**Deliverable3:**

**Requirements**

**List a set of requirements designated for phase 1.**

**1.1 User Management**  
In our Django project, user management is built upon Django’s robust authentication framework. Users can securely register and log in using forms that enforce strong password policies and proper validations. Once registered, users can update their profiles, including personal details, medical history, and dietary preferences. Before the system suggests any diet plan, it ensures that users record any allergies, which are stored as part of their profile data. Role-based access is implemented through Django’s groups and permissions, enabling fine-grained control over what patients, doctors, and administrators can view or modify within the system. This approach not only enhances security but also streamlines the user experience by presenting tailored functionalities based on user roles.

**1.2 AI-Powered Health Risk Analysis**  
The health risk analysis feature leverages AI algorithms integrated into the Django application to analyze patient health data. By collecting data from user profiles and health records, the system runs background processes—potentially using Celery for asynchronous tasks—to predict potential health risks. The machine learning models, which could be developed with libraries like TensorFlow or scikit-learn, are seamlessly integrated with Django’s view and model layers. The results are then presented to both patients and doctors through secure dashboards, where actionable recommendations for lifestyle changes and preventive care are clearly outlined.

**1.3 Personalized Recommendations**  
Personalization is a core aspect of the application. Based on individual health data and recorded allergies, the AI component generates diet plans that are both nutritious and safe. The system also considers each user's physical activity and personal goals to suggest appropriate exercises and lifestyle adjustments. This process involves complex data analysis that runs in the background and updates recommendations dynamically. Django’s ORM efficiently manages these interactions with the database, ensuring that all recommendations are accurately tied to the most recent user inputs and health records.

**1.4 Appointment Scheduling**  
Appointment scheduling in the Django project is handled by a dedicated module that allows patients to book, reschedule, or cancel appointments through a user-friendly interface. The scheduling system integrates with Django’s models to store appointment data and uses Django’s messaging framework to provide real-time notifications to doctors when new appointments are booked or existing ones are modified. This seamless interaction between patients and doctors ensures that both parties have up-to-date information on appointment statuses, which is critical for effective healthcare delivery.

**1.5 Real-Time Alerts and Notifications**  
To keep users informed and proactive about their health, the system incorporates real-time alerts and notifications. Health alerts and medication reminders are generated based on the patient’s medical records and treatment plans. When the AI analysis detects a critical health risk, immediate notifications are sent to the appropriate doctors. This functionality is implemented using Django Channels for real-time web socket communication, ensuring that alerts are delivered promptly. Additionally, Django’s built-in scheduling and background task management frameworks work together to handle periodic notifications and reminders.

**1.6 Electronic Health Records (EHR) Management**  
The management of electronic health records (EHR) is designed with a strong emphasis on security and accessibility. All medical records are stored securely in the database using Django models, with access controlled by stringent permissions that ensure only authorized personnel can update or view sensitive data. Data encryption and regular audits are part of the overall strategy to maintain patient confidentiality and comply with healthcare regulations. The Django admin interface is further customized to support efficient record-keeping and quick retrieval of medical histories during patient consultations.

**1.7 Payment and Billing**  
Payment and billing features are integrated into the system to allow patients to view and pay their bills online securely. The Django application communicates with third-party payment gateways (such as Stripe or PayPal) using RESTful APIs to process transactions. All payment data is transmitted over secure protocols, and sensitive information is handled in accordance with best practices for financial security. Detailed billing records are maintained within Django’s database, allowing both patients and administrators to track payments, refunds, and outstanding balances with ease.

**1.8 Admin Panel for Hospital Management**  
An extensive admin panel is provided to facilitate hospital management tasks. Built on Django’s powerful admin interface, this panel allows system administrators to manage users, doctors, appointments, and generate various reports. In addition to user management, the admin panel includes functionalities for monitoring system logs and ensuring security compliance. Custom views and dashboards have been developed to present aggregated data and insights, enabling administrators to make informed decisions about resource allocation and operational efficiency.

**1.9 Data Management and Retention**  
Data management is automated within the Django project to ensure that inactive user data is handled appropriately. A scheduled task or management command routinely checks for user accounts that have been inactive for 90 days and clears their data from the database. This approach not only helps in maintaining data hygiene but also complies with data retention policies. The process is designed to run in the background without disrupting the user experience, ensuring that only active and relevant data occupies the system’s resources.

**1.10 Security Features: Two-Factor Authentication (2FA)**  
Security is a cornerstone of the project, and two-factor authentication (2FA) is implemented to provide an extra layer of protection during login. Integrating 2FA within Django can involve using third-party packages or custom middleware to generate time-based one-time passwords (TOTPs) or SMS/email codes that users must verify. This feature significantly reduces the risk of unauthorized access by ensuring that even if a password is compromised, additional verification is required before access is granted. The entire authentication flow is seamlessly integrated into Django’s existing authentication system, reinforcing the security posture of the application.

**1.11 Medication Management & Reminders**

To support adherence to treatment plans, the system can incorporate a dedicated medication management module. Patients can log their prescriptions, dosages, and intake schedules via user-friendly forms. Django’s background task management (using Celery or Django’s built-in scheduling tools) can trigger timely reminders and notifications to ensure that medications are taken as prescribed. Additionally, this module can track historical medication data, offering insights into patient compliance which can be reviewed by healthcare providers.

**1.12 Health Analytics and Visualization Dashboard**

A comprehensive analytics dashboard can be added for both doctors and administrators to gain insights into patient health trends. By leveraging Django’s ORM to gather data from user profiles, health records, and AI-powered risk assessments, the dashboard can display interactive charts and graphs (using libraries such as Chart.js or D3.js). This visualization layer aids in monitoring treatment outcomes, resource allocation, and overall system performance, enabling data-driven decisions that improve patient care.

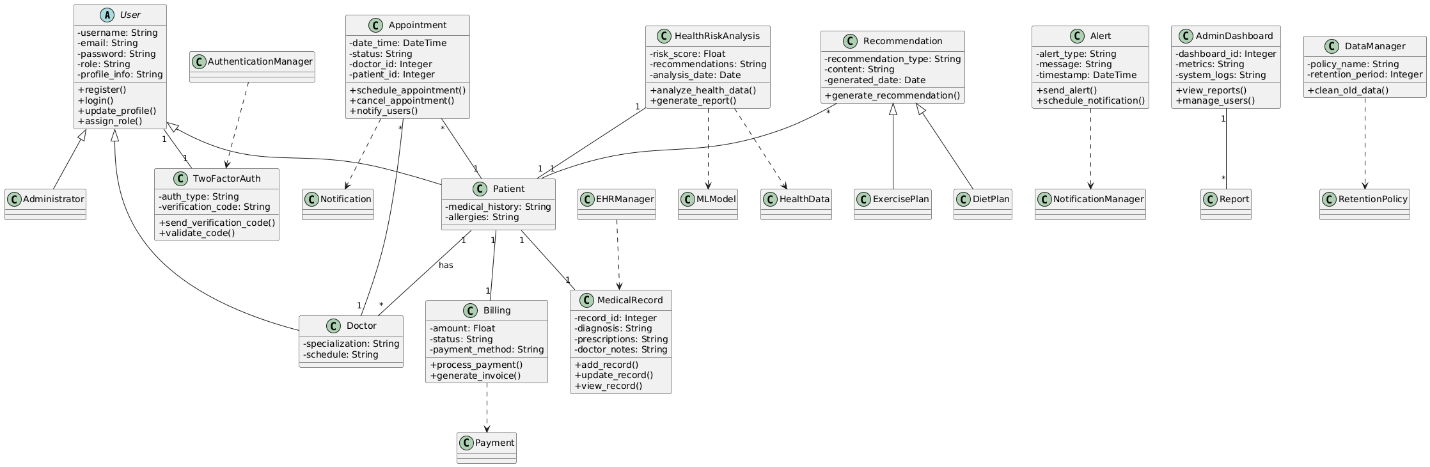
**1.13 Patient Feedback and Survey Module**

In order to continuously improve service quality and patient satisfaction, a feedback module can be integrated into the system. After appointments or specific interactions, patients can be prompted to complete short surveys. This module utilizes Django forms and models to capture ratings, comments, and suggestions. The collected data is then aggregated and presented to healthcare providers and administrators in real time, allowing them to make necessary adjustments to the care process and overall patient experience.

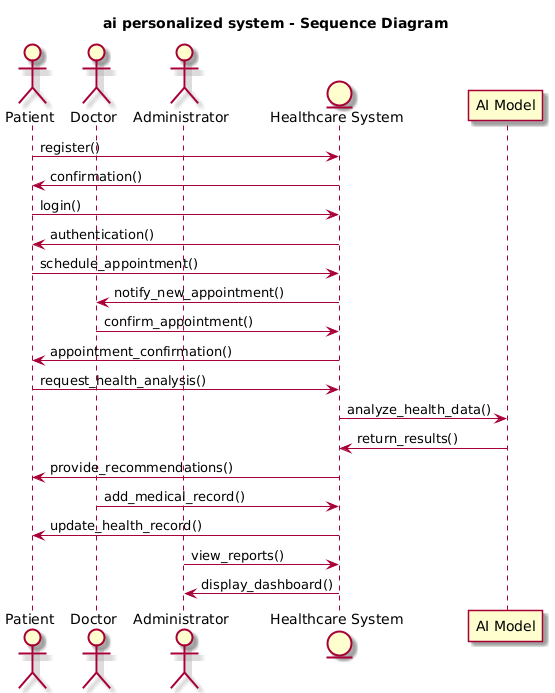
Each of these features has been carefully developed and integrated within the Django framework to ensure a robust, secure, and user-centric healthcare management system.

b. UML design for phase 1.

Class diagram



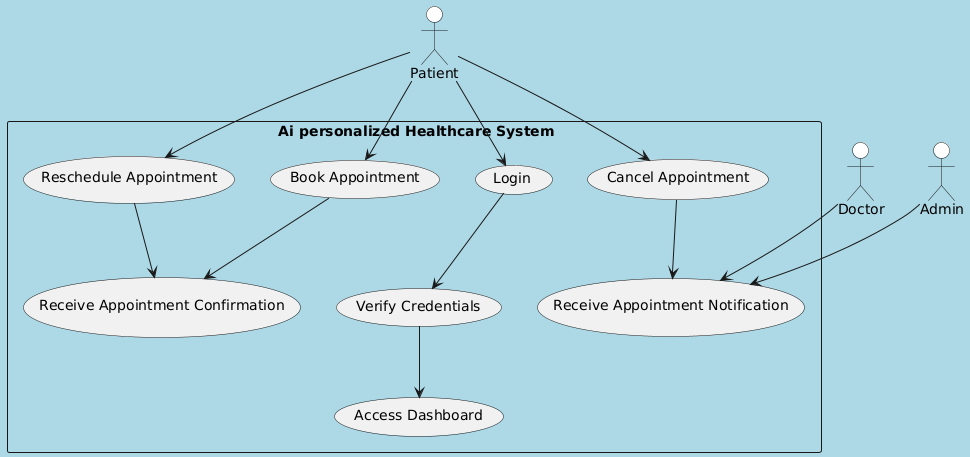
**Sequence diagrams**



This sequence diagram shows the sequence of events.

**Use case diagrams**

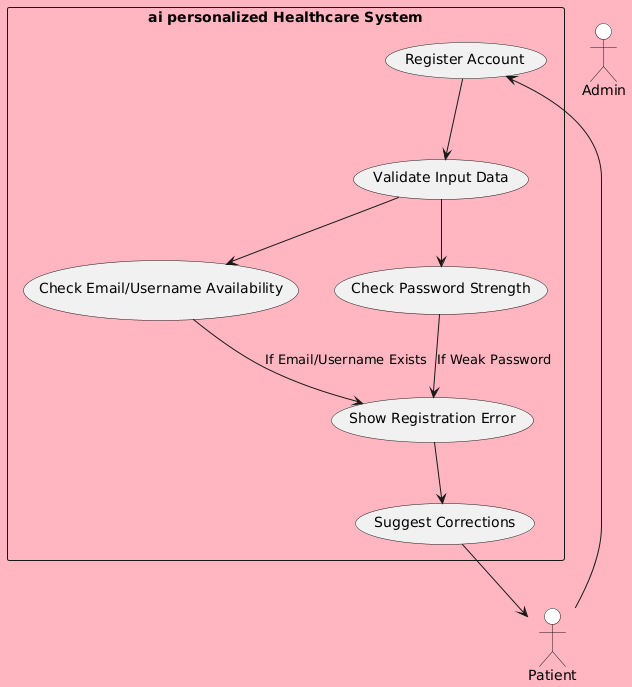
Successful creating appointments



This use case diagram show a successful user login and after that a successful creation of appointments, reschedule them , cancel and also receiving a successful email confirmation of their appointment is created , also the client will receive an email to tell them that their appointment is cancelled

Failed to register patient

This use case diagram shows how system behave when client to register on the system using wrong credentials that is all of the inputs required. Once that scenarios occurs then the user is required to use correct details again so that the system authorize them to register.



**Test cases**

from django.test import TestCase

from django.contrib.auth.models import User

class LoginTestCase(TestCase):

def setUp(self):

"""Set up test user before running tests."""

self.username = 'martin234'

self.password = '1234'

self.user = User.objects.create\_user(username=self.username, password=self.password)

def test\_login\_success(self):

# **TEST 1: Successful Login Test**

# This test verifies that a user with valid credentials can log in successfully.

# It submits a POST request to the '/view\_patient/' endpoint with the correct username and password.

# On a successful login, Django typically redirects the user, indicated by a 302 HTTP status code.

# The test asserts that the response status code is exactly 302, confirming a successful login.

response = self.client.post('/view\_patient/', {'uname': self.username, 'pwd': self.password})

self.assertEqual(response.status\_code, 302)

def test\_login\_failure\_invalid\_password(self):

**# TEST 2: Login Failure due to Invalid Password**

# This test checks that a valid username with an incorrect password does not allow a user to log in.

# It sends a POST request to the '/login/' endpoint with the valid username but an invalid password.

# The expected behavior is that the login form is re-rendered and the response returns a 200 status code.

# The test asserts that the response status code is 200, confirming that login did not succeed.

response = self.client.post('/login/', {'uname': self.username, 'pwd': 'wrongpassword'})

self.assertEqual(response.status\_code, 200)

def test\_login\_failure\_invalid\_username(self):

**# TEST 3: Login Failure due to Invalid Username**

# This test ensures that a login attempt with a non-existent username fails.

# It submits a POST request to the '/login/' endpoint using a username that is not registered.

# The login form is expected to be re-rendered, and the server should return a 200 HTTP status code.

# The test confirms this behavior by asserting that the response status code is 200.

response = self.client.post('/login/', {'uname': 'wronguser', 'pwd': self.password})

self.assertEqual(response.status\_code, 200)

def test\_login\_required\_redirect(self):

**# TEST 4: Login Required Redirect Test**

# This test checks that an unauthenticated user trying to access a protected page is redirected to the login page.

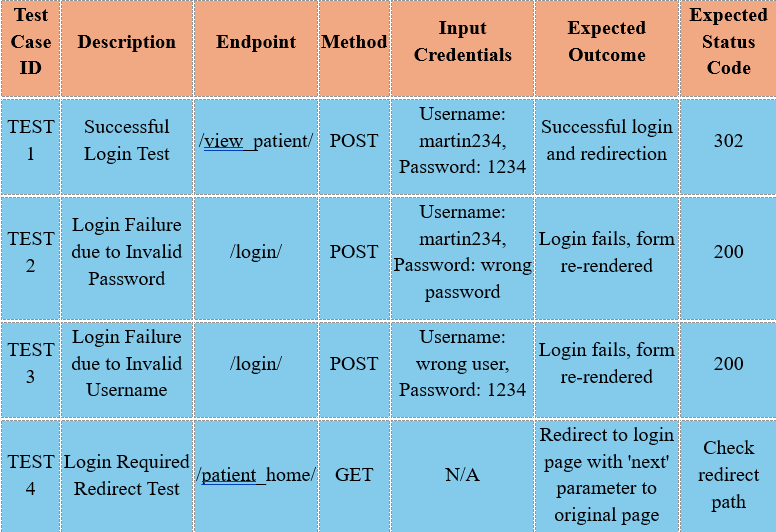
# It makes a GET request to the '/patient\_home/' endpoint, which requires the user to be logged in.

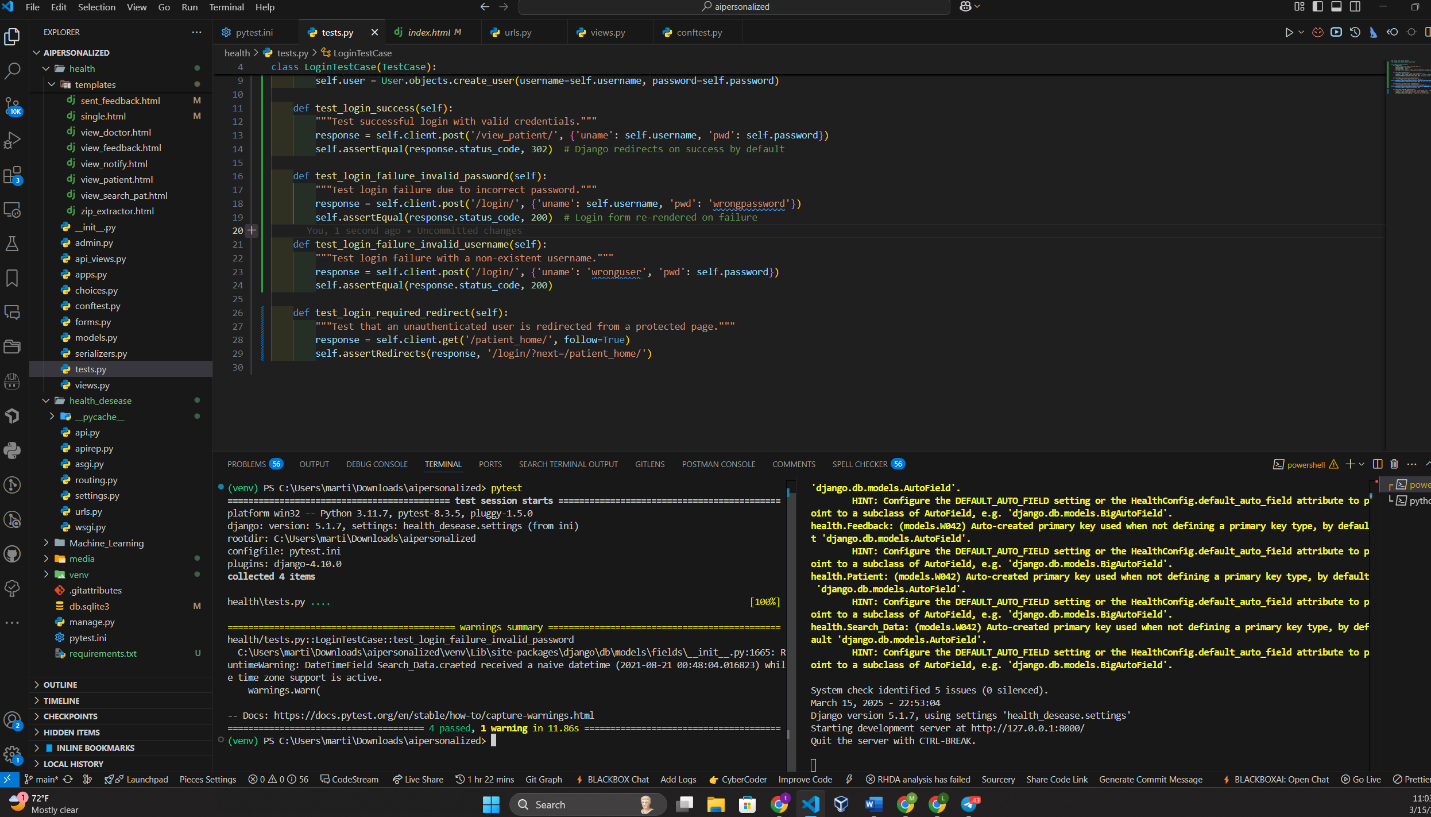
# The expected behavior is a redirection to the login page with the original page specified as a 'next' parameter.

# The test asserts that the redirect is correctly performed to '/login/?next=/patient\_home/'.

response = self.client.get('/patient\_home/', follow=True)

self.assertRedirects(response, '/login/?next=/patient\_home/')





**User manuals**

For this project after you have opened the file, on your favorite IDE navigate using this command on terminal to the folder contain this code

*cd aipersonalized*

Once on this file create a virtual environment using this command on terminal

*python -m venv venv*

After you have create the virtual environment now you need to activate it, using this command

*venv/Scripts/activate*

Once activated you now proceed to install all the modules that have been used on this project using this command

*pip install -r requirements.txt*

Once all the modules are installed you will run this command to start the server

*python manage.py runserver*

Your project will run successful

**Compile and run**

Once activated you now proceed to install all the modules that have been used on this project using this command

*pip install -r requirements.txt*

Once all the modules are installed you will run this command to start the server

*python manage.py runserver*

Your project will be successful

Running test case

On your terminal open another window. Activate the virtual environment with this command

*venv/Scripts/activate*

After activation run this command to run the test.

*pytest*

**Peer review**

One of our peers suggested that we incorporate a module into the system that enables users to change their passwords. In this module, the user first enters their current password, then creates and confirms a new password to ensure both entries match. Once the new password is verified, it is updated and stored in the database. I am pleased to report that we have successfully implemented this module, and it is functioning as intended.

**Reflection**

We were able to

1. creating login and registration form

2. creating prediction models and training data

3. integrating and testing prediction models on the system

4. creating an admin dashboard and giving them roles

5. creating and integrating SQLite database to the system

6. creating and giving different permissions to different users and defining their purpose

7. creating a module that allows update client details

8. creating a reset password module

**Member contribution**

|  |  |  |  |
| --- | --- | --- | --- |
| Member Name | Contribution Description | Overall Contribution (%) | Note (if applicable) |
| Rohit Ibrahimpatnam | Led the overall system design, structured the phased development plan, and outlined key system components. | 14% | Ensured the report aligned with system objectives. |
| Anusha Pujari | Coordinated report structure, project scope definition, and risk management. | 14% | Assisted in organizing the implementation timeline. |
| Harshini Sai Sangadi | Developed system architecture, backend framework, and API integrations. | 14% | Project Lead: Provided insights into backend infrastructure and security. |
| Ram Gopal Anne | Authored AI-driven health recommendations, machine learning integration, and dataset analysis. | 14% | Focused on AI functionality and data modeling. |
| Rehamath Shaik | Wrote UI/UX design specifications, frontend requirements, and user interface functionality. | 14% | Led frontend design documentation and usability considerations. |
| Sai Murali Kilaru | Handled database design, cloud deployment, scalability strategies, and performance optimizations. | 14% | Focused on cloud infrastructure and system efficiency. |
| Vinay Kumar Parvathini | Contributed to testing strategies, quality assurance, and security testing documentation. | 14% | Led QA and ensured system robustness. |