   - **Documentation:**
     All code should be well documented, and version controlled. Developer guides and system design documents must be maintained to assist in future updates or handovers.

Assumptions, Constraints, and Acceptance Criteria

1. **Assumptions**
   - Patients and doctors will own smartphones with internet access and have basic digital literacy.
   - Admin users from healthcare institutions will be available to manage user accounts and monitor activity.
   - The necessary medical guidance and patient recovery plans will be provided by healthcare professionals.
   - All stakeholders understand the importance of digital recovery tools and are motivated to use the app regularly.

2. **Constraints**
   - Timeline Constraint:
     The app must be developed and delivered within a 10-week academic project period. Future enhancements or full deployment may occur post-academic phase.
   - Budget Constraint:
     Initial development must use low-cost or open-source tools and services due to academic funding limitations.
   - Infrastructure Limitation:
     Cloud services will initially be hosted under a free or basic plan with limited server capacity.
   - Regulatory Boundaries:
     All data storage and processing must remain within Canadian jurisdiction to comply with PHIPA.

3. **Acceptance Criteria**

3.1. User Experience

Criterion	Description	Acceptance Standard	Verification Method

Navigation	Users can easily access core features.	Clear and logical menu structure.	User testing and feedback
Accessibility	Support for users with disabilities.	Meets WCAG 2.1 Level AA and AODA standards.	Accessibility audits and testing tools.
Design Consistency	Visual consistency throughout the app.	Uniform color palette, icons, and layout across pages.	Visual inspection and design review.

3.2.Performance and Speed

Criterion	Description	Acceptance Standard	Verification Method
Load Time	Speed of opening app pages and data.	Key screens load within 2 seconds.	Performance testing tools (e.g., GTmetrix)
Responsiveness	Proper scaling across devices.	Optimized UI for mobile, tablet, and desktop.	Responsive testing tools
Server Latency	Time taken for backend to respond.	API response under 200ms on average.	Server logs and monitoring tools

3.3.Core Features and Outcomes

Criterion	Description	Acceptance Standard	Verification Method
Medication Reminders	Patients are alerted for every medication dose.	Notifications are sent at correct times.	Simulated testing and push logs
Checklist Completion	Patients can mark tasks as done.	Checklist status updates are accurately reflected.	User testing and backend logs
Image Upload & Review	Doctors can review patient-uploaded wound images.	Images successfully uploaded and reviewed with feedback sent.	Functional testing and UI validation

Reporting	Admins and doctors can view recovery statistics.	Reports generated match user activity and compliance data.	Report data verification
Data Security	All sensitive operations are protected.	Encryption and 2FA in place; access is role-based.	Penetration testing and access audits

Web Search

Our team has explored various technologies and methodologies to develop the Recover Smart application after conducting a comprehensive web search. Our primary focus is to ensure that the app remains reliable, efficient, and easy to use while maintaining cost-effectiveness. The chosen solutions will enhance the app's functionality, security, and overall user experience, making it a valuable tool for patients, doctors, and healthcare providers.

Application Development Technology and Framework

Recover Smart App is a multimodal tool that has built all the aspects: front end and back end, database, version control, and, of course, tools for UI/UX design. For Front end, we look at Flutter (Dart) and React (JavaScript) for a good trade-off between speed and usability (Uzayr, 2022). The server side is smoothly run and scales well using the combination of Python with Django for the back end. Data is being efficiently stored by Oracle and MySQL. In addition, we also decide to use Git and GitHub for tracking changes and collaborating with our group. Finally, Figma is selected as our choice for UI/UX design it helps us develop a neat and user-friendly interface (Bouras, 2024).

Front-end

Flutter (Dart): Flutter enables us to create apps that work on both iOS and Android using a single codebase. We chose it mainly because of its diverse functionality, which speeds up development, as well as its ability to create fluid animations and modern user interface components.

React (JavaScript): React is one of the great tools for creating responsive and dynamic pages. We can easily reuse UI components due to their component-based architecture, which improves productivity in development. React was used for this project's web dashboard, guaranteeing hospital administrators and employees smooth navigation and speedy user experience.

Back end

Python (Django): Python is known for its scalability and security and Django is a high-level Python framework. We chose it because it has built-in capabilities that speed up development, like database administration and authentication. Django oversees processing API calls, controlling data flow, and guaranteeing database and app security in our application.

Database

Oracle Database: Oracle is a strong and dependable database system, perfect for managing large amounts of data. Its high level of security is crucial for handling sensitive patient info. We use it mainly to store medical records, treatment history, and other important data related to patients.

MySQL: MySQL is an open-source database that's easy to use and works well with Django. It is efficient and reliable for handling requests. We use MySQL for storing user login details, checklist information, and communication records between patients and doctors.

Security Consideration

Secure Data Management

Keeping patient information private and secure is of top priority to the Recover Smart app. Since the app deals with confidential medical information, we are implementing multiple levels of security to protect user information from unauthorized use and potential breaches.

- **Data Encryption:** All sensitive patient information, including medical records and personal information, will be encrypted using current encryption standards before it is stored or transmitted. This ensures that even if data is intercepted, it will be illegible to unauthorized individuals.

- **Secure Authentication:** To provide login security, we will leverage Django's built-in authentication framework that provides password storage using the PBKDF2 algorithm with SHA256 hashing and salting algorithms to secure against brute-force attacks. Additionally, we will integrate multi-factor authentication (MFA) for an additional security layer.

- **Error Handling and Protection:** Proper error handling mechanisms will be in place so that error messages do not reveal any system vulnerabilities or proprietary information. Instead, errors will be logged internally in a secure manner while maintaining a smooth user experience.

- **Database Security:** The app's Oracle database will be secured through the use of role-based access controls that enable only authorized persons to access specific data. Regular security audits and updates will be performed to keep the system guarded against new threats.

- **Regular Backups and Disaster Recovery:** To prevent loss of data, automated daily backups will be performed on critical patient data. The backups will be stored in encrypted cloud storage and an off-site location for duplication. When system crashes, cyberattacks, or data degradation by accident happen, disaster recovery will be initiated to restore the system with little downtime.

- **Integrity Checks and Data Validation:** To ensure the integrity and accuracy of patient records, frequent integrity checks will be performed with the assistance of in-built database validation mechanisms. Checksums, transaction logs, and consistency checks will be employed to identify unauthorized modifications or corruption through data integrity techniques.

User Access Controls

To ensure the privacy and security of user's data, the Recover Smart app will incorporate a user access control mechanism that manages permissions and restricts data access according to user roles. This will prevent sensitive patient information from falling into the wrong hands and ensure compliance with healthcare security regulations.

- **User Roles and Permission:** The application consists of different user roles with various levels of access:

    - **Patients**: Patients can view limited content of their data, like their individualized checklist, appointment, medication, exercise, and wound care reminders. They may upload photos of their wounds or progress and view notifications and reminders.

    - **Doctors:** Doctors can see patient profiles, such as medical history and progress in recovery. They can see patient-uploaded photos, edit recovery instructions, and monitor patient status remotely. Doctors can also interact with patients through the app.

    - **Healthcare Administrators:** Administrators have the highest level of access to the application. They can create patient profiles, see recovery reports, track overall recovery progress, and keep the application settings and configurations.

- **Authentication and Authorizations:** The application uses secure authentication and role-based access control (RBAC) to ensure that only the permitted users can access specific data and functionality. The users are required to log in with encrypted passwords, and once authenticated, they are provided with access based on their role i.e. patients, doctors, or healthcare administrators. The application also complies with the least privilege principle, which means each user accesses only the data and functionality necessary for their role. In addition, two-factor authentication (2FA) may be enabled to provide an additional layer of security for sensitive areas, such as patient data and medical history.

- **Access Logging and Monitoring:** Access logging and real-time monitoring are used in tracking the user's activity and detecting any possible suspicious activity. Any activity performed within the app, such as patient editing patient profiles or doctor reviews, is logged for auditing purposes so that transparency and accountability are maintained as per PIPEDA. The system continuously monitors user activity and lags as suspicious if any unusual logins are attempted like unsuccessful logins using the wrong password. This ensures that any potential security breach is identified and addressed promptly.

- **Encryption and Data Protection:** The application keeps patient data secure by encrypting all sensitive information. We use SSL/TLS protocols to secure data communication between the

server and the app. The application uses **Oracle** as the database management system, which has security features such as data encryption, access controls, and audit logging. Access to the database is tightly controlled using role-based access and multi-factor authentication.

System Updates and Scalability

Recover Smart is designed to be highly scalable and agile so that it can scale in line with changes in health provider and patient needs. As medical healthcare technology improves, the app similarly continues to be updated with greater functionality, more security, as well as improvement in the quality of experience for users.

1. **System Update**
   Recover Smart app is designed to enable system updates through a structured release management process, which ensures that the app is up to date with new features, bug fixes, and security patches. Updates are regularly pushed out, with a focus on maintaining system stability and user experience. In system updates, there is continuous testing to guarantee the new versions are accurate before release, and users are notified in advance of updates to reduce disruption. The app also follows Continuous Integration/Continuous Deployment (CI/CD) principles, which allows seamless and automated updates, while version control ensures that any issues can be quickly rolled back if necessary.

2. **Scalability**
   The application is designed in such a way that it can easily scale up as the demand for users grows and the addition of features. The ability to scale is achieved by combining elastic architecture, cloud infrastructure, and modular software architecture.

   - **Cloud Infrastructure:** The team has proposed to host the backend on a cloud infrastructure Oracle, which can be scaled up or down based on usage and traffic. This enables the application to support more users without sacrificing performance. Cloud databases and storage allow effortless scaling of patient data storage as the application expands to support more users and more medical data.

   - **Load Balancing:** Load balancing techniques are implemented in the application to distribute incoming traffic across multiple servers in such a way that no single server is burdened. This makes the app work smoothly even during high traffic or heavy usage.

   - **Microservices Architecture:** With microservices, the application is simpler to develop and scale. Each function or feature such as notifications, patient data management, and doctor-patient communication are treated as separate services. With this, each can be upgraded, maintained, and scaled without impacting the system.

Maintenance and Version Management.

For the Recover Smart project, we have decided to use GitHub as the main version control system. GitHub is a web-based platform built around Git, a distributed version control system (DVCS), that essentially manages and keeps track of the changes related to our software development (Shao, et al., 2020). It allows our team to store and organize code while enabling tracking of changes over time and collaborative work. GitHub allows various contributors to participate in this centralized workspace project without interruption, reviewing the history and reverting to previous versions whenever required. This makes the entire process of development smooth.

There are various key parameters on which we prefer GitHub over other online platforms.

- **Installed Projects Manage Effectively:** GitHub is the platform, that organizes, manages, and keeps track of project progress with an orderly mode of working via version control, document management processes, and task coordination among the team.

- **Easy Collaboration:** While working on different features of the project from a different location, team members do not miss things like pull requests, code reviews, and issue tracking which, in turn, facilitate better communication and collaboration.

- **Reliability and Backup:** Every developer's site has a complete backup of the repository, so no one is going to lose anything because if one computer crashes, everything is still there.

- **Branching and Merging:** GitHub has a very structured approach to branching. By having us work on different features under different branches, very clean and efficient development cycles may be guaranteed since no feature or bugfix will ever become part of the main codebase until it is complete.

- **Security and Access Control:** Access Control features of GitHub can help us put restrictions and management on team members' access permissions; hence not all of them can modify critical sections of the project.

GitHub Usage in Recover Smart

- **Repository Management:** We now have dedicated GitHub repositories to store our code and documentation, along with all other necessary files for the project. Therefore, we can centralize the activities of development using these repositories.

- **Version Tracking:** Each modification done to the code is tracked, resulting in the possibility of reverting to a previous version, scrutinizing changes even once they are made, and ultimately keeping the contributors accountable for their changes.

- **Collaboration Workflow:** They currently clone the repository from GitHub to then work on their local machines making changes and pushing updates back to GitHub. It also requires a pull request for review before merging any changes.
- **Issue Tracking and Task Management:** We can get the status of ongoing works, bug fixes, feature enhancements, and all other tasks very effectively managed using GitHub's issue-tracking system.

- **Code Review Process:** We employ a rigorous review system through which all modifications must pass peer review before merging, thus ensuring that the resultant code meets a high standard in addition to lower possible errors.

Application Design Principle

Recover Smart application is designed with a firm foundation that ensures cross-platform support, high performance, security, and maintainability. Since the application is developed on Flutter for Android and iOS and driven by a Django backend with Oracle as the database, the design principles are efficiency, scalability, and user experience.

Cross-Platform Development

Cross-platform development is one of the top priorities of Recover Smart app development, and we aim to provide a seamless and uniform experience on both Android and iOS platforms. Flutter was our choice as the front-end framework since it allows us to write a single codebase that can run on both platforms and still maintain very high performance as well as design levels.

- **Flow:** Flutter makes use of responsive layouts that ensure that UI elements are proportionately adjusted to different screen sizes available. An app can therefore provide a good and smooth user experience, with an appropriately balanced look and feel, across mobile and tablet devices without distortion or usability issues.

- **Breakpoints:** We use breakpoints based on screen size and orientation to allow the app to alter its content and structure in a way that it can be best viewed on any device. On smaller devices, functionalities like the navigation menu are reconfigured into a simpler and more convenient form.

- **Max and Min Values:** They are utilized to maintain the layout in good balance so that the content does not go too far on wider screens or is too small on narrower screens.

- **Fonts:** We prioritize system fonts the most so that we can load our app efficiently and keep it consistent because they are pre-loaded on devices. Custom fonts can be utilized to make the app look more attractive without impacting the performance.

- **Customization:** The big collection of ready-made widgets and third-party plugins in Flutter facilitates the customizing of the UI and functionality of the app so that it caters to Recover Smart's unique look and user needs.

User Experience (UX)

The Recover Smart app is all about the user experience i.e. patient, doctor, and healthcare provider. Interaction processes are made as intuitive, effective, and satisfying as possible so that all individuals involved may have a great time using the app.

- **Navigation**: The app has a clear and accessible navigation structure that lists menus directing the users right to the section they want without any trouble.

- **Clear Menus:** Logical and simple menus are employed, with dropdown options for subcategories. This assures easy navigation for users having varied tech-savviness.

- **Header and Footer:** A consistent header and footer across pages give easy access to critical features, which would include but not be limited to settings, notifications, and account details.

- **Accessibility:** Accessibility has been a foremost priority. Truly, it provides support for the screen reader and the high contrast mode and uses a large, legible font to help those people with visual impairment.

- **Usability:** It's all about the uncluttered interface, where very little cognitive load is placed on the users. Progress bars, intuitive buttons, and easy-to-follow instructions bring about usability across a diverse user base.

UI/UX Tools

Competitor Analysis

We distinguish Recover Smart from other healthcare apps by unique aspects of design and functions.

- **Identifying Competitors:** We evaluated the applications in the healthcare and post-hospital recovery category from different perspectives, especially for similar functionalities. This analysis would help us to identify the areas that could spur innovation.

- **Feature Comparison:** Competitor apps were evaluated for key features such as remote monitoring, personalized recovery checklists, and secure image uploads. We aim to create a smoother, more intuitive experience that meets privacy and security standards to protect sensitive patient information. (Hamidli, 2023)

Differentiation Factors:

Unique Design Features: Recover Smart integrates healthcare-specific features like doctor-patient communication and progress monitoring into a unified and easy-to-use interface.

- **User Experience:** By emphasizing real-time updates and push notifications, we maintain our patient engagement and keep them on track with their recovery.

- **Technology:** Unlike other competitors that tend to emphasize generic healthcare needs, Recover Smart uses advanced technologies to focus on secure remote monitoring, embracing measures like wound tracking and cloud-based data management in ways that differentiate it from others.

-

Project Management

Methodology

In software development, common project management methodologies are Agile, Scrum, Waterfall, and Hybrid (Kuhrmann, et al., 2017). For the Recover Smart project, we adopted a hybrid methodology that covers elements of Waterfall and Scrum to ensure a planned structure with room for improvement in iterations.

Agile Methodology:

For the development of *Recover Smart*, we chose the Agile methodology because it gives us the flexibility to adapt and improve the app continuously. Since healthcare needs can evolve, an iterative approach helps us refine features based on real-time feedback rather than sticking to a rigid, predefined plan.
To structure our Agile process, we follow the **Scrum framework**, which keeps our team aligned and focused on delivering value at every stage. Here's how we work:

- **Bi-weekly Sprints:** We develop and test features in two-week cycles, ensuring steady progress.

- **Sprint Reviews & Planning:** At the end of each sprint, we review what's been done, discuss any challenges, and plan the next steps.

- **Incremental Deliverables:** Every sprint results in something tangible—whether it's a completed module, an improved user interface, or backend enhancements.

- **Backlog Management:** We maintain a prioritized list of features, improvements, and bug fixes to keep track of what matters most.

- **Regular Check-ins:** Every other day, we have quick meetings to discuss progress, resolve any issues, and adjust priorities if needed.

This approach keeps development flexible and ensures that *Recover Smart* is always evolving to better serve patients and healthcare providers.

Project and Issue Management Tools

For effective task management, tracking, and collaboration, we use Trello as our project management software. With its board-based layout, Trello keeps our workflow organized, from task distribution to storing a complete issue-chasing backlog that can include bugs to new feature requests or ongoing enhancements. Tasks are labeled with such lists as "To Do," "In Progress," and "Done" to provide complete visibility and accountability of tasks.

Online Meeting Tools

Collaboration is the key to the success of a project. We used Teams to schedule virtual meetings, plan sprints, and discuss documentation. The video conferencing, screen-sharing, and chat functionalities of Microsoft Teams ensure seamless communication among team members, even when working remotely.

Documentation

Proper documentation is essential for maintaining project continuity, tracking development progress, and ensuring the future scalability of Recover Smart. To achieve this, we utilize a combination of tools that streamline collaboration and organization. Microsoft Office Suite, including OneDrive, Word, and Excel, is used for creating, sharing, and managing project-related documents, ensuring that all team members have access to up-to-date information. For visualizing key components of the system, such as system architecture, use case diagrams and database schemas, we rely on Draw.io, which helps in designing clear and structured diagrams.
Additionally, GitHub serves as our primary platform for maintaining code repositories, facilitating version control, and enabling seamless collaboration among developers. This structured documentation approach ensures that all stakeholders including developers, testers, and future maintainers have a good understanding of the system's design, development workflows, and deployment processes, ultimately supporting the long-term success of the project.

Use Case Diagram

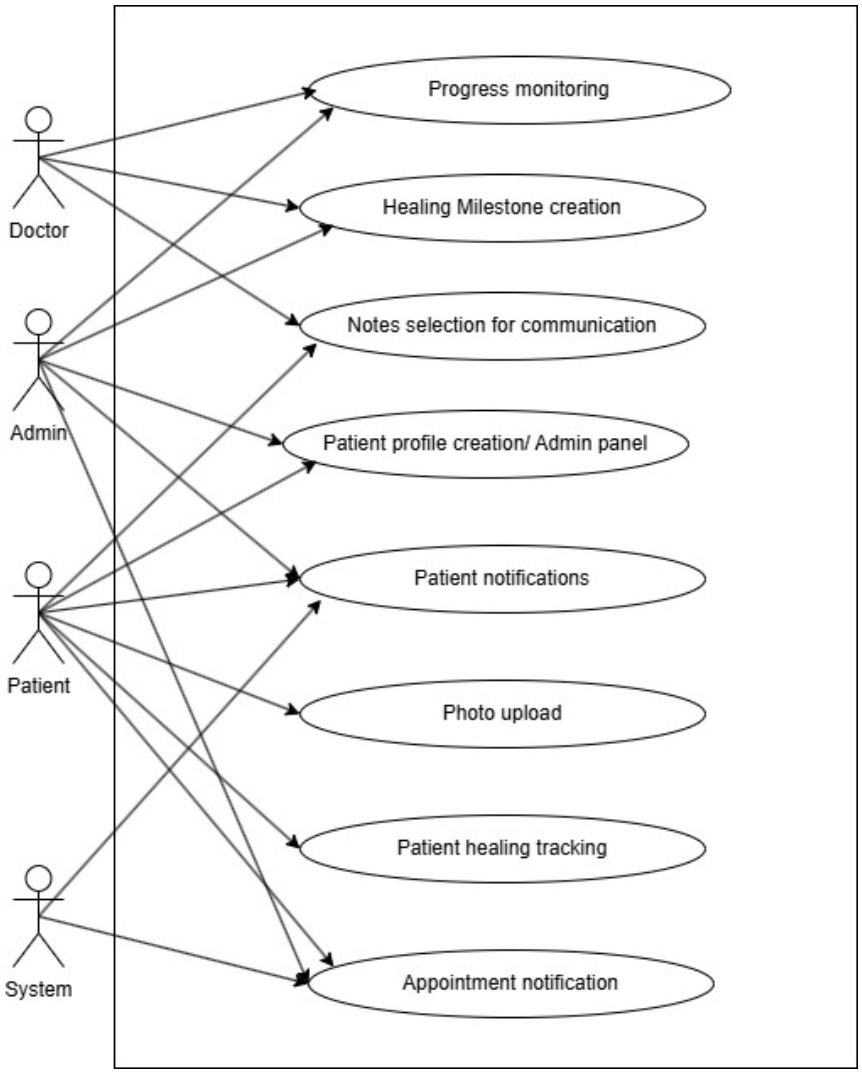


Figure 3: Use Case Diagram

This use case diagram for the **Recover Smart** app visually represents the interactions between different user roles and the system's functionalities.

- **Doctor**
  The doctor creates personalized healing milestones and monitors patient progress through the app. They communicate with patients using notes and assess uploaded photos to ensure proper recovery.

- **Admin**
  The admin manages patient profiles and supports communication by selecting standardized notes. They also help coordinate notifications and ensure the system runs smoothly behind the scenes.

- **Patient**
  The patient follows their recovery plan using the app, receiving reminders for medications, appointments, and exercises. They can upload photos and track their healing progress over time.

- **System**
  The system automatically sends appointment and recovery reminders to patients. It supports doctors and admins by handling routine background tasks and ensuring timely notifications.

Use Cases

Use Case ID	Use Case Name	Actors	Description	Preconditions	Steps	Postconditions
UC01	Patient Profile Creation / Admin Panel	Admin, Patient	Admin issues a registration link with guidelines for the patient to create a profile, allowing access to the app.	Admin must have access to the system and patient details.	1. Admin creates a profile for the patient. 2. System generates a registration link. 3. Patient receives the registration link and guidelines. 4. Patient completes the registration. 5. System validates credentials and activates the patient's account.	Patient profile is created and active, allowing access to the app.
UC02	Healing Milestones Creation	Admin, Doctor	Admin creates a personalized Healing Milestones list for a patient based on doctor's recommendations.	Patient profile must exist, and the doctor must provide recovery recommendations.	1. Doctor provides recovery guidelines for the patient. 2. Admin enters milestones into the app, specifying timelines, tasks, and medications. 3. System assigns Healing Milestones to the patient's profile.	A personalized Healing Milestones list is available for the patient to view and follow.
UC03	Patient Healing Tracking	Patient	Patients track their healing progress by checking off completed milestones.	Patient profile must exist, and milestones must be assigned by the admin.	1. Patient logs into the app. 2. Patient accesses the Healing Milestones list. 3. Patient checks off completed milestones.	Healing progress is updated, and completed milestones are logged in the system.
UC04	Photo Upload	Patient	Patients upload photos of their	Patient profile must exist, and	1. Patient selects the "Upload Photo" option.	The uploaded photo is available for

Use Case ID	Use Case Name	Actors	Description	Preconditions	Steps	Postconditions
			surgical site for monitoring purposes.	the app must support photo upload functionality.	2. Patient takes or selects a photo from their device. 3. System saves the photo under the patient's profile.	doctors to review under the patient's profile.
UC05	Patient Notifications	Admin, Patient, System	The app automatically determines notifications for reminders, such as doctor appointments, medications, or follow-ups, while admins have the ability to push additional notifications.	Admin must have access to the system, and patient notifications must be enabled.	1. The app determines necessary notifications based on patient data. 2. System sends the notification to the patient's device. 3. Admin can push additional notifications as needed. 4. Patient receives and views the notification.	Patient receives timely reminders and can view them in the app.
UC06	Progress Monitoring	Admin, Doctor	Doctors and admins monitor patient recovery by accessing dashboards containing Healing Milestones, uploaded photos, and other relevant data.	Patient profile must exist, and recovery data (milestones, photos) must be uploaded.	1. Doctor logs into the app. 2. Doctor accesses the patient's dashboard. 3. Doctor reviews progress and provides feedback if necessary.	Doctor gains insights into patient recovery and can adjust the recovery plan as needed.
UC07	Notes Section for Communication	Patient, Doctor	Instead of a chatbot, there will be a space where patients and doctors can add important notes for surgery recovery-related communication.	Both doctor and patient profiles must exist, and the notes section must be enabled.	1. Patient accesses the notes section. 2. Patient adds notes regarding their condition, questions, or progress. 3. Doctor reviews and updates notes with responses or recommendations.	Communication is logged, and the patient receives necessary guidance through the notes section.
UC08	Appointment Notifications	Admin, System	The app automatically schedules and sends reminders to patients regarding upcoming doctor appointments.	Patient profile must exist, and appointments must be scheduled in the system.	1. The app schedules a notification for an appointment. 2. System sends the notification to the patient. 3. Patient receives and views the appointment details.	Patient is reminded of the scheduled appointment and can plan accordingly.

Sequence Diagram

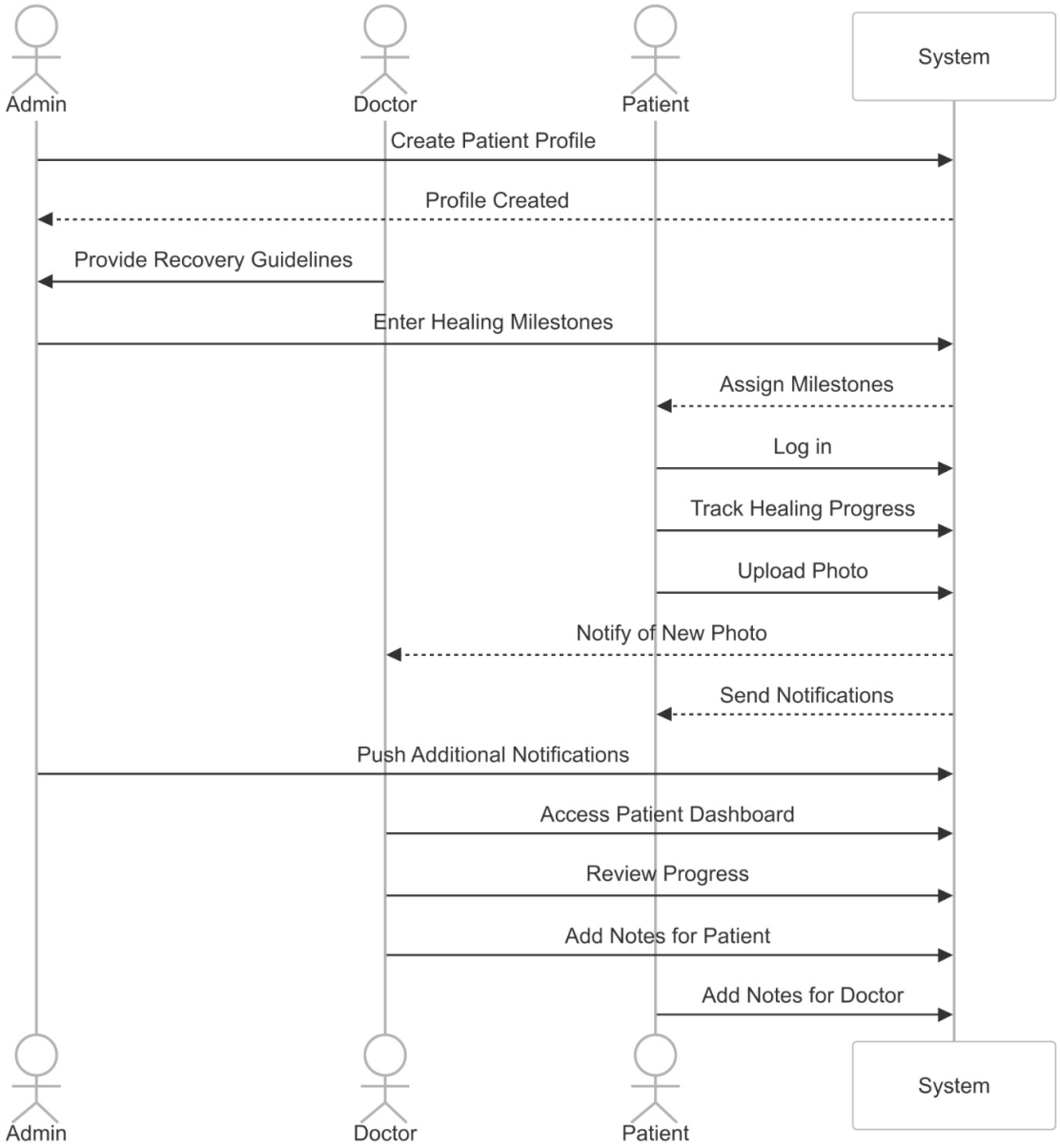


Figure 4: Sequence Diagram

- **Admin**
  The admin begins the recovery process by creating a new patient profile in the system. Once the profile is created, the admin ensures the necessary setup is complete for the doctor to provide recovery guidelines. Admins may also be involved in pushing additional notifications to the patient, helping keep communication active and recovery on track. Their role is foundational, acting as the bridge between the system and the healthcare team to ensure patient data and access are correctly configured.

- **Doctor**
  The Doctor plays a key role in customizing the patient's recovery journey. After receiving a newly created patient profile, the doctor provides personalized recovery guidelines and enters specific healing milestones tailored to the patient’s needs. The doctor is notified when a patient uploads a photo, enabling remote assessment of healing progress. Additionally, doctors access the patient dashboard to monitor progress, review submitted information and add notes for patients to guide or encourage them throughout the healing process.

- **Patient**
  The Patient is the central user of the Recover Smart app, actively engaging in their recovery. Once assigned milestones, the patient logs into the app to track their healing progress and complete recovery tasks. They can upload photos of their wounds or other healing indicators for the doctor to review. Patients also receive system-generated and custom notifications, helping them stay on schedule with appointments and tasks. To ensure communication, patients can add notes directed to their doctor, asking questions or giving updates on their condition.

- **System**
  The System acts as the automated engine behind Recover Smart, facilitating communication and task management. It responds to commands such as creating patient profiles, assigning milestones, and notifying users about key events—like new photo uploads. The system sends notifications to patients about tasks and appointments, helping them remain consistent in their recovery journey. It also supports both admin and doctor actions by handling automated responses and ensuring smooth information flow among all users.

System Architecture

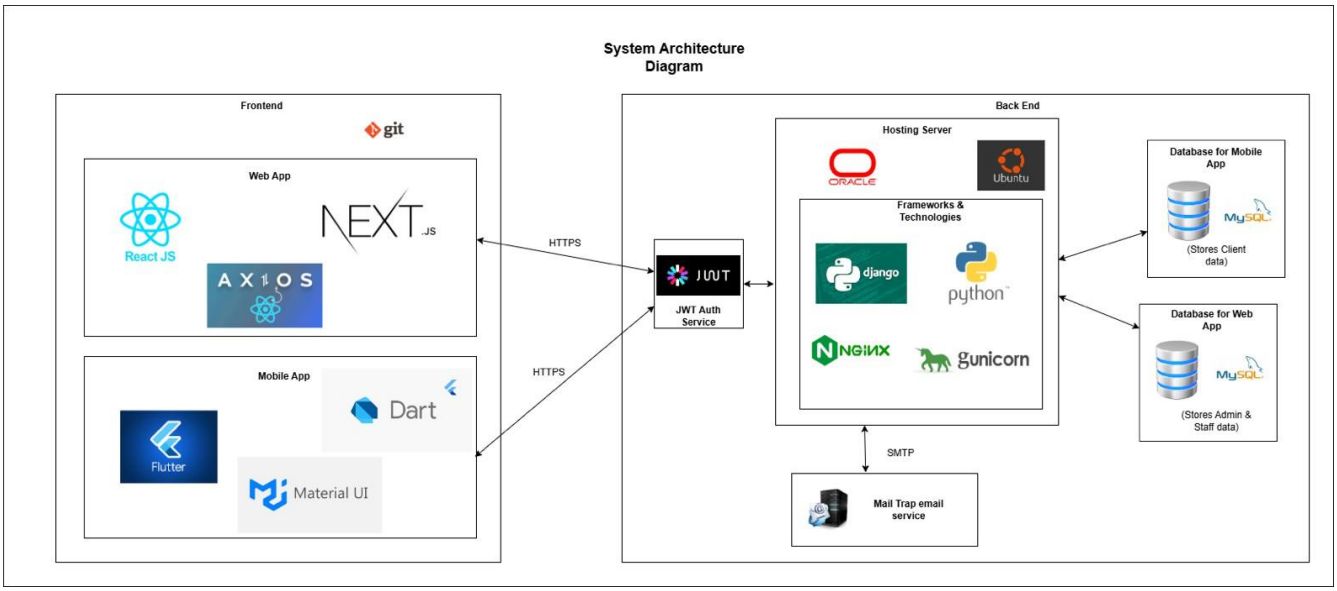


Figure 5: System Architecture

This system architecture diagram illustrates the design of the Recover Smart application as a whole and how its components communicate. The application is designed to facilitate post-hospital recovery by providing patients, doctors, and healthcare administrators with an efficient, secure, and scalable solution. Here is an explanation of each:

1. **Frontend (User Interface)**
   The frontend is the part of the application that can be viewed patients, doctors, and healthcare administrators who will interact with the system directly

   Web application:
   The web application is used by medical practitioners and healthcare administrators to track their patient improvement and manage their recovery program.

   - **React.Js:** React.js is a JavaScript library that makes building fast, interactive, and reusable UI components for web apps super easy (Reddy, 2021).

   - **Next.js:** Next.js takes React to the next level by adding features like server-side rendering (SSR) and static site generation (SSG), which help improve page load speed and SEO.

   - Axios: Axios is a handy library for handling HTTP requests. It makes it simple to fetch and send data between your web app and the backend, all while keeping things secure over HTTPS.

Doctor can quickly view patient recovery updates by logging into the web application. To make sure the doctor has the most recent information, the system uses Axios to retrieve real-time data from the backend. The doctor may authorize the patient's recovery or, if necessary, order more hospital stays based on the patient's progress. Additionally, they can send reminders, prescriptions, or updates, all of which are safely saved in the backend for access by the patient and medical administrators.

Mobile Application

The mobile application is used by patient track their progress and adhere to recovery checklists.

- Flutter: Flutter is used as it is cross-platform mobile development framework which used single codebase for developing the application in both IOS and Android.

- Dart: The programming language that Flutter uses to create apps that run smoothly and with high performance.

- Material UI: It provides pre-built user interface elements that improve the application's appearance and feel.

A patient logs into the mobile app to get a personalized recovery checklist, upload wound images for doctor review, and receive reminders for medications, appointments, and follow-ups, helping them stay on track with their recovery

2. **Backend**

The backend is responsible for processing requests, handling authentication, and storing data. The backend runs on Oracle Cloud, offering a secure and scalable hosting environment. It uses Ubuntu, a reliable Linux-based OS, to support the Django backend. This setup manages data and app logic while ensuring high performance and security.

Backend Technologies & Frameworks

The backend is built using Django (a Python framework) and runs on a NGINX + Gunicorn stack.

- **Django (Python Framework):** The backend is built on a robust Model-View-Controller (MVC) architecture, ensuring a well-structured and efficient system. It handles API requests from frontend applications, manages user authentication, validates data, and implements essential business logic to keep everything running smoothly (Ghimire, 2020).

- **Python:** It is programing language used for backend development. It is good in handling data processing, analytics.
- **NGINX (Web Server & Load Balancer):** It manages incoming HTTP/HTTPS request from web/mobile application. It ensures the system efficiency my distributing load effectively (Reese, 2008).

- **Gunicorn (Python WSGI HTTP Server):** It is high performance application server that run Django application. It can handle multiple requests concurrently.

3. **Security and Authentication**
   To protect sensitive patient and hospital data, the system has JWT Authentication system in our application.

   JWT (JSON Web Token): The JWT authentication service handles secure login and logout for users. After a successful login, it generates an encrypted token, which is sent with every request to verify the user's identity and maintain security. When a user (i.e. doctor, patient, or administrator) logs in the backend verifies their credentials and issues a JWT token. This token is then stored on the client side and used to securely authenticate API requests.

4. **Database**
   The system uses MySQL, a relational database, to store structured data efficiently. This ensures that all information is well-organized, secure, and easily accessible when needed.

   - **Database for Mobile App**: The mobile app's database is designed to store essential patient-related data. This includes user profiles, recovery checklist progress, uploaded images of wounds, and appointment details. By keeping this information in one place, the app helps patients track their recovery while allowing doctors to monitor their progress remotely.

   - **Database for Web App**: The web app's database is focused on managing doctor and administrator data. It stores information such as doctor and staff accounts, hospital management records, and reports or analytics. This allows healthcare providers to efficiently manage patient recovery and hospital operations.

   Whenever a patient updates their recovery progress, the data is securely saved in MySQL. When a doctor logs in, the system retrieves relevant patient data from the database, ensuring they always have up-to-date information to make informed decisions.

5. **Email Notification**
   The Recover Smart app keeps patients and doctors informed by utilizing an SMTP-based email service, Mail Trap, for automated communication. This system is designed to send essential emails, including appointment reminders when the deadline is near, medication schedules, and doctor's feedback on progress images. By automating these notifications, the app ensures that users stay on track with their recovery plans without missing important updates.
   The backend system generates emails based on each user's recovery schedule and sends them via the Mail Trap SMTP server. Once sent, these emails are delivered directly to the patient's or doctor's inbox, ensuring timely and efficient communication. This automated process enhances post-hospital recovery by keeping both patients and healthcare providers informed, engaged, and proactive

Entity Relationship Diagram

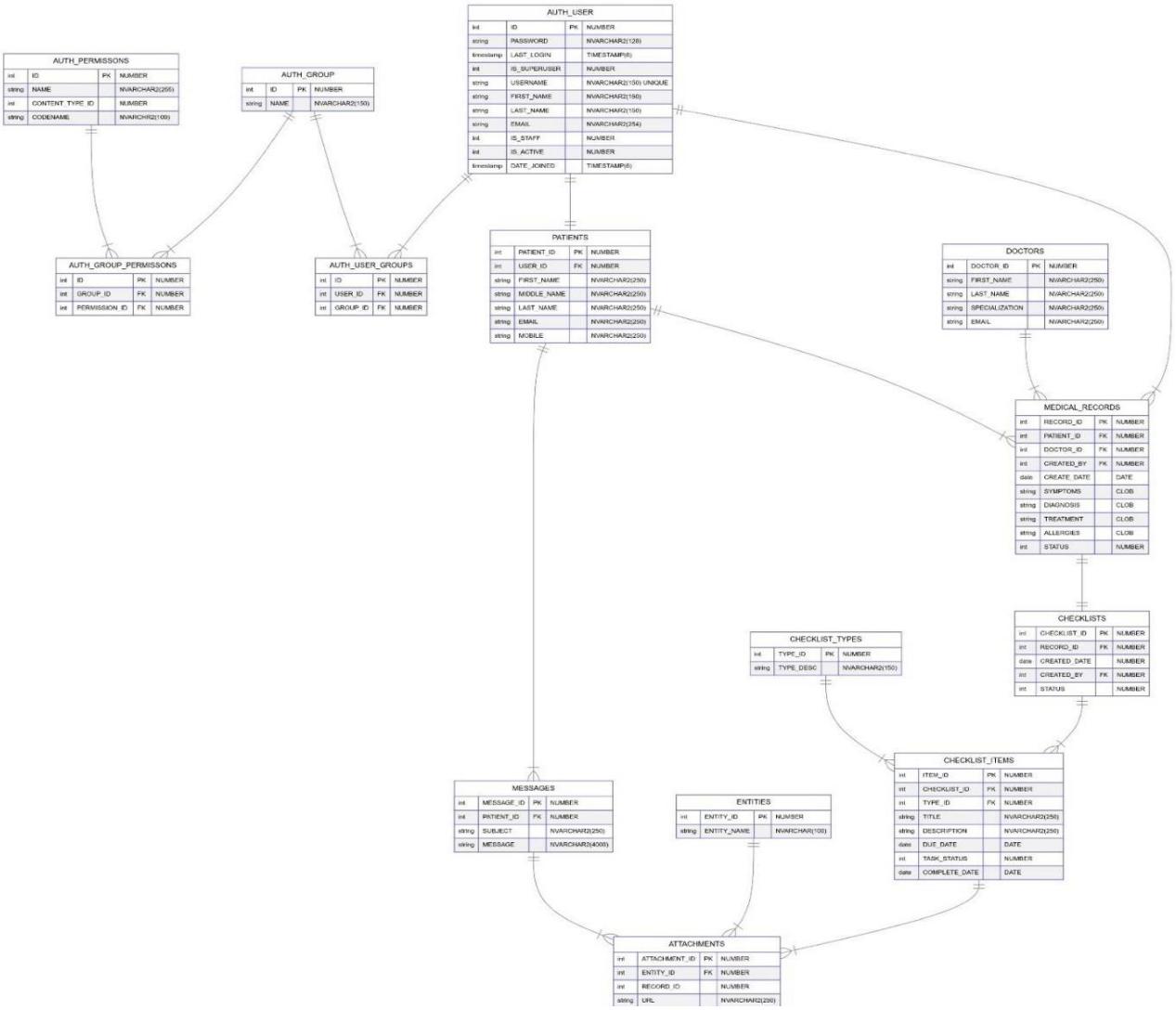


Figure 6: Entity Relationship Diagram

The Entity-Relationship Diagram (ERD) of the Recover Smart application represents how different entities interact within the system. It structures the database to manage users, doctors, patients, medical records, checklists, messaging, and attachments. This system enables seamless communication between doctors and patients, organizes patient recovery plans, and ensures secure data management.

Authentication & User Management

The AUTH_USER table stores user (i.e. Doctor, paitient, administrator) credentials, including usernames, passwords, last login timestamps, and personal details like first name, last name, and email. To manage user access, the AUTH_GROUP table defines different roles, while AUTH_GROUP_PERMISSIONS determines what each role is allowed to do. The

AUTH_USER_GROUPS table links users to specific groups, enforcing role-based access control for security and organized interactions.

Patients and Doctors

The PATIENTS table contains essential patient details such as first name, middle name, last name, and contact information like email address and mobile number. Similarly, the DOCTORS table stores doctor-specific data, including their first name, last name, specialization, and email. These two tables form the backbone of the system, where doctors oversee patient recovery, review progress, and provide medical advice.

Medical Records and Checklists

The MEDICAL_RECORDS table maintains each patient's medical history, tracking assigned doctors, symptoms, diagnosis, treatment progress, and allergies. To help patients follow structured recovery plans, the CHECKLISTS table is linked to medical records. Each checklist consists of a set of recovery tasks, further broken down into individual CHECKLIST_ITEMS, which contain a task description, due date, status, and completion date. The CHECKLIST_TYPES table categorizes checklists based on their purpose, ensuring patients follow the right recovery steps.

Messaging and Notifications

The MESSAGES table stores conversations between doctors and patients. Each message is linked to a specific patient, doctor, or medical record using ENTITY_ID. This helps doctors provide updates, feedback, and answer any questions patients may have about their treatment and recovery in real-time.

Attachments and Entities

The ATTACHMENTS table lets patients and doctors upload files like medical reports or wound images. This helps doctors track recovery progress without needing in-person visits. The ENTITIES table connects different parts of the system, making sure everything is organized. Together, these tables make it easy to store important medical documents and keep track of a patient's healing process.

Relationships

In the system, users can be both doctors and patients, and there can be more than one medical record of a patient. Each medical record contains several checklists, which store some checklist items to ensure recovery is being carried out. Patients and doctors communicate through the MESSAGES table, and the ATTACHMENTS table stores important medical documents such as wound pictures in case of remote diagnosis. This database design offers a structured and efficient way of monitoring patient recovery, communication between doctor and patient, and remote monitoring, which makes healthcare more accessible and productive.

Deployment Diagram

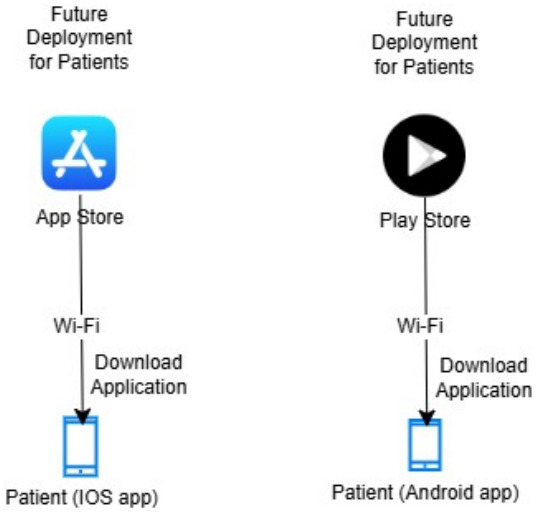


Figure 2: Application Source

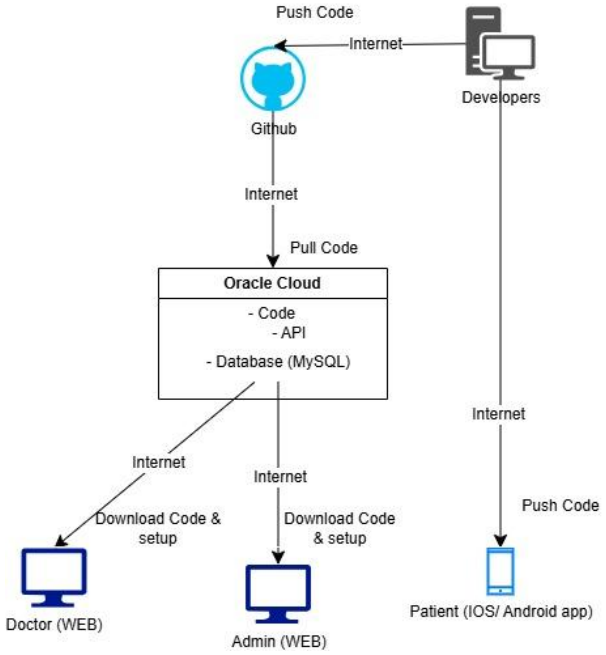


Figure 7: Basic Deployment Diagram (Developers)

The deployment process starts with developers writing and testing code for both the web and mobile applications. They push updates to GitHub, ensuring proper version control. Oracle Cloud then pulls the latest code and deploys the web application for doctors and administrators, backend APIs for both web and mobile apps, and any necessary database updates.

Doctors and administrators access the web app through their browsers, while patients download the mobile app from the App Store or Play Store in future. However, currently the application is directly pushed to the android by the developers. Both apps communicate with the backend on Oracle Cloud to send and retrieve patient data. Updates and bug fixes follow the same process, ensuring the system stays secure and up to date.

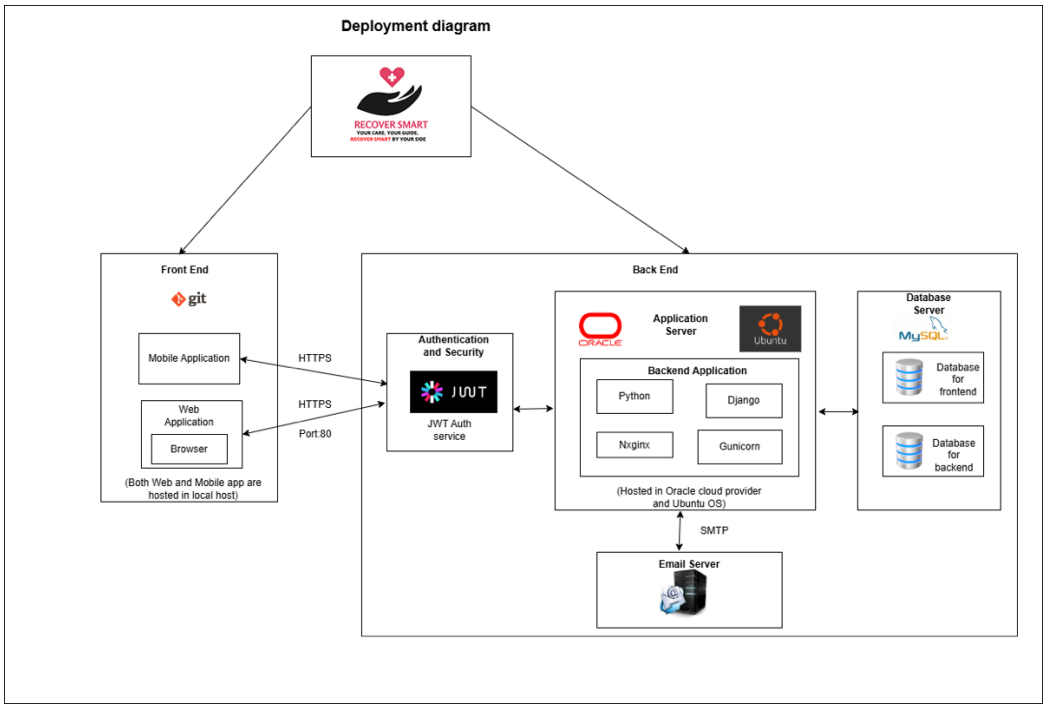


Figure 8: Detailed Deployment Diagram

This deployment diagram gives a clear picture of how the Recover Smart application will be set up, managed, and used. It shows how the code moves through the deployment process, how the app is hosted on Oracle Cloud, and how different users i.e. doctors, administrators, and patient access it. It also shows how user can download the application on their device. By mapping out the entire system, the diagram helps explain how everything connects, from backend operations to real-time user interactions. It highlights the infrastructure that keeps the app running smoothly, ensuring reliable access for those who need it most.

Elements of Deployment Diagram

1. **Developers:**
   In an application developers play a crucial role in developing and maintaining the Recover Smart application. They are responsible for writing, updating, and managing the code to ensure the app runs smoothly. Whenever changes or improvements are made, they push the latest updates to GitHub, which serves as the version control system.
   The codebase itself is structured into different components, including the frontend code for both web and mobile applications, a backend API built using Django and Python, and database scripts designed for MySQL. This organized approach helps keep the application scalable, efficient, and easy to maintain.

2. **GitHub (Version Control & Code Repository)**
   GitHub is the central hub where all the latest code is stored and managed. Developers push their updates there, making sure every change is tracked and documented. This keeps things organized and makes it easy to review updates or roll back if something goes wrong. When it's time to deploy, Oracle Cloud pulls the latest code from GitHub, ensuring the application is always running on the most up-to-date version.

3. **Oracle Cloud (Application Hosting & Database Management)**
   Oracle Cloud is the backbone of the Recover Smart application, serving as its primary hosting environment. It handles everything from running the backend services for both web and mobile apps to ensuring smooth and secure data exchange through API services. This setup allows doctors, patients, and administrators to interact with the platform seamlessly. The database, powered by MySQL, stores all critical information, including patient records, recovery checklists, doctor feedback, and uploaded images. To keep the application up to date, Oracle Cloud pulls the latest code from GitHub and deploys it either automatically or manually, depending on the update requirements. This process ensures that the system remains stable, secure, and always ready to support users in their recovery journey.

4. **Doctor (WEB) – Accessing the System**
   Doctors use the web application to monitor patient recovery, review uploaded wound images and approve or request follow-up visits. They can also provide prescriptions and care instructions, ensuring patients receive proper guidance remotely.
   Since the web application is hosted on Oracle Cloud, doctors can access it from anywhere over the internet. Whenever updates or bug fixes are released, the latest version is automatically available. If any setup is required on their end, they can easily download and install updates to ensure they're always using the most up-to-date and reliable version of the application.

5. **Admin (WEB) – Managing the System**
   Healthcare administrators use the web application to manage patient records, assign doctors, and oversee the overall recovery process. They also track recovery trends and analyze reports to improve patient care and hospital efficiency. Another key part of their role is managing security, ensuring that only authorized users have access to sensitive patient information.
   Like doctors, administrators access the application through Oracle Cloud over the internet. When updates or bug fixes are available, they can download and install them as needed to keep the system running smoothly and securely.

6. **Patient (Mobile App - iOS/Android)**
   Patients can use the Recover Smart mobile app on iOS and Android to stay on top of their recovery. The app provides personalized checklists with medication reminders, wound care steps, and other important tasks to ensure a smooth healing process. Patients can also upload wound images for doctors to review and receive direct feedback or alerts about their progress. To keep them informed, the app sends appointment reminders through push notifications or email, helping them stay engaged in their recovery.
   When updates are available, developers push the latest changes, and patients receive them through the App Store or Play Store. The app seamlessly connects to the backend, which is hosted on Oracle Cloud, ensuring real-time communication between patients, doctors, and administrators.

Deployment Guide

GitHub

GitHub serves as the central repository for version control and collaborative development of the Recover Smart application. The entire codebase for both the frontend and backend is hosted on GitHub, allowing multiple developers to contribute simultaneously using features like branching and pull requests. GitHub enables issue tracking, change history monitoring, and streamlined code reviews, making the development process transparent and manageable. It also supports integration with CI/CD pipelines for automated testing and deployment, ensuring the system remains stable and reliable.
Backend Link: https://github.com/Ruzaik11/nexus-cv-backend
Admin Panel: https://github.com/sachith-perera/recover-smart-admin
Front End: https://github.com/sachith-perera/recover-smart-app

Flutter Web App and Mobile App

The frontend of the application is developed using Flutter, which enables cross-platform development for Android, iOS, and web using a single codebase. This approach simplifies maintenance and ensures a consistent user interface and experience across all platforms. The Flutter code is compiled into platform-specific formats such as APK for Android and optimized HTML/CSS/JS for web deployment.
Key features include:

- Unified codebase for web and mobile
- Integration with RESTful APIs
- Responsive design and fast performance

The app communicates with the backend using HTTP requests and consumes RESTful APIs to manage data and user interactions. Users can either download the mobile version from an app store or access the web version directly through a browser.

Browser

Users can access the Flutter web application using modern browsers such as Google Chrome, Mozilla Firefox, and Microsoft Edge. The web version is responsive and automatically adapts to different screen sizes, making it accessible on desktops, tablets, and smartphones. The app is hosted on a cloud server and served to users through the Nginx web server, ensuring fast load times and a secure connection. This browser-based access provides flexibility and ease of use for users who prefer not to install the mobile app.

JWT Authentication

JWT (JSON Web Token) is implemented to handle user authentication and session management securely. When a user logs in, the server generates a token that encodes the user's ID and session expiration time. This token is sent to the client and included in the authorization header of subsequent requests.
Main benefits:

- Stateless authentication
- Secure token-based access control
- Lightweight and scalable mechanism

The backend verifies the token for each request, ensuring that the user is authenticated and authorized to access the requested resources.

Backend App (Django)

The backend application is built using Django, a powerful and scalable Python web framework. It handles all core functionalities such as user authentication, checklist generation, wound image management, and role-based access control. Django REST Framework (DRF) is used to create APIs that the frontend consumes. Gunicorn serves as the WSGI server that runs the Django application, while Nginx acts as a reverse proxy to handle client requests efficiently.
Highlights:

- Modular and secure architecture
- REST API support using DRF
- Efficient request handling with Nginx and Gunicorn

The backend connects to the MySQL database and uses Django ORM to interact with the data layer.

Oracle Cloud (Ubuntu OS)

The application is hosted on Oracle Cloud Infrastructure using a virtual machine that runs Ubuntu OS. This environment provides a secure and customizable Linux-based server for deploying the backend, frontend, and database services. The virtual machine is configured with necessary software packages, firewalls, and security rules.
Important aspects include:

- Use of SSH keys for secure access
- Deployment of Gunicorn, Nginx, and MySQL
- Scalability through Oracle Cloud infrastructure

MySQL Database

MySQL serves as the primary relational database for the Recover Smart application. It stores structured data such as user information, patient recovery checklists, uploaded image metadata,

and admin records. The database schema is designed with normalization in mind to ensure data integrity and minimize redundancy.
Key features:

- Secure database access and credentials
- Django ORM for seamless data operations
- Backup strategies to prevent data loss

SMTP Server

An SMTP server is configured to facilitate email communication within the application. This includes sending account verification emails, password reset links, appointment reminders, and administrative notifications. The SMTP server is integrated into the Django backend using its email backend configuration.
Configuration highlights:

- Gmail SMTP with TLS encryption
- Settings defined in Django settings file
- Ensures reliable email delivery and improved communication

Class Diagram

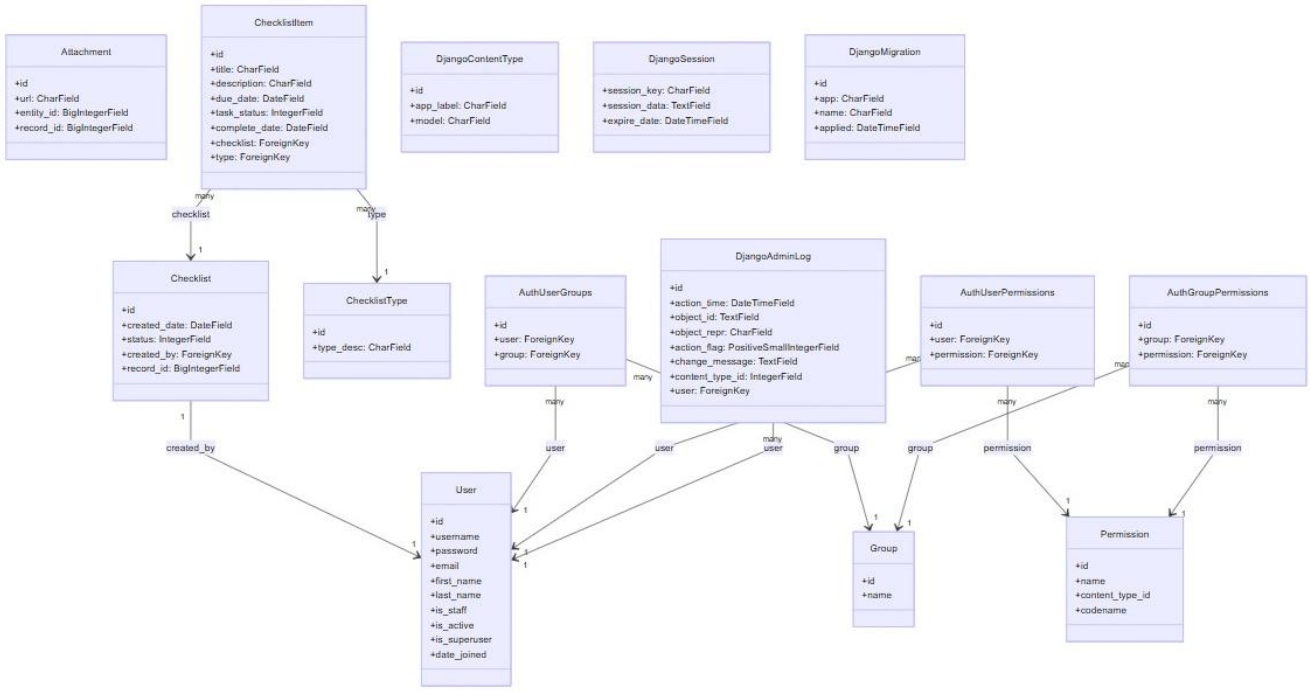


Figure 9: Class Diagram

User & Auth Models

These models handle user identity, roles, and access permissions within the Recover Smart application, which supports multiple user types like doctors, healthcare staff, and administrators.

1. **User**
   This is the main model storing user details including username, password, email, first_name, last_name, and status flags like is_staff, is_active, and is_superuser. These flags help differentiate between regular users, staff members, and administrators. It's connected to many other models such as:
   - **Checklist** via the created_by foreign key to indicate who created the checklist.
   - **DjangoAdminLog** to track changes made by the user.
   - **AuthUserGroups** and **AuthUserPermissions** for managing group and direct permission associations.

2. **AuthUserGroups**
   This model links users to groups, allowing many-to-many relationships. A user can belong to multiple groups, and each group can hold specific permissions, helping to streamline role management.

3. **AuthUserPermissions**
   Provides a way to assign specific permissions directly to a user. This allows granular control over user access when group-level permissions aren't enough.

4. **Group**
   Represents a collection of permissions bundled together, often based on user roles such as "Doctor", "Admin", or "Nurse". Groups simplify permission management by assigning them once to many users.

5. **AuthGroupPermissions**
   Associates permissions with groups, creating a many-to-many relationship. This enables a group to have multiple permissions and vice versa.

6. **Permission**
   Represents individual access controls (e.g., can_add_checklist, can_view_patient). Each permission is linked to a codename and a content type via content_type_id, which maps it to a specific model.

Checklist Management Models

These are at the heart of the app's patient recovery tracking feature, helping doctors assign tasks and track recovery progress.

1. **Checklist**
   Acts as the primary recovery plan for a patient. Each checklist is tied to a user (creator) and a record_id, likely referencing a patient or case. It includes metadata like created_date and status, indicating the phase of recovery (active, completed, etc.).

2. **ChecklistItem**
   Represents an individual recovery task, such as taking medication or performing an exercise. It contains fields like title, description, complete_date, and status. Each item is linked to:
   - A Checklist (parent plan)
   - A ChecklistType (category or type of the task)

3. **ChecklistType**
   Categorizes checklist items to improve clarity and user experience. Common types might include "Wound Care", "Physical Therapy", "Medication", etc. This classification helps filter tasks in the UI and analytics.

Attachment

1. **Attachment**
   Handles uploaded files like wound images or documents. Fields like url, entity_id, and record_id ensure that media is linked to the correct patient or checklist. This is crucial for doctors reviewing visual progress or documentation.

Django System Models

These models are part of Django's built-in framework, helping with backend operations like logging, migrations, and session tracking. (Forcier, Bissex, & Chun, 2008)

1. **Django AdminLog**
   Logs administrative actions in the system, such as record creation, updates, or deletions. It's linked to a user who performed the action and contains fields like action_time, object_repr, and change_message.

2. **Django ContentType**
   Helps Django's permission system understand what model a permission is associated with. It's a core part of Django's dynamic content-type framework.

3. **Django Session**
   Manages session data for users who are logged in. Stores things like login timestamps, session expiry, and user-specific data in a serialized format.

4. **Django Migration**
   Keeps track of which database migrations have been applied. This ensures version control and prevents repeated application of the same schema changes.

Relationships Summary

- **User** is the hub of most relationships—tied to checklists, logs, groups, and permissions.
- **Checklist** and **ChecklistItem** form the backbone of recovery tracking, helping patients follow step-by-step recovery plans.
- **Attachment** enables the app to support media uploads for visual feedback or documentation.
- The **permissions system** is robust, combining direct user permissions and group-based access control to manage different user roles efficiently.

User Journey

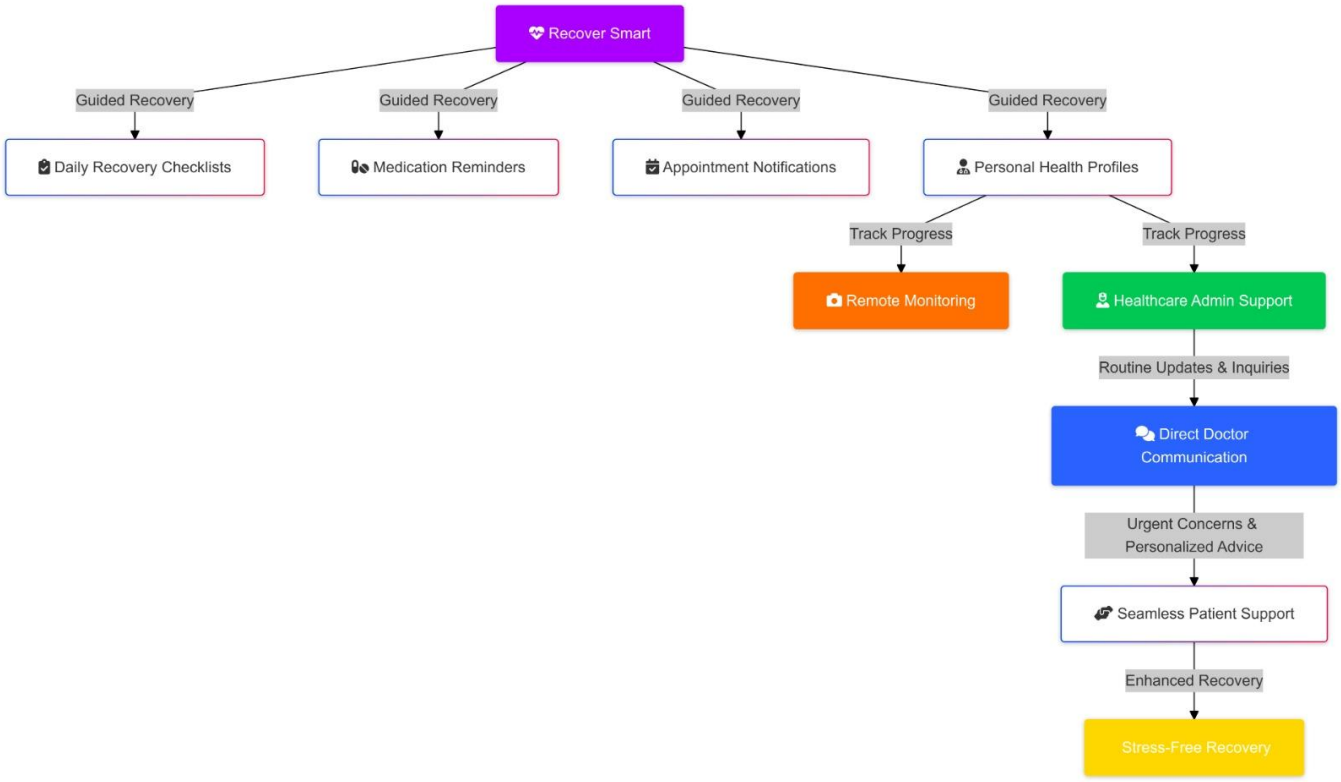


Figure 10: User Journey

The *Recover Smart* app offers a structured and user-friendly recovery experience for patients after surgery. Once discharged from the hospital, patients embark on a guided recovery journey using the app. The features are designed to provide daily support, ensure medical adherence, and allow real-time communication with healthcare professionals, promoting a seamless transition from hospital to home care.

Guided Recovery

This phase is central to the patient experience. It ensures that all essential recovery tasks are managed in an organized and timely manner.

- **Daily Recovery Checklists**
  Patients are presented with personalized checklists each day, tailored to their recovery needs. These tasks may include wound cleaning, prescribed exercises, dietary actions, or hydration reminders. Completing these tasks provides both visual and data-driven feedback to the patient and their healthcare provider. Over time, this builds consistency and empowers patients with a sense of progress and accountability during recovery.

- **Medication Reminders**

The app sends timely, automated alerts reminding patients to take their prescribed medications. These notifications are tailored to the user's schedule and continue until the medication is marked as taken. This greatly reduces the risk of missed doses, which is a common challenge in at-home recovery. The healthcare provider can also track compliance, ensuring that medication adherence is closely monitored even outside the hospital.

- **Appointment Notifications**
  To avoid missed check-ups, the app delivers alerts about upcoming doctor visits, whether virtual or in-person. Patients receive details including the time, type of appointment, and any pre-visit instructions. They can also respond to the notifications by confirming, requesting a reschedule, or asking for additional support. This streamlines appointment management and ensures better follow-through on follow-up care.

- **Personal Health Profiles**
  Each patient maintains a secure, real-time health profile within the app. This includes photos of wound healing, symptom updates, recovery logs, and notes from doctors. These profiles evolve as the patient progresses and serve as a comprehensive digital health record. The system's personalization helps adapt recovery guidance to individual needs and ensures that doctors have a clear view of the patient's condition.

Track Progress

Tracking progress is a key aspect of Recover Smart, helping healthcare professionals intervene early if necessary and offering patients reassurance during their healing.

- **Remote Monitoring**
  Through uploaded photos and checklist updates, doctors gain visibility into a patient's recovery without requiring an in-person visit. This allows them to monitor for signs of infection or delayed healing and respond quickly with guidance or treatment adjustments. The system reduces the need for physical appointments, improves patient confidence, and increases the efficiency of clinical decision-making.

- **Healthcare Admin Support**
  Support staff, such as hospital administrators or nurses, assist in managing the recovery flow from behind the scenes. They help schedule reminders, document progress, and respond to basic queries. This frees up doctors to focus on more critical medical needs while ensuring patients still receive timely responses and assistance for non-clinical concerns.

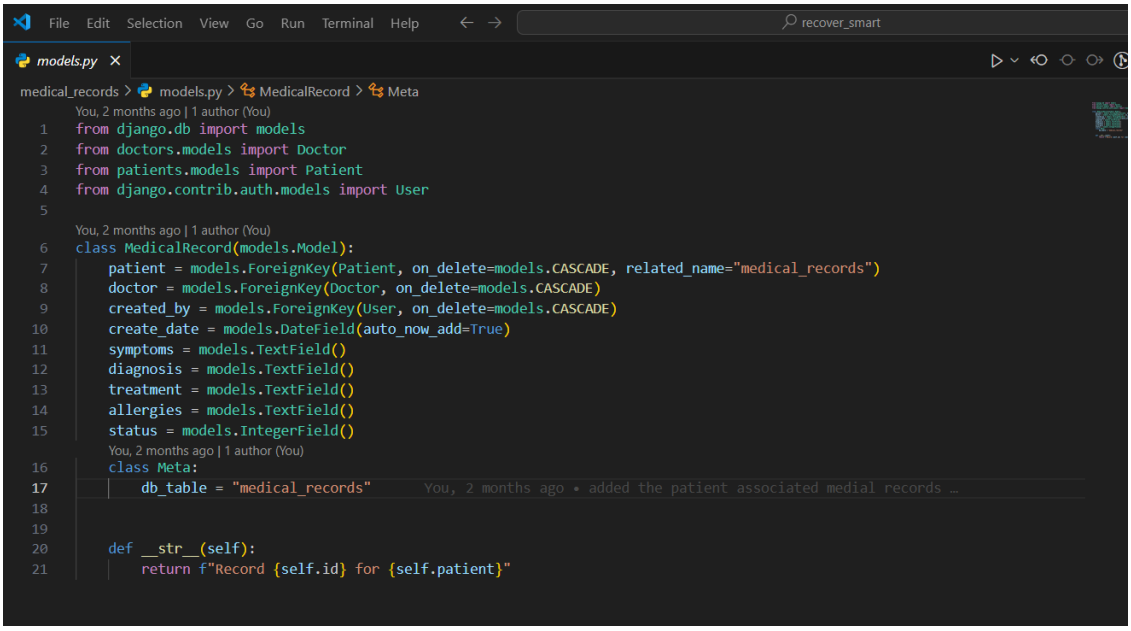
- **Routine Updates & Inquiries**
  Patients are encouraged to send updates or questions directly to their healthcare providers through the app. Whether it's a mild symptom or a clarification request, these communications are streamlined, documented, and easy to manage for both parties. This results in quicker feedback and helps reduce complications that arise from delays.

- **Urgent Concerns & Personalized Advice**
  In situations where patients experience unexpected symptoms or have urgent questions, they can immediately notify their doctor through the app. The provider responds with clear and personalized advice, whether it's reassurance or an instruction to seek in-person care. This immediate connection significantly improves patient safety and satisfaction.

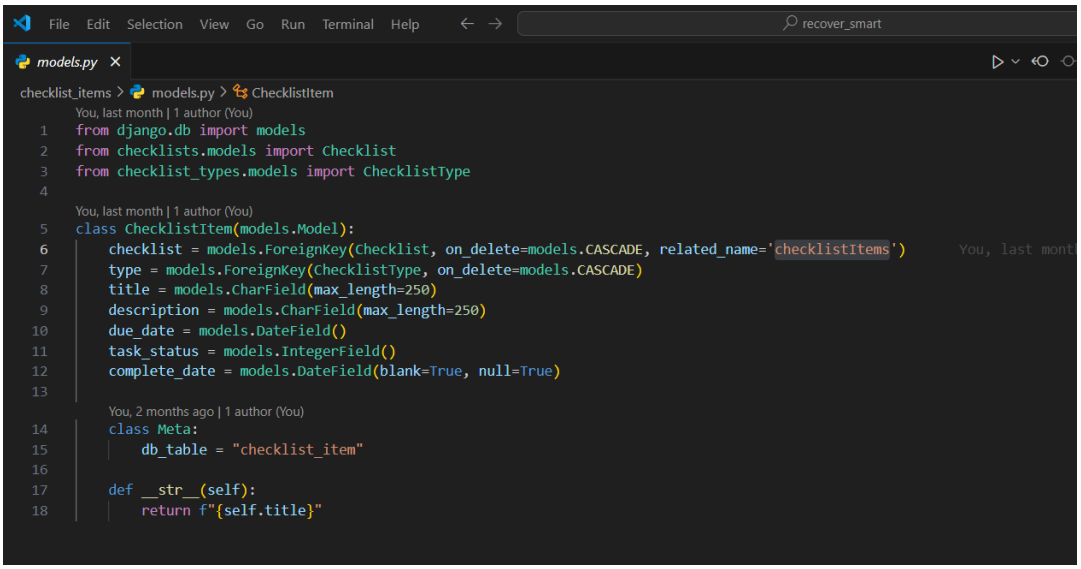
Enhanced Recovery

By combining structured checklists, smart notifications, and real-time health tracking, Recover Smart ensures that patients feel supported throughout their healing journey. Doctors remain connected and informed without being physically present, which allows for efficient use of their time and expertise. Ultimately, this app reduces the risk of complications and readmissions, providing a smooth and stress-free recovery experience for both patients and healthcare professionals.

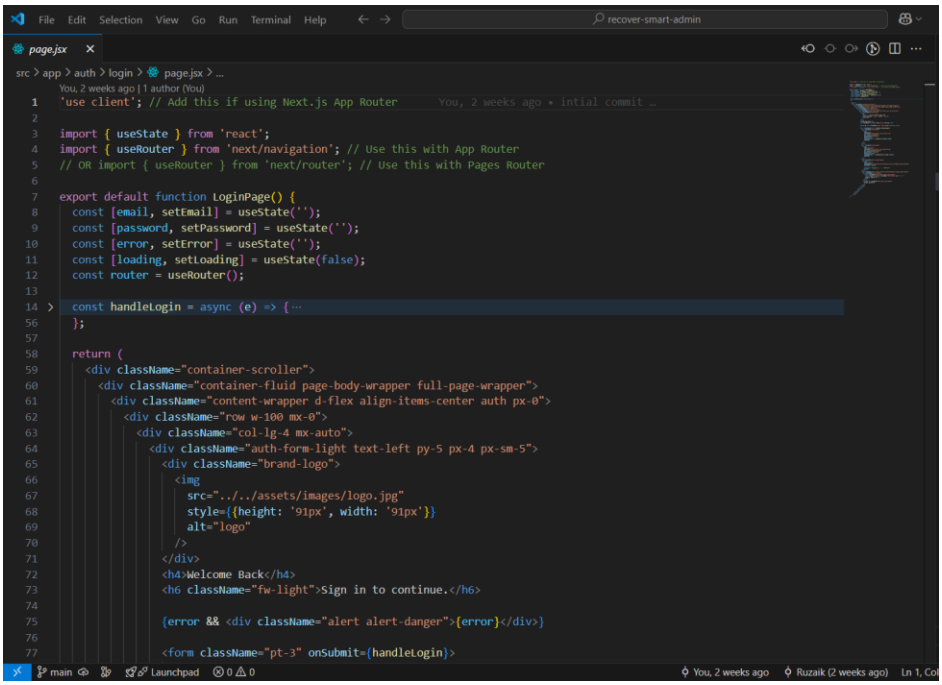
API Integration



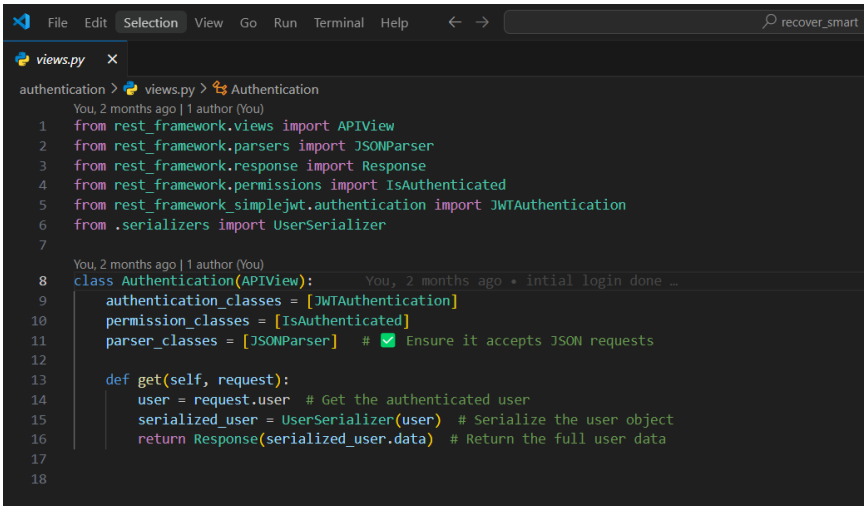
The MedicalRecord model in Recover Smart uses ForeignKey to link each record to a Patient, Doctor, and User (creator). It stores medical details in TextField (symptoms, diagnosis, treatment, allergies) and tracks the recovery status using IntegerField. The create_date (DateField) auto-logs entries, while the __str__ method ensures readable record representation.



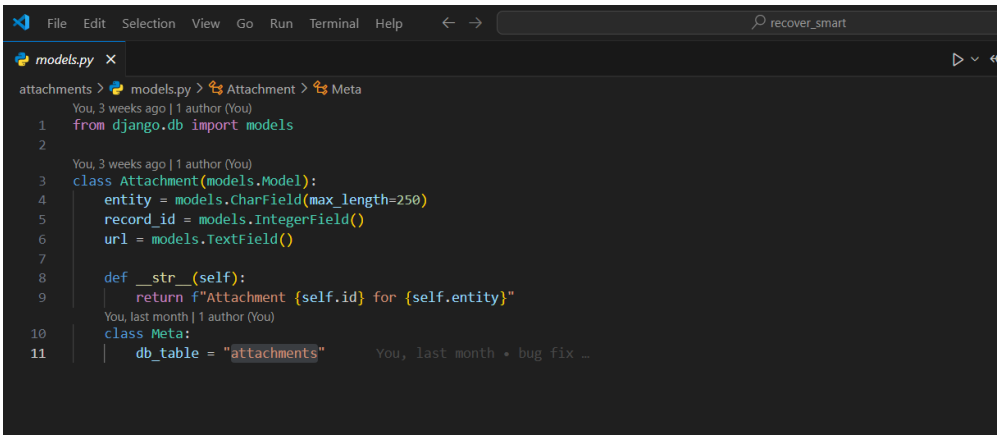
The **ChecklistItem** model in *Recover Smart* represents tasks in a recovery checklist. It links Checklist and ChecklistType via ForeignKey. Task details are stored in title and description (CharField), with due_date (DateField) and task_status (IntegerField) tracking progress. The complete_date (DateField) records completion, while __str__ returns the task title for readability.



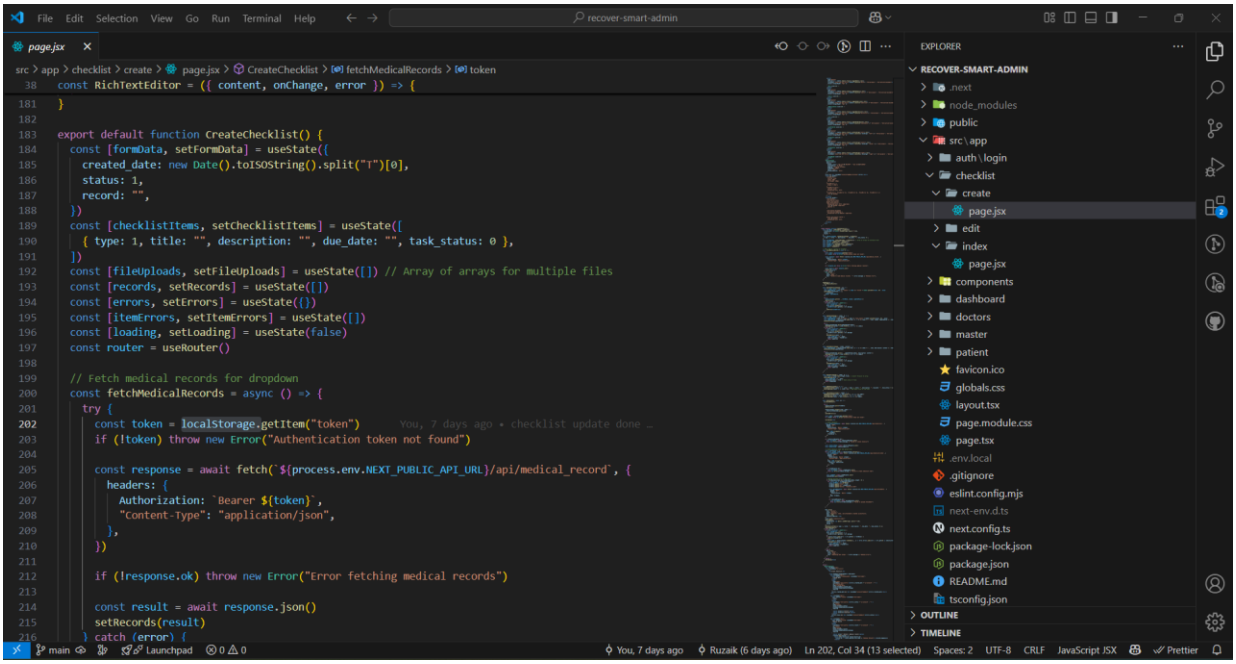
This Next.js login page for Recover Smart Admin uses React hooks like useState to manage email, password, error, and loading state. The handleLogin function (async) processes form submission. The useRouter hook helps navigate after login. The UI includes a logo, welcome message, error alert, and login form that triggers handleLogin on submit. It likely interacts with a backend to verify credential.



This Django REST Framework authentication API handles user authentication using JWTAuthentication. It ensures only authenticated users (via IsAuthenticated) can access the endpoint. The GET method retrieves the authenticated user from request.user, serializes it using UserSerializer, and returns the user data as a JSON response. The JSONParser ensures the API accepts JSON requests. This view is likely used for fetching user details after successful login.



The models.py file defines an Attachment model for the **Recover Smart** app, handling file attachments related to entities. It includes three fields: entity (CharField) to store the related entity's name, record_id (IntegerField) to link to a specific record, and url (TextField) for the file location. The __str__ method returns a readable string format, and Meta.db_table = "attachments" specifies the database table name. This model helps manage attachments efficiently.



The page.jsx file in the **Recover Smart Admin** panel handles checklist creation. It uses React's useState for managing formData, checklistItems, and fileUploads. The fetchMedicalRecords function retrieves medical records using an authentication token stored in localStorage. The request is sent to an API (NEXT_PUBLIC_API_URL), with an authorization header. Errors are caught and logged. This component ensures secure data retrieval and checklist creation, making patient record management more efficient for healthcare providers.

API Functions

The APIs that are used for the application are listed below. Further, they are documented in the website named postman (Postman, 2025) and the link to it is mentioned below and its details are explained in detail in the following part of the report:

https://documenter.getpostman.com/view/7158200/2sAYkErLE2#2d96da28-2ab4-43e5-b8a9-13a8a72ebf34

Table 3: API endpoint and funcionality

Endpoint	HTTP Method	Functionality
/api/attachments	GET, POST	Manage file attachments
/api/attachments/list	GET	Retrieve a list of attached files
/api/attachments/file	GET	Retrieve a single attached file
/api/auth/user/	POST	User authentication
/api/auth/	POST	Obtain authentication token
/api/token/refresh/	POST	Refresh authentication token
/api/checklists/item/	GET, POST	Manage checklist items
/api/checklists/edit/<int:pk>/	PUT	Edit a checklist item by ID
/api/checklists/item/<int:pk>	GET	Retrieve a checklist item by ID
/api/checklists/	GET, POST	Manage checklists
/api/checklists/<int:pk>	GET, DELETE	Retrieve or delete a checklist by ID
/api/checklists/record/<int:pk>	GET	Retrieve a specific checklist record
/api/medical_record/my_records	GET	Retrieve authenticated user's medical records
/api/medical_record	GET, POST	Manage medical records

Figure 11: API User Authentication

When the user tries to log in, they send a POST request to /api/auth/ with their username and password. If the login is successful, the server responds with an authentication token. This token allows the user to access the system. To get their account details, the user then makes a GET request to /api/auth/user/, but this time, they need to include the token in the request header as a Bearer Token to prove they're authorized.

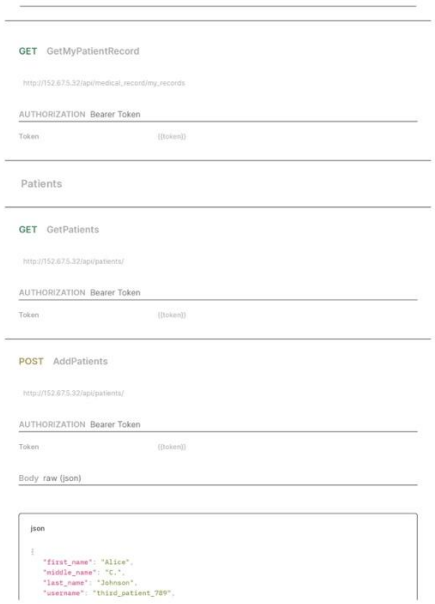


Figure 12:API - Patient Record

This image shows API endpoints for managing patient records. A GET request is sent to /api/medical_record/my_records/,to get a user's medical records, a but it requires a Bearer Token for authentication. Similarly, a GET request is sent to /api/patients/,to retrieve a list of patients. it

also requires a Bearer Token. To add a new patient, a POST request is sent to /api/patients/, including the Bearer Token for authentication and patient details (like first name, middle name, last name, and username) in JSON format within the request body.

Figure 13: API - Update Patient Information

This image shows how to update and get patient information using an API. The PUT request is used to change a patient's details like first name, last name, email, and phone number by sending the data to a specific web address. It requires a Bearer Token to make sure only authorized users can update the information. The data is sent in JSON format. The GET request is used to fetch patient details from another web address, and it also needs a Bearer Token for security.

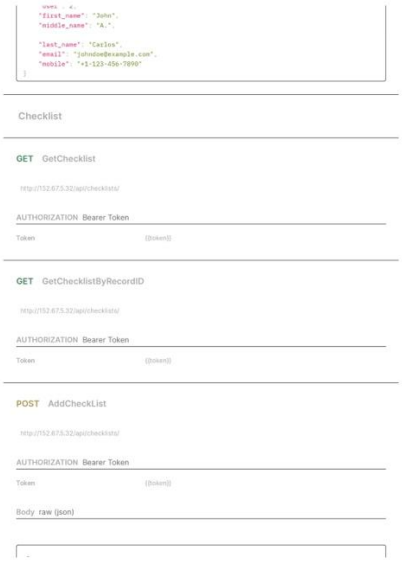


Figure 14: API - Managing Checklist

This image shows API documentation for managing checklists in a system. The GET request allows users to retrieve a list of checklists by making a request to a specific URL with a Bearer

Token for security. Another GET request lets users fetch a checklist using a specific record ID, also requiring a token. The POST request is used to add a new checklist by sending the details in JSON format to the server. All these requests ensure that only authorized users can access or modify checklists. write this in simple word

Figure 15: API - Creating Checklist

This image shows API instructions for updating and creating checklists. The PUT request (UpdateCheckList) is used to change the status of an existing checklist by sending updated information to the server, requiring a Bearer Token for security. The POST request (Create) is for adding a new checklist item by sending the details in JSON format. Both requests ensure that only authorized users can modify or add checklist data.

Figure 16: API - Update Checklist

The image shows API endpoints related to checklist items. The PUT request updates a checklist item using its ID, requiring authentication and a JSON body with relevant details. The DELETE request removes a checklist item using its ID and also requires authentication. The GET request retrieves a checklist item's details, requiring a valid Bearer Token for access. Additionally, there is a POST request for uploading a file related to the checklist item.

Figure 17: API - Managing Medical Record

This image shows how to use an API to manage medical records. It explains how to add a new record (POST) and update an existing one (PUT). The example includes details like the date, symptoms, diagnosis, treatment, and allergies. To use these features, a security token (Bearer Token) is needed. There is also an option to upload files related to checklist items.

Figure 18: API - Get Medical Record

This image shows how to use an API to get medical records and manage users. The **GET** request allows retrieving a specific medical record using its ID. The **POST** request is used to create a new

user, while **GET** requests are used to fetch user details or edit a specific user. All requests require a security token (Bearer Token) for secure access.

Figure 19: API - Delete Users

This image shows how to use an API for deleting users. It includes a DELETE request to remove a user, a POST request to upload attachments related to a checklist item, and a GET request to retrieve a list of uploaded files. Each request requires a token for authorization, and the upload request needs form data, including an ID, entity type, and file.

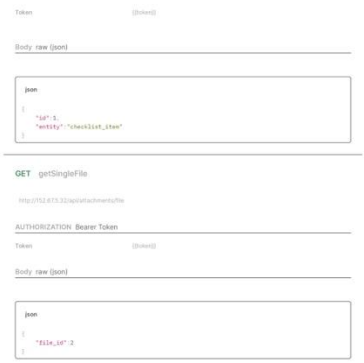


Figure 20: API - Handling Attachment

The image shows API documentation for handling attachments in a healthcare application. It includes a POST request to upload an attachment, requiring a JSON body with an ID and entity type (checklist_item). Additionally, there is a GET request to retrieve a specific file by providing its file_id in a JSON request body. Both requests require authentication using a Bearer Token.

Data Dictionaries

Table 4: Data Dictionary - User Authentication

Column Name	Data Type	Constraints	Description
ID	NUMBER	PRIMARY KEY NOT NULL	Unique identifier for users
PASSWORD	NVARCHAR(128)	NOT NULL	User password (hashed)
LAST_LOGIN	TIMESTAMP	NOT NULL	Last login timestamp
IS_SUPERUSER	NUMBER(1)	NOT NULL	Superuser flag
FIRST_NAME	NVARCHAR(150)	NOT NULL	First name of user
LAST_NAME	NVARCHAR(150)	NOT NULL	Last name of user
EMAIL	NVARCHAR(254)	UNIQUE NOT NULL	User email address
IS_STAFF	NUMBER(1)	NOT NULL	Staff user flag
IS_ACTIVE	NUMBER(1)	NOT NULL (0=Inactive, 1=Active)	Active status
DATE_JOINED	TIMESTAMP	NOT NULL	User account creation date

Table 5: Data Dictionary – Patients

Column Name	Data Type	Constraints	Description
PATIENT_ID	NUMBER	PRIMARY KEY NOT NULL	Unique identifier for patients
USER_ID	NUMBER	FOREIGN KEY (AUTH_USER.ID) NOT NULL	Links to AUTH_USER
FIRST_NAME	NVARCHAR(250)	NOT NULL	Patient first name
MIDDLE_NAME	NVARCHAR(250)	NOT NULL	Patient middle name
LAST_NAME	NVARCHAR(250)	NOT NULL	Patient last name
GENDER	NVARCHAR(50)	NOT NULL	Gender of patient
MOBILE	NVARCHAR(45)	UNIQUE NOT NULL	Contact number

Table 6: Data Dictionary – Doctors

Column Name	Data Type	Constraints	Description
DOCTOR_ID	NUMBER	PRIMARY KEY NOT NULL	Unique doctor identifier
FIRST_NAME	NVARCHAR(250)	NOT NULL	Doctor's first name
LAST_NAME	NVARCHAR(250)	NOT NULL	Doctor's last name
SPECIALIZATION	NVARCHAR(250)	NOT NULL	Medical specialty

EMAIL	NVARCHAR(200)	UNIQUE NOT NULL	Contact email

Column Name	Data Type	Constraints	Description
RECORD_ID	NUMBER	PRIMARY KEY NOT NULL	Unique identifier for records
PATIENT_ID	NUMBER	FOREIGN KEY (PATIENTS.PATIENT_ID) NOT NULL	Links to patient
DOCTOR_ID	NUMBER	FOREIGN KEY (DOCTORS.DOCTOR_ID) NOT NULL	Links to doctor
CREATED_BY	NUMBER	FOREIGN KEY (AUTH_USER.ID) NOT NULL	User who created the record
CREATE_DATE	DATE	NOT NULL	Date of record creation
DIAGNOSIS	NVARCHAR(500)	NOT NULL	Patient diagnosis
TREATMENT	NVARCHAR(MAX)	NOT NULL	Treatment details
STATUS	NUMBER	NOT NULL (0=Inactive, 1=Active)	Record status

Table 7: Data Dictionary - Records

Table 8: Data Dictionary – Messages

Column Name	Data Type	Constraints	Description
MESSAGE_ID	NUMBER	PRIMARY KEY NOT NULL	Unique identifier for message
PATIENT_ID	NUMBER	FOREIGN KEY (PATIENTS.PATIENT_ID) NOT NULL	Links to patient
SUBJECT	NVARCHAR(250)	NOT NULL	Message subject
MESSAGE	NVARCHAR(1000)	NOT NULL	Message content

Table 9: Data Dictionary - Checklist Types

Column Name	Data Type	Constraints	Description
TYPE_ID	NUMBER	PRIMARY KEY NOT NULL	Unique checklist type ID
TYPE_DESC	NVARCHAR(250)	NOT NULL	Checklist description

Table 10: Data Dictionary – Attachments

Column Name	Data Type	Constraints	Description
ATTACHMENT_ID	NUMBER	PRIMARY KEY NOT NULL	Unique attachment ID

ENTITY_ID	NUMBER	NOT NULL	Associated entity ID
RECORD_ID	NUMBER	FOREIGN KEY (RECORDS.RECORD_ID) NOT NULL	Links record
URL	NVARCHAR(2048)	NOT NULL	File URL

Table 11: Data Dictionary - Checklist Items

Column Name	Data Type	Constraints	Description
ITEM_ID	NUMBER	PRIMARY KEY NOT NULL	Unique checklist item ID
CHECKLIST_ID	NUMBER	FOREIGN KEY (CHECKLIST.CHECKLIST_ID) NOT NULL	Links checklist
TYPE_ID	NUMBER	FOREIGN KEY (CHECKLIST_TYPES.TYPE_ID) NOT NULL	Type of checklist
DESCRIPTION	NVARCHAR(250)	NOT NULL	Item description
DUE_DATE	DATE	NOT NULL	Due date for checklist item
TASK_STATUS	NUMBER	NOT NULL (0=Pending, 1=Completed)	Task status
COMPLETE_DATE	DATE	NOT NULL	Date of completion

UI/UX Design

Front End

Launch Screen

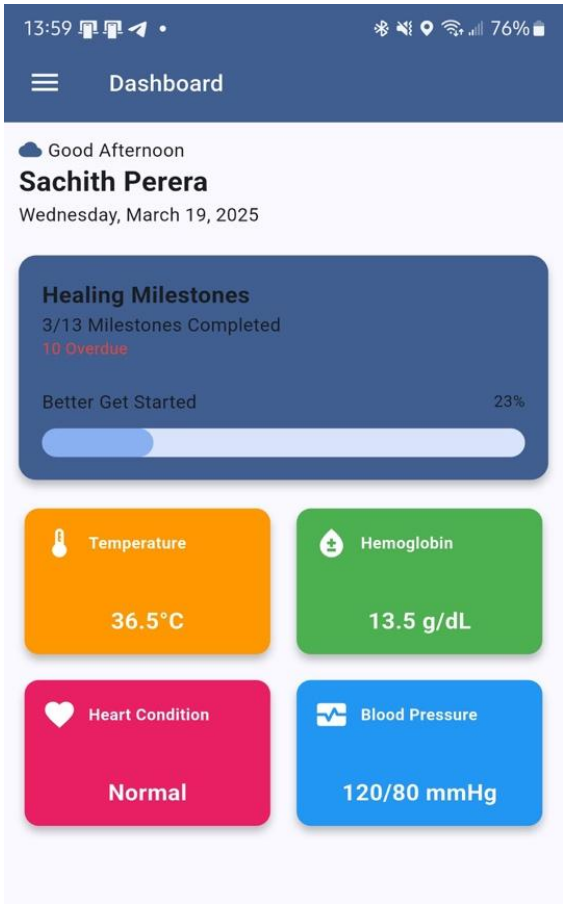


Figure 21: Launch Screen

The above image shows the Dashboard screen of the Recovery Smart mobile application, it is launch screen where the patient can view the quick overview of their recovery progress and key health metrics.

On the header section it has blue header bar. On the top left side, there's a hamburger menu icon (☰) where users can tap to open extra settings or navigation options. In the center of header, it is displayed as "Dashboard" which makes it clear that this is the main screen of the app. Below the header the application displays the greeting message along with the patient's full name and current data and time.

Healing Milestone section: This section helps the patient keep track of their recovery progress using a milestone system. The heading "Healing Milestones" is bolded to make it stand out. Just below, the app shows Milestones Completion status. To highlight any delays, the text "Overdue" is shown in red, signaling that the patient has fallen behind and

still has tasks to complete. A progress bar gives a clear visual of how much progress has been made in the recovery journey.
The main section of the dashboard gives a quick overview of the patient's health using four color-coded cards. The orange card shows body temperature, helping to check for fever or infection. The green card displays hemoglobin levels, which are important for oxygen flow in the blood. The red card highlights the heart condition, making sure any issues are noticed quickly. The blue card shows blood pressure, which is key for monitoring heart health. This information makes it easy for patients to understand their health status at a glance and stay on track with their recovery.

Healing Milestones Section

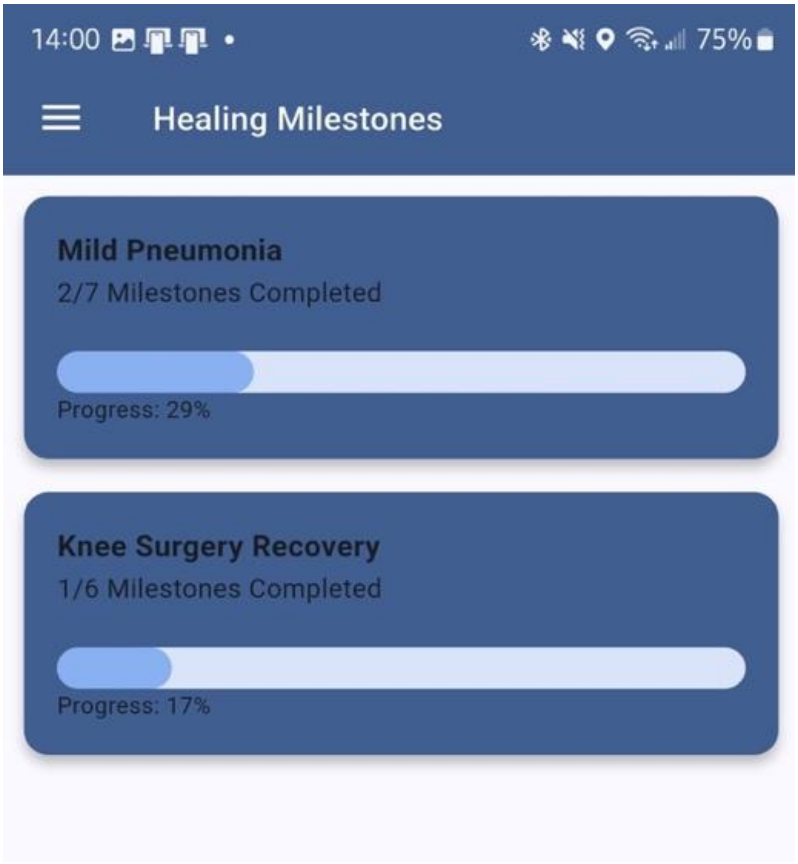


Figure 22: Milestone Screen

The Healing Milestone section is an important part of the app that helps patients track their recovery. They can access it directly from the main dashboard by tapping on "Healing Milestone." Depending on their health condition, they might have multiple milestones to complete. Each health issue is organized as a separate milestone, making it simple for patients to focus on one thing at a time. With a clear checklist to follow, they can complete tasks step by step, making the recovery process more manageable and structured.

Hamburger Menu (☰)

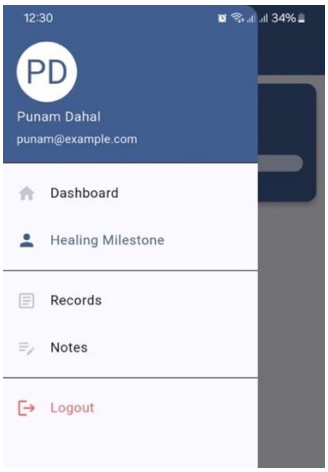


Figure 23: Hamburger Menu

The application has hamburger menu for quick navigation with in the application. The patient have option of view their profile, navigate to dashboard, healing milestone. Patient can also view additional option like viewing the records, taking the notes from this menu. Users also have logout option if they want to logout from the application.

Milestone Checklist

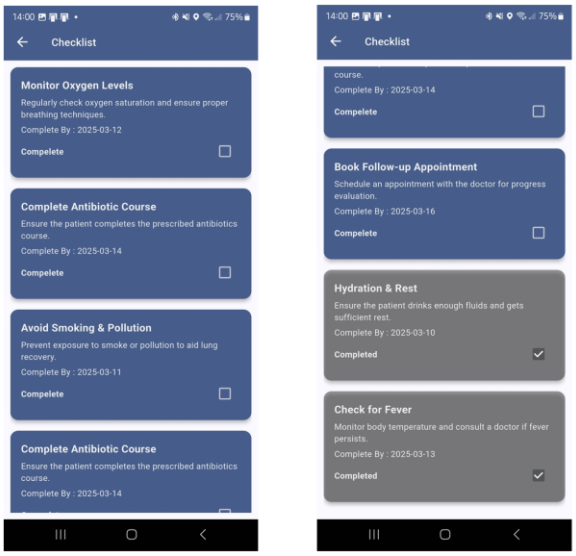


Figure 24: Milestone Checklist

The checklist includes a set of tasks recommended by the doctor to support the patient's recovery. Each milestone in the recovery process may have multiple checklists, depending on the severity of the illness. Patients are encouraged to review these checklists and complete all the tasks to ensure a smooth and speedy recovery. As they perform each task, they can mark it as complete by clicking on the checkbox, helping them track their progress and stay on top of their recovery plan.

Backend

Login Page

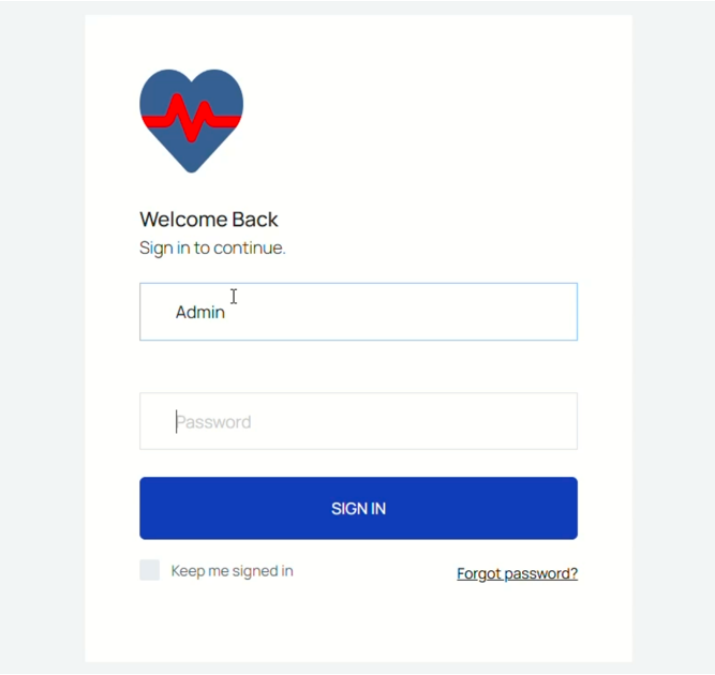


Figure 25: Admin Login Page

The login page for admin panel is simple, with the app logo and fields to enter your username and password. The user also has option to reset their password by going through Forgot password link.

Dashboard

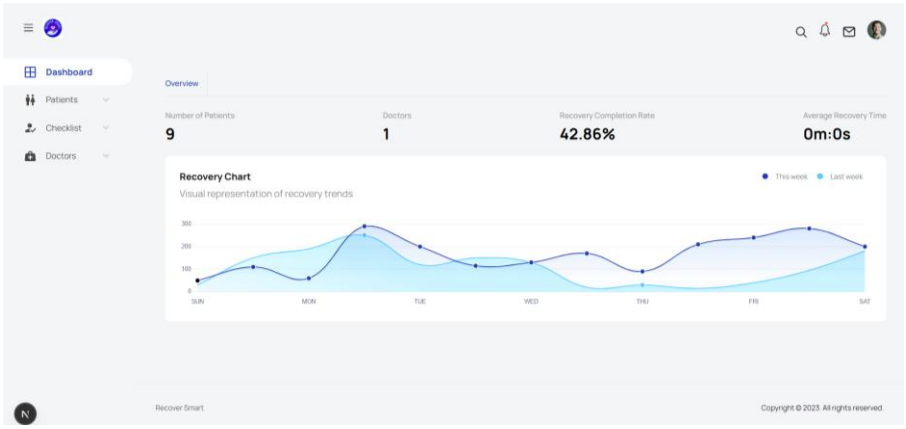


Figure 26: Dashboard of Admin Panel

Once the user is login to the page user can view the dashboard of the admin panel. User can navigate through hamburger menu. The hamburger menu has options for the Dashboard, Patients, Checklist and Doctors through which they can navigate to each section.

On the dashboard the user can overview the number of patients, number of doctors, Recovery completion rate and average recovery time. It also contains the recovery chart where user can also view the visual representation of the recovery trends.

View Patients

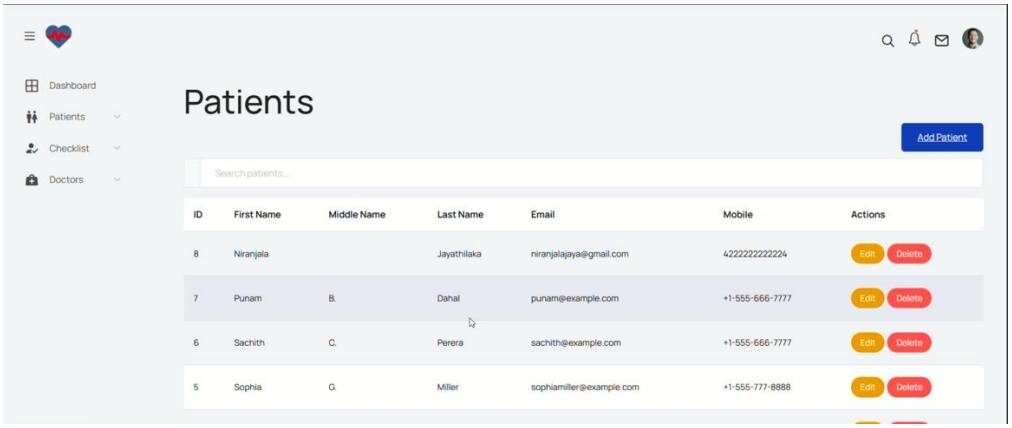


Figure 27: View Patients Information

This is the patient section on the admin panel. The admin can view the list of the patients on the system. The patient's information is displayed in tabular format so that it will be easier to view their details. The admin can view the patient information like patient Id, first name, middle name, last name, email and mobile number. The admin also has option to edit or delete the patient information from the system.

Add Patient

Figure 28: Add Patient

Admin can add the patient in the system in two different ways. Either they can click on "Add Patient" button from the patient list or they can navigate to Add patient section which is subsection of the patients through hamburger menu. For adding the patient, the admin must fill multiple fields with patient information like their first name, middle name (optional), last name, username, email address, mobile number.

Edit Patient

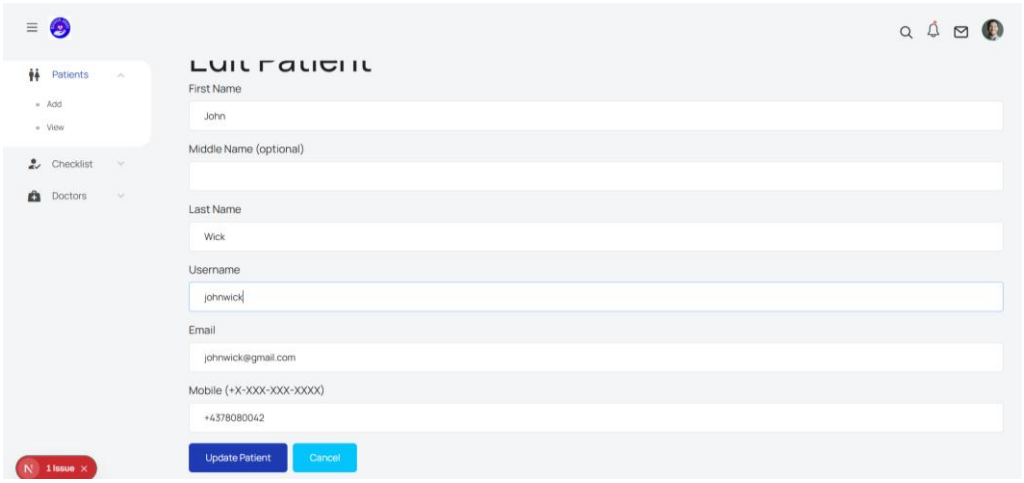


Figure 29: Edit Patient Information

Admin can edit the patient information and update on the system. User can simply go to patient list and click on edit task for respective patient. Once the information is edited, the updated information is reflected on patient table.

Create Medical Record

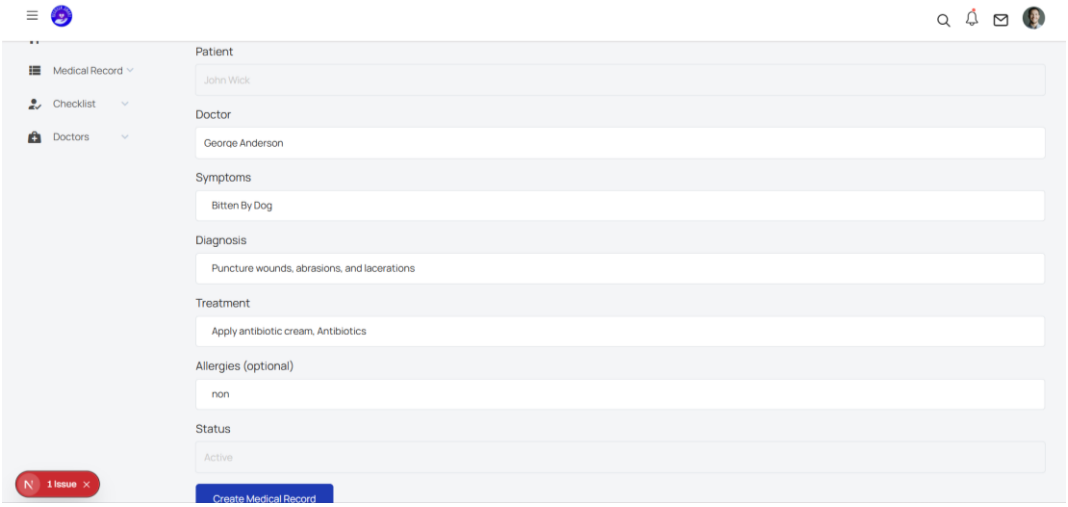


Figure 30: Create Medical Record

The admin can create the patient medical record my navigating to medical record section in hamburger menu. The information like patient name, doctor, symptoms, diagnosis, treatment, allergies (if available) and status needs to be filled to create the medical record.

View Checklist

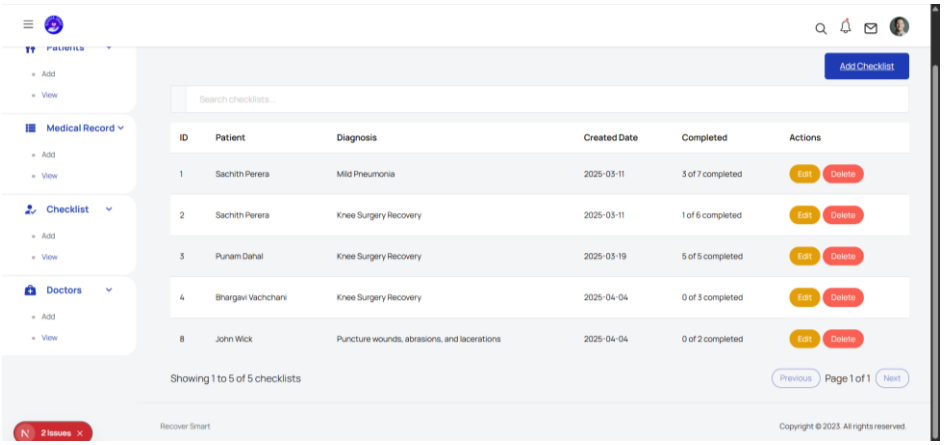


Figure 31: View Checklist

To view the checklist, user can navigate to checklist section from the hamburger menu. The information for the checklists is also displayed in tabular format which make it easier to read. In the checklist the user can view the checklist id, patient full name, diagnosis details, created date and completion status of checklist item. The admin can also edit or delete the checklist information from the system.

Create Checklist

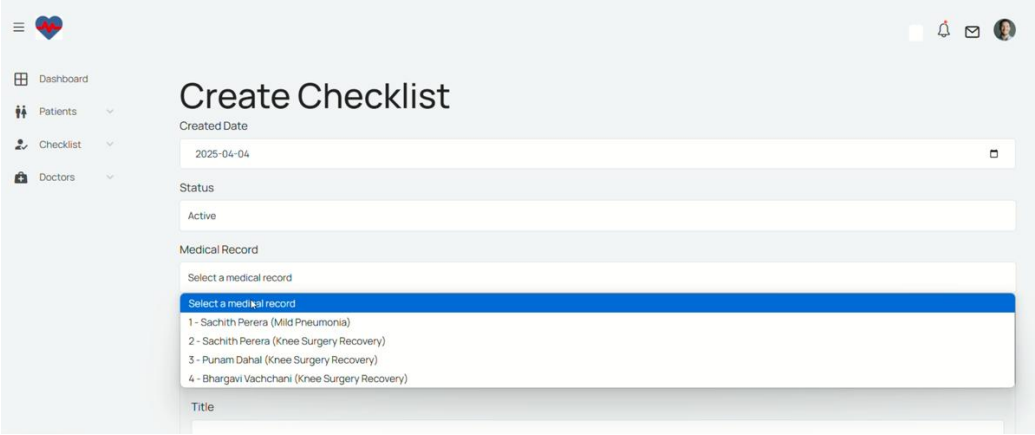


Figure 32: Create Checklist

To add the checklist, admin can click on “Add Checklist” link either on view checklist page or on subsection of checklist from navigation menu. To create the checklist, admin need to fill with checklist details like created date, checklist status, medical record of the patient and checklist item.

Checklist Item

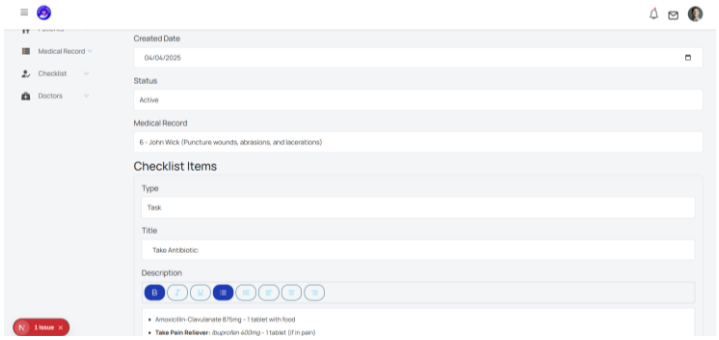


Figure 33: Checklist Items

The checklist item also has multiple information like type of checklist, title, description, due date for checklist, task status, attachment if required. The user can add multiple checklist item on single checklist based on the medical record of the patient.

Add doctor

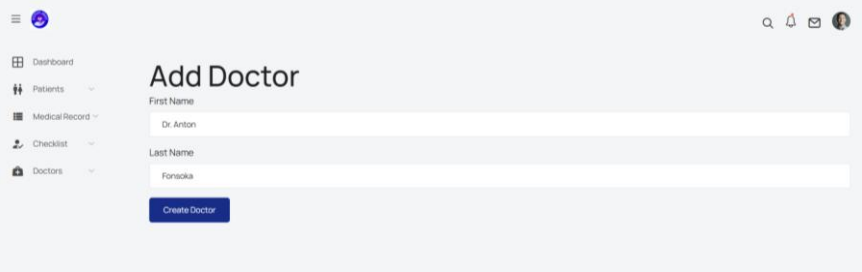


Figure 34: Add Doctor

To add the doctor, admin can navigate to doctor section from the menu. Admin should fill the doctor's first name and last name to create the doctor on the system.

View Doctor

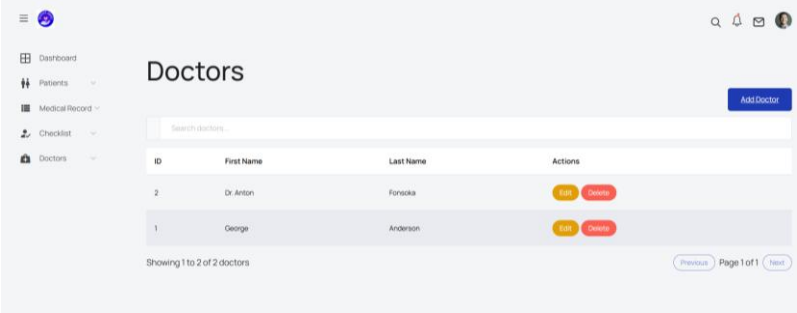


Figure 35: View Doctor

To view the list of doctors in the application, the user can navigate through doctor's section. Doctor's information like first name, last name. The admin can edit and delete the doctor from the system.

Test Cases

To keep the Recover Smart app running smoothly, a thorough set of test cases are put together. These cover everything from user authentication and patient recovery workflows to doctor-patient communication, notifications, and system performance. By testing both common and edge-case scenarios, a seamless and reliable experience is ensured.
You'll find the full set of test cases in the Appendix section of this report. Here's a quick overview of the key test scenarios:

1. **UI/UX Test Cases**
   These test cases are designed to ensure the app is easy to use, responsive, and simple to navigate.
   - **Login Screen:** Validates input fields, error handling for incorrect credentials, and session management.
   - **Dashboard:** Verifies navigation, visibility of key features, and responsiveness across devices.
   - **Forms & Buttons:** Ensures proper data input validation, button functionality, and error messages.

2. **Functional Test cases**
   These test cases focus on verifying the core functionality of the application.
   - **User Authentication:** Tests login, logout, password recovery, and session expiration scenarios.
   - **Milestone Completion:** Ensures users can view, interact with, and mark checklist tasks as completed.
   - **Medical Record Management:** Validates the upload, retrieval, and security of medical documents.
   - **Messaging System:** Tests sending, receiving, and storing messages between patients and doctors.
   - **Medication & Exercise Reminders:** Ensures timely notifications and proper tracking of user interactions.

3. **Future work**
   The test cases outlined in **Appendix Section** currently focus on defining expected behaviors. The next steps involve executing these tests and documenting the results under "Actual Outcome" and "Status (Pass/Fail)." This process will help identify any issues and ensure the application meets the highest standards of performance and reliability.

Test Cases Analysis

Based on the test cases provided, this appears to be a healthcare application with both patient and doctor interfaces. The application facilitates patient milestone tracking, wound photo uploads, doctor-patient communication, and reminder functionalities.

Completed Test Cases (Mobile App)

1. TC1: Sign in
   - Verifies authentication functionality with various scenarios
   - Tests valid/invalid credentials, field validations, and UI elements
   - Status: Completed

2. TC2: Complete Checklist
   - Tests patient milestone tracking functionality
   - Verifies milestone display, search, completion marking, and persistence
   - Includes responsive design and performance testing
   - Status: Completed

3. TC3: Upload Wound Photo
   - Tests document/photo upload functionality
   - Verifies file selection, upload process, and doctor access
   - Includes special notes functionality
   - Status: Completed

4. TC4: Notes
   - Tests communication between patients and doctors
   - Verifies message display, sending, timestamps, and UI elements
   - Tests chat history persistence and navigation
   - Status: Completed

Pending Test Cases (Admin Panel)

5. TC5: View Monitoring Data
   - Tests doctor's ability to view patient data
   - Verifies access to wound photos and completed checklists
   - Status: Pending

6. TC6: Receive Medication Reminder
   - Tests medication reminder functionality
   - Verifies reminder creation, triggering, and notification
   - Status: Pending

7. TC7: Exercise Reminder
   - Testing exercise reminder functionality
   - Similar to medication reminder but for exercises

- Status: Pending

8. TC8: Minimize In-Person Consultations
   - Tests the overall impact of the application
   - Measures reduction in unnecessary in-person consultations
   - Status: Pending

Implementation Status

As mentioned above, the mobile app testing is completed, which is reflected in the "Completed" status of TC1-TC4. These test cases cover the core patient-facing functionalities including authentication, milestone tracking, photo uploads, and communication.

The admin panel implementations have some pending components, specifically TC5-TC8, these functionalities appear to be backend or administrative features that are still being implemented. The detailed test cases are listed in Appendix.

Coding Standards

We have maintained a clean, secure, and efficient codebase by following below coding standards.

1. **Follow PEP 8 (Python Style Guide)**
   We have used 4-space indentation for readability and kept line length under 79 characters to improve code clarity. We have followed snake_case for variables and function names (user_name, get_user_data()) and CamelCase for class names (UserProfile, OrderHistory). Consistent quotes have been maintained throughout the code, using either single ('hello') or double ("hello") (Rossum, Warsaw, & Coghlan, 2001).

2. **Views: Keep It Clean**
   We have used Class-Based Views (CBVs) instead of function-based views for CRUD operations to ensure modular and reusable code. Business logic has been moved to models or service layers to keep views lean and maintainable. We have also applied decorators like @login_required to enforce authentication and prevent unauthorized access. (Leino, 2016)

3. **Model & Field Naming**
   We have used CamelCase for models (UserProfile, OrderDetail) and snake_case for field names (created_at, email_address). To maintain clarity in relationships, we have assigned descriptive related names for foreign key associations, such as related_name="user_posts". (Alsuhaibani, et al.)

4. **Database Best Practices**
   We have optimized database queries by implementing select_related() and prefetch_related() to prevent the N+1 problem. Raw SQL usage has been avoided unless necessary, ensuring security and maintainability through Django's ORM. (Holt, Victoria, Ramage, Kear, & Heap, 2015)

5. **Security Best Practices**
   We have ensured that sensitive data such as API keys and database credentials are not exposed by storing them in .env files. CSRF protection has been enabled to prevent cross-site request forgery attacks. Additionally, we have validated and sanitized all form inputs to prevent SQL injection and XSS attacks.

PIPEDA Compliance for Recover Smart

At Recover Smart, we are committed to protecting patient privacy and ensuring compliance with the Personal Information Protection and Electronic Documents Act (PIPEDA) (Nikki, 2007). Since our system facilitates post-surgery interactions between patients and doctors, we take data security and regulatory adherence seriously.

Responsibilities Under PIPEDA

Accountability

To maintain strong privacy controls, Recover Smart follows strict guidelines. Any third-party services used, like cloud providers, must comply with privacy laws. Regular privacy risk assessments are conducted to identify and reduce possible risks, keeping patient data safe.

Transparency & Consent
Patients are informed clearly about what data is collected, why it's needed, and how it is used. Consent is required before collecting, sharing, or using personal information. The app includes a simple, easy-to-understand privacy policy that is regularly updated so users always know their rights.

Limited Data Collection & Use
Recover Smart only collects the minimum personal data required for its features to work effectively. Patient data is not used beyond its original purpose unless the user provides additional consent through the app's permission settings.

Security & Data Protection
Strong encryption is in place to protect data when stored and transmitted. Role-based access control (RBAC) and multi-factor authentication (MFA) prevent unauthorized access. Security audits and vulnerability assessments are performed regularly to ensure the system remains secure.

User Access & Data Accuracy
Patients have control over their information and can view, update, and correct their data anytime. The app ensures that stored information is accurate and current to avoid misinformation that could affect recovery progress.

Data Retention & Deletion
Personal data is only kept for as long as necessary. When it is no longer needed, the app follows secure deletion protocols to remove the data safely while protecting user privacy.

Breach Response & Notification
If a security breach occurs, built-in alerts notify affected users and relevant authorities immediately. Backup communication channels ensure that urgent medical needs are not affected by any system disruption.

Ensuring PIPEDA Compliance in Recover Smart
As a mobile healthcare app, Recover Smart includes additional safeguards to ensure compliance with privacy best practices.

Clear & Accessible Privacy Policies
Users are shown a privacy notice when installing the app. Important privacy details are provided upfront, while more in-depth information is available if needed.

Secure App Infrastructure
Recover Smart uses encryption to protect health data. Secure authentication methods prevent unauthorized access, and real-time monitoring ensures that critical patient events are detected and responded to quickly.

Minimal Data Permissions
The app only requests the necessary permissions needed for its functionality. Patient data is never shared with third parties without explicit consent, which users can manage within the app.

Timely Consent & Policy Updates
Consent is requested at multiple stages—before installation, during data collection, and when new features requiring additional permission are introduced. Users are notified when privacy policies change, or new permissions are required.

Compliance with Standards

Recover Smart is fully aligned with Canadian healthcare standards, making sure we meet all the important rules and regulations to provide a secure, accessible, and effective mobile healthcare app for patients, doctors, and admins.

Privacy and Security Compliance

Recover Smart follows both the Personal Information Protection and Electronic Documents Act (PIPEDA) and the Personal Health Information Protection Act (PHIPA), protecting personal health information (PHI). The app makes sure that (Austin, 2006):

- We get explicit consent from all users (patients, doctors, and admins) to upload and share images.
- Patients have full access to their information and can update or fix anything wrong.
- All data is encrypted both when it’s stored and sent, making sure it's safe and confidential.

Interoperability and Data Standards

Recover Smart makes sure it integrates well with existing healthcare systems, following Canada Health Infoway guidelines (Lorenzi, Kouroubali, Detmer, & Bloomrosen, 2009) and the best practices of Health Level Seven (HL7) and Fast Healthcare Interoperability Resources (FHIR) standards. This helps to:

- Exchange data including photos securely with healthcare providers and systems, so everything works smoothly.
- Keep data consistent and well-organized, which is important for the app to fit into Canada’s digital health environment.

Accessibility Standards

We made Recover Smart comply with the Accessible Canada Act (ACA) and the Web Content Accessibility Guidelines (WCAG 2.1) so that it’s:

- Usable by everyone, including those with disabilities. The app works with screen readers, offers high-contrast modes, and lets users adjust text sizes.
- Designed for inclusivity, giving all users the ability to manage their health easily.

Clinical Safety Standards

Recover Smart follows the clinical safety rules from the Canadian Medical Association (CMA). This means:

- All health-related content, like checklists, medicine schedules, and wound care instructions, is accurate and checked by healthcare professionals.
- The app sends notifications and reminders for medication and appointments, helping to keep patients safe.

Digital Health Standards

We made sure Recover Smart fits into Canada's Digital Health Strategy by (Marchildon, Allin, & Merkur, 2013):

- Offering tools that help patients keep track of their health and communicate with their doctors, providing continuous care.
- Making sure it can connect with other digital health platforms to improve overall healthcare coordination.

Cybersecurity

Recover Smart follows the cybersecurity best practices from the Canadian Centre for Cybersecurity. The app:

- Uses encryption to protect all health data, keeping it safe from unauthorized access.
- Makes sure users log in securely by using two-factor authentication (2FA) and has role-based access control for extra security.

Medical Device Regulations

If the app has features like wound analysis through photos, we're following Health Canada's Medical Device Regulations. We classify these features properly and test them to meet safety standards. If any part of Recover Smart is considered a medical device, we'll make sure it's classified correctly and certified as required.
By following all these standards, Recover Smart ensures it's fully compliant, secure, and easy for everyone to use. We're committed to offering a trusted healthcare solution.

Business Analysis for Recover Smart

Business Needs and Objectives

The project addresses the critical need for structured post-discharge care, reducing hospital readmission rates and improving patient outcomes. The major objectives of this include:

- Enhanced Patient Monitoring: This will enable doctors to track patient progress remotely through photo uploads of the wound and completion of checklists.

- Improved Compliance: Reminders regarding medication, appointments, and exercises will be automatically sent to ensure that recovery plans are adhered to.
- Operational Efficiency: Reducing hospital workloads by minimizing unnecessary readmissions and in-person consultations.

Target Users and Stakeholders

- Patients: Individuals recovering at home who need structured care plans and regular follow-ups.
- Doctors: Healthcare professionals assessing patient progress and providing remote recommendations.
- Hospitals and Clinics: Medical institutions looking to improve patient care quality while optimizing resources.
- Administrators: Responsible for onboarding patients and managing user access.

Technological Implementation

- Cross-Platform App Development Framework: Flutter is selected for Android and iOS.
- Back-end development: Python is selected for the back-end development incorporating the Django framework.
- Cloud-based Backend: Oracle database for patient records, updates in real-time.
- Frontend technologies: JavaScript, jQuery, HTML, CSS, and bootstrap are used for the project.

Cost-Benefit Analysis

- Cost Savings for Hospitals: Reduced readmission rates lower operational costs.
- Better Health Outcomes: Early interventions prevent complications, hence improving patient satisfaction.
- Revenue Potential: Subscription-based or hospital-integrated pricing models for sustaining the solution.

Risks and Mitigation Strategies

Risk	Impact	Mitigation Strategy
Data Privacy Concerns	High	Implement strong encryption and comply with healthcare data regulations.
User Adoption Challenges	Medium	Provide user training and intuitive UI/UX design.
Technical Issues (App Crashes, Bugs)	Medium	Conduct careful testing before deployment.
Delayed Medical Response	High	Implement alert escalation mechanisms for critical cases.

Summary

Recover Smart is a healthcare support application designed to assist patients in managing their recovery process after hospital discharge. The app aims to bridge the gap between patients and healthcare providers by offering personalized checklists, reminders for medications and exercises, and secure communication channels for ongoing monitoring.
Developed using Flutter for cross-platform compatibility and Django for backend services, Recover Smart enables patients to stay on track with their recovery by receiving timely notifications, uploading wound images, and receiving feedback from doctors remotely. It also empowers doctors and healthcare administrators with tools to track patient progress, reduce unnecessary in-person visits, and streamline post-discharge care management.
With features such as JWT-based authentication, MySQL database support, and email integration through SMTP, the application ensures secure and efficient data handling. Deployed on Oracle Cloud Infrastructure, the system is designed for scalability, performance, and compliance with healthcare data regulations like PIPEDA and PHIPA.
Overall, Recover Smart combines modern technology with user-centered design to improve the post-hospital recovery experience, reduce complications, and enhance communication between patients and healthcare providers.

Future Recommendation

As *Recover Smart* continues to evolve, there are several areas of improvement and expansion that can enhance the user experience, increase efficiency, and extend the application's impact. Based on the current testing phase and real-world functionality assessment, the following future recommendations are proposed:

1. **Reliable and Timely Notification System**
   While the reminder feature has been implemented, its notification delivery still requires thorough validation. Future work should focus on ensuring that reminders, especially for medications and exercises are triggered precisely at the scheduled time without any delays. Additionally, support for multiple notification channels (push notifications, SMS, email) should be tested further and made more configurable to accommodate user preferences.

2. **Improved Reminder Tracking and Logging**
   Introduce a log system where users and healthcare providers can view the history of triggered reminders, missed tasks, or user actions taken after reminders. This can help doctors better understand patient adherence and tailor follow-ups accordingly.

3. **Notification Delivery Status Monitoring**
   Add a feature that confirms whether a notification was successfully delivered and seen by the user. This will help identify potential communication gaps and ensure patients are staying informed.

4. **Optimization of In-Person Consultation Data Analysis**
   As part of the system's value proposition, it's important to track how the app reduces unnecessary hospital visits. Future enhancements should include automated analytics that categorize consultations as necessary or avoidable, considering external factors such as health trends, patient age group, or policy changes. This insight will help healthcare administrators evaluate the app's effectiveness over time.

5. **User-Customizable Reminder Settings**
   Allow patients to adjust the frequency, method (push/SMS/email), and sound preferences for reminders within the app. This improves personalization and accessibility, especially for elderly users or those with specific needs.

6. **Error Handling and Retry Mechanisms**
   In the event that a reminder or notification fails to send (e.g., due to poor network or email configuration), the system should log the error and attempt re-delivery. This will increase reliability and trust in the system.

7. **Integration with Wearable Devices**
   Future updates could explore integration with smartwatches or fitness bands to sync exercise reminders, recovery progress, and vitals tracking, offering a more holistic recovery monitoring experience.

8. **Post-Deployment Monitoring Tools**
   Include tools that monitor backend performance, uptime, and user activity to proactively identify bugs or usability issues in the live system.

Bibliography

Alsuhaibani, Reem, Newman, C., Decker, M., Collard, M., & Maletic, J. (n.d.). On the naming of methods: A survey of professional developers. *In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)* (pp. 587-599). IEEE.

Austin, L. M. (2006). Reviewing PIPEDA: Control, Privacy and the Limits of Fair Information Practices. *SSRN Electronic Journal. https://papers.ssrn.com/sol3/Delivery.cfm/SSRN_ID1169162_code333799.pdf?abstractid=1169162&mirid=1*.

Bouras, A. (2024). Python and algorithmic thinking for the complete beginner: Learn to think like a programmer by mastering Python programming and algorithmic foundations. *Packt Publishing Ltd*.

Forcier, J., Bissex, P., & Chun, W. (2008). Python Web Development with Django. *http://cds.cern.ch/record/1211020/*.

Hamidli, N. (2023). Introduction to UI/UX design: key concepts and principles. *Preuzeto*, p. 28.

Holt, Victoria, Ramage, M., Kear, K., & Heap, N. (2015). The usage of best practices and procedures in the database community. *Information Systems*(49), 163-181.

Kuhrmann, M., Diebold, P., Munch, J., Tell, P., Garousi, V., Felderer, M., . . . Prause, C. R. (2017). Hybrid software and system development in practice: waterfall, scrum, and beyond. *Proceedings of the 2017 international conference on software and system process*, 30-39.

Leino, M. (2016). *Coding Standards in Web Development.*

Lorenzi, N. M., Kouroubali, A., Detmer, D. E., & Bloomrosen, M. (2009). How to successfully select and implement electronic health records (EHR) in small ambulatory practice settings. *BMC Medical Informatics and Decision Making, 9(1). https://doi.org/10.1186/1472-6947-9-15*.

Marchildon, G. P., Allin, S., & Merkur, S. (2013). Canada: Health system review PubMed, 15(1). *https://pubmed.ncbi.nlm.nih.gov/23628429*, 1 - 179.

Nikki, S. (2007). Canada reviews PIPEDA. *Information Management Journal*. Retrieved from https://www.proquest.com/docview/227757754?parentSessionId=PqD6BJMC5mIJX2QGZgAuNkkEyYCDH0xBksfTOcQbOOc%3D&accountid=39340&sourcetype=Scholarly%20Journals

Postman. (2025, 03 21). *RecoverSmart*. Retrieved from Postman: https://documenter.getpostman.com/view/7158200/2sAYkErLE2#2d96da28-2ab4-43e5-b8a9-13a8a72ebf34

Reddy, M. P. (2021). Analysis of component libraries for REACT JS. IARJSET, 8(6). *https://doi.org/10.17148/iarjset.2021.8607*, 43-46.

Rossum, V., Warsaw, B., & Coghlan, N. (2001). *PEP 8–style guide for python code.* (Vol. 28). Python. org.

Shao, H., Sun, D., Wu, J., Zhang , Z., Zhang, A., Yao, S., . . . Abdelzaher, T. (2020). paper2repo: Github repository recommendation for academic papers. *Proceedings of The Web Conference 2020.*

Uzayr, S. b. (2022). *Front end development and UX design.* (1st Edition ed.). CRC Press.

Appendix

Test Cases

Login

Test Case ID:	TC1	**Test Case name:**	Sign in No. 1
Test Case Description:	This test case verifies that the Sign in page correctly validates user input and displays appropriate error messages for incorrect inputs.It also ensures successful Sign in with valid credentials.		
Pre-Condition:	The application is installed and accessible. The user is on the Sign in page.	**Post Condition:**	N/A

Step No.	Action	Expected Output	Actual Output	Comments	Status (Pass/Fail)
1	Enter a valid username and password, then click 'Sign in'	User is successfully logged in and redirected to the homepage	User is successfully logged in and redirected to the homepage	Ensure valid credentials are used	Completed
2	Leave both username and password fields blank, then click 'Sign in'	Error message: "Username and Password cannot be empty"	Error messages to enter the Username and Password	Verify error message appears correctly	Completed
3	Enter a valid username but leave the password field blank, then click 'Sign in'	Error message: "Password cannot be empty"	Error messages to enter the Password	Check if password validation works	Completed
4	Enter a valid password but leave the username field blank, then click 'Sign in'	Error message: "Username cannot be empty"	Error messages to enter the Username	Check if username validation works	Completed
5	Enter an incorrect username and password, then click 'Sign in'	Error message: "Invalid username or password"	Error message: "Invalid username or password"	Ensure incorrect credentials fail Sign in	Completed
6	Enter a valid username but an incorrect password, then click 'Sign in'	Error message: "Invalid username or password"	Error message: "Invalid username or password"	Validate incorrect password scenario	Completed
7	Enter an incorrect username but a valid password, then click 'Sign in'	Error message: "Invalid username or password"	Error message: "Invalid username or password"	Validate incorrect username scenario	Completed
8	Enter valid credentials and press 'Enter' key instead of clicking 'Sign in'	User is successfully logged in and redirected to the homepage	User is successfully logged in and redirected to the homepage	Check if 'Enter' key triggers Sign in	Completed
9	Check if the password field hides input characters	Password should appear as asterisks or dots	Password should appear as asterisks or dots	Ensure password field masks input	Completed
10	Enter credentials and refresh the page	Fields should be cleared	Fields should be cleared	Verify form reset on refresh	Completed

RECOVER SMART
YOUR CARE. YOUR GUIDE.
RECOVER SMART BY YOUR SIDE

Completed Milestone

Test Case ID:	TC2	**Test Case name:**	Complete Checklist
Test Case Description:	This testcase ensure patients can complete and submit checklists successfully.		
Pre-Condition:	The patient must be logged into their account. The checklist feature must be enabled. The system must have an active internet connection.	**Post Condition:**	The submitted checklist is stored and accessible by the doctor.

Step No.	Action	Expected Output	Actual Output	Comments	Status (Pass/Fail)
1	Log in as a patient.	Patient successfully logs in.	Patient successfully logs in.	Ensure valid credentials are used.	Completed
2	Verify the Milestone page loads correctly	Open the Patient Milestone page in the app	Open the Patient Milestone page in the app	The page should load without errors, displaying all elements properly	Completed
3	Verify that the title "You have got X Milestones to complete" appears correctly	Open the page and check the title section	Open the page and check the title section	The title should dynamically update based on the number of milestones left	Completed
4	Verify that the search bar is present and functional	Locate the search bar and try entering a milestone name	Locate the search bar and try entering a milestone name	The search bar should be visible and allow users to input text	Completed
5	Verify that all milestone tasks are displayed	Open the Milestone page and review the list	Open the Milestone page and review the list	The list should display all milestones assigned to the patient	Completed
6	Verify that a milestone can be marked as completed	Click the checkbox next to a milestone	Click the checkbox next to a milestone	The checkbox should be checked, indicating completion	Completed
7	Verify that completed milestones remain checked after refresh	Mark a milestone as completed, refresh the page	Mark a milestone as completed, refresh the page	The milestone should remain checked after reloading the page	Completed
8	Verify that incomplete milestones remain unchecked after refresh	Ensure a milestone is unchecked, refresh the page	Ensure a milestone is unchecked, refresh the page	The milestone should remain unchecked	Completed
9	Verify that searching for an existing milestone filters the list correctly	Enter a milestone name in the search bar	Enter a milestone name in the search bar	The list should update to show only matching milestones	Completed
10	Verify that searching for a non-existing milestone shows no results	Enter a non-existent milestone name in the search bar	Enter a non-existent milestone name in the search bar	The list should be empty, showing no results	Completed
11	Verify that the checkboxes are easily clickable	Click the checkbox and observe the response	Click the checkbox and observe the response	The checkbox should toggle state smoothly when clicked	Completed
12	Verify that the milestone text is readable and well-spaced	Observe milestone text on different devices	Observe milestone text on different devices	The text should be clear, readable, and well-spaced	Completed
13	Verify that the page loads within 3 seconds	Load the page and measure the response time	Load the page and measure the response time	The page should load within 3 seconds	Completed
14	Verify compatibility on different devices (mobile, tablet, desktop)	Open the page on different screen sizes	Open the page on different screen sizes	The UI should be responsive and adjust properly on different devices	Completed

RECOVER SMART
YOUR CARE. YOUR GUIDE.
RECOVER SMART BY YOUR SIDE

Minimize In-Person Consultation

Test Case ID:	TC8	**Test Case name:**	Minimize In-Person Consultations
Test Case Description:	Monitor the impact of the application in reducing unnecessary in-person consultations while maintaining quality care.		
Pre-Condition:	The application must be implemented for a test group. A baseline measurement of in-person consultations must be established before implementation. Tracking mechanisms must be in place for monitoring consultation frequency.	**Post Condition:**	A measurable reduction in unnecessary in-person consultations is observed.

Step No.	Action	Expected Output	Actual Output	Comments	Status (Pass/Fail)
1	Implement the application for a test group.	The system is deployed and functional.		Ensure all users (patients and doctors) are properly onboarded.	Pending
2	Track the number of in-person consultations before and after implementation.	Consultation data is collected accurately.		Ensure data integrity and categorize necessary vs. unnecessary consultations.	Pending
3	Compare the data.	A reduction in unnecessary in-person consultations is observed.		Consider external factors like seasonal trends or policy changes.	Pending

Exercise Reminder

Test Case ID:	TC7	**Test Case name:**	Exercise Reminder
Test Case Description:	The patient receives an exercise reminder notification on time		
Pre-Condition:	The patient must be logged into their account. The exercise reminder feature must be enabled. The system must have an active internet connection (if cloud-based). Notifications must be enabled on the patient's device.	**Post Condition:**	The patient receives an exercise reminder notification on time.

Step No.	Action	Expected Output	Actual Output	Comments	Status (Pass/Fail)
1	Log in as a patient.	Patient successfully logs in.		Ensure valid credentials are used.	Pending
2	Set an exercise reminder with a specific time.	The system saves the reminder successfully.		Verify that the reminder is stored in the system.	Pending
3	Wait until the scheduled reminder time.	System triggers the reminder notification.		Ensure no delays in triggering the reminder.	Pending
4	Check if the reminder notification is received.	Patient receives a notification via app, SMS, or email.		Verify that the notification method works as expected.	Pending

RECOVER SMART
YOUR CARE. YOUR GUIDE.
RECOVER SMART BY YOUR SIDE

Receive Medication Reminder

Test Case ID:	TC6	**Test Case name:**	Receive Medication Reminder
Test Case Description:	This test case verifies that patients receive medication reminders at the scheduled time.		
Pre-Condition:	The patient must be logged into their account. The device must have notifications enabled. The system must have an active internet connection (if cloud-based).	**Post Condition:**	The patient receives the medication reminder notification on time.

Step No.	Action	Expected Output	Actual Output	Comments	Status (Pass/Fail)
1	Log in as a patient.	Patient successfully logs in.		Ensure valid credentials are used.	Pending
2	Set a medication reminder with a specific time.	The system saves the reminder successfully.		Verify that the reminder is stored in the system.	Pending
3	Wait until the scheduled reminder time.	System triggers the reminder notification.		Ensure no delays in triggering the reminder.	Pending
4	Check if the reminder notification is received.	Patient receives a notification via app, SMS, or email.		Verify that the notification method works as expected.	Pending

View Monitoring Data

Test Case ID:	TC5	**Test Case name:**	View Monitoring Data
Test Case Description:	This test case verifies that doctors can view patient monitoring data, including wound photos and completed checklists.		
Pre-Condition:	The doctor must be logged into their account. The patient must have uploaded wound photos and submitted checklists. The system must have an active internet connection.	**Post Condition:**	The doctor can successfully view the patient's wound photos and checklists.

Step No.	Action	Expected Output	Actual Output	Comments	Status (Pass/Fail)
1	Log in as a doctor.	Doctor successfully logs in.		Ensure valid credentials are used.	Pending
2	Navigate to the patient monitoring section.	The monitoring section loads without errors.		Verify that the page loads within an acceptable time.	Pending
3	Select a patient.	The patient's monitoring data is displayed.		Ensure that only authorized doctors can access the data.	Pending
4	View the uploaded wound photos and completed checklists.	The wound photos and checklists are displayed without issues.		Ensure that all relevant data is visible and correctly linked to the patient.	Pending

RECOVER SMART
YOUR CARE. YOUR GUIDE.
RECOVER SMART BY YOUR SIDE

Notes

Test Case ID:	TC4	**Test Case name:**	Notes
Test Case Description:	The patient is able to send notes to the doctor through the app and the doctor will reply them instantly		
Pre-Condition:	The patiend and the doctor must be able to log into the system	**Post Condition:**	The doctor can be seen the notes of specific patient

Step No.	Action	Expected Output	Actual Output	Comments	Status (Pass/Fail)
1	Verify the Notes page loads correctly	Open the Notes page	Open the Notes page	The page should load without errors, displaying all elements properly	Completed
2	Verify that the doctor's name is displayed correctly	Open the page and check the top section	Open the page and check the top section	The doctor's name should be displayed properly	Completed
3	Verify that messages/notes are displayed in the correct order	Observe the arrangement of messages	Observe the arrangement of messages	Notes should appear in chronological order with timestamps	Completed
4	Verify that the chat bubble design is clear and distinguishable	Observe sent and received messages	Observe sent and received messages	User and doctor messages should have different styles/colors for clarity	Completed
5	Verify that new notes/messages can be typed	Tap on the text input field	Tap on the text input field	The keyboard should appear, and the user should be able to type a message	Completed
6	Verify that the send button is enabled when text is entered	Type a message in the text field	Type a message in the text field	The send button should become active when text is entered	Completed
7	Verify that a message is sent successfully	Type a message and press send	Type a message and press send	The message should appear in the chat with the correct timestamp	Completed
8	Verify that typing status is displayed when the doctor is typing	Wait for the doctor to start typing	Wait for the doctor to start typing	A "X is typing..." indicator should appear	Completed
9	Verify that messages are timestamped correctly	Send/receive a message and check the timestamp	Send/receive a message and check the timestamp	The timestamp should be accurate based on the current time	Completed
10	Verify that the page scrolls properly when more messages are added	Add multiple messages until they exceed the screen limit	Add multiple messages until they exceed the screen limit	The page should allow smooth scrolling without lag	Completed
11	Verify that old messages remain visible after app reload	Send a message, then refresh the app	Send a message, then refresh the app	The message history should remain intact after reloading	Completed
12	Verify that blank messages cannot be sent	Try sending an empty message	Try sending an empty message	The send button should remain disabled for empty input	Completed
13	Verify that long messages wrap properly within the chat bubble	Send a long message	Send a long message	The text should wrap inside the chat bubble without getting cut off	Completed
14	Verify that the input field clears after sending a message	Send a message	Send a message	The input field should clear automatically after sending	Completed
15	Verify that the back button navigates correctly	Press the back button	Press the back button	The app should navigate back to the previous screen without errors	Completed

RECOVER SMART
YOUR CARE. YOUR GUIDE.
RECOVER SMART BY YOUR SIDE

Upload Records

Test Case ID:	TC3	**Test Case name:**	Upload Wound Photo
Test Case Description:	This test case verifies that patients can successfully upload records successfully.		
Pre-Condition:	The patient must be logged in. The system must have an active internet connection. The upload feature must be enabled.	**Post Condition:**	The uploaded document is stored and accessible to the doctor.

Step No.	Action	Expected Output	Actual Output	Comments	Status (Pass/Fail)
1	Log in as a patient.	Patient successfully logs in.	Patient successfully logs in.	Ensure correct credentials are used.	Completed
2	Navigate to the upload records section when click on the "Take Prescribed Medication" milestone.	The upload page loads without errors.	The upload page loads without errors.	Verify that the page loads within acceptable time limits.	Completed
3	Select a valid file format (.jpg, .png, .pdf) to upload.	The selected file appears in the preview (if applicable).	The selected file appears in the preview (if applicable).	Ensure file format and size meet system requirements.	Completed
4	Click the "Upload" button.	The system processes the upload and confirms success.	The system processes the upload and confirms success.	Check for any error messages during upload.	Completed
5	Confirm that the file is displayed correctly in the patient's records.	The uploaded photo is visible in the system.	The uploaded photo is visible in the system.	Ensure the photo is not distorted or missing.	Completed
6	Navigate to the special notes text box and type the current condition of the patient	Patient is able to insert special notes about his/her condition.	Patient is able to insert special notes about his/her condition.	Ensure that the special notes text box is enabled	Completed
7	Log in as a doctor and access the patient's record documents.	The doctor can view the uploaded document and special notes.	The doctor can view the uploaded document and special notes.	Verify that only authorized doctors can access the photo	Completed