NIT6150

Advanced Project

System Analysis and Design Report

< Online appointment booking system >

Team Leader: Vedant Jadhav (S8115752)

Team Members: Amit Gupta (s8086846)

Lalit Maharjan (s8085161)

Sharmila Timalsina(s8110001)

Table of content

1. System Development Approach

We chose the **Agile Development Approach** for our project because of its adaptability and widespread acceptance in the industry. Agile offers incremental development, ongoing feedback, and flexibility, which are essential for a user-friendly and efficient **Online Appointment Booking System**. **(Highsmith, 2002)**

Justification:

- **Frequent Feedback:** Agile allows us to make improvements based on input from users.
- **Faster Development:** Small, iterative releases ensure quick delivery.

- **Scalability:** New features (e.g., notifications, payments) can be added easily.
- **Industry Standard:** Agile is widely used in the IT industry by software industry. **(Toda et al., 2019)**

---

2. Functional Requirements

Below are the essential functional requirements of the system:

User Functionalities:

- Users can register and log in (if required).
- Users are able to see open appointment times.
- Users can book an appointment.
- Users receive booking confirmation.

Admin Functionalities:

- Admin can manage bookings (view, update, delete).
- Admin can update available slots.
- Admin can generate reports on bookings.

a. Data Requirements and Diagrams

ER Diagram (Entity-Relationship Diagram)

The system includes the following entities:

- **Users** (UserID, Name, Email, Phone, Password)
- **Appointments** (AppointmentID, UserID, ServiceType, Date, Time, Status)
- **Services** (ServiceID, ServiceName, Duration, Price) **(Légora et al., 2020)**

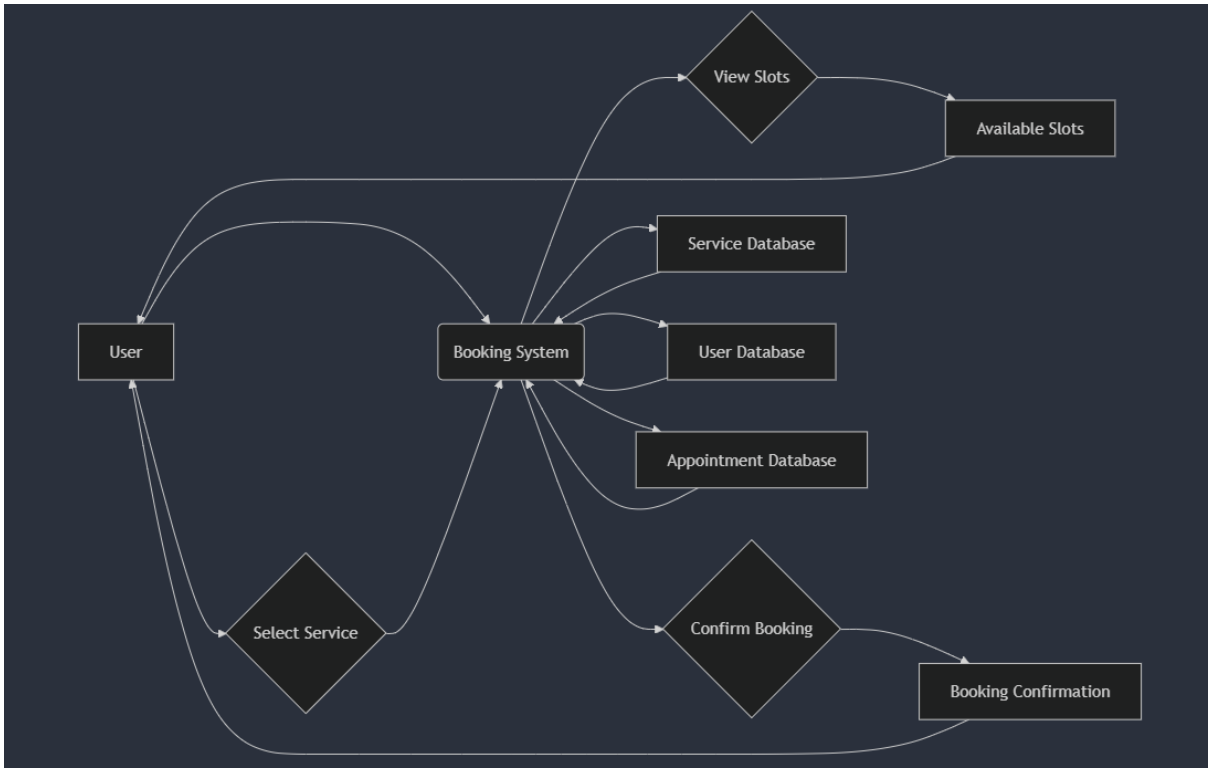
Data Flow Diagram (DFD)

- **Level 0 DFD:** Shows a high-level view of the system, including user interactions.
- **Level 1 DFD:** Details booking processes (viewing slots, selecting service, confirming booking).

Level 0 Diagram with the description:

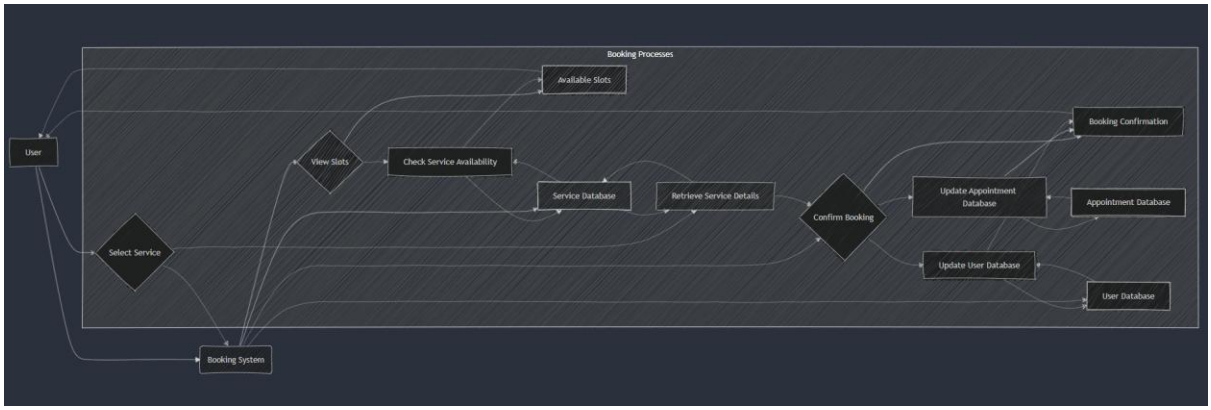
- User: Denotes the external entity engaging with the system.
- Booking System: Represents the overall system.
- View Slots: Denotes the procedure for checking accessible appointment slots.
- Available Slots: Represents the data flow of available or accessible slots to the user.
- Select Service: Denotes the process of choosing/selecting a service.
- Confirm Booking: Denotes the process of confirming or validating the booking.
- Booking Confirmation: Represents the data flow of booking confirmation or validation to the user.
- Service Database: Denotes the data store for service information.

- User Database: Denotes the data store for user information.
- Appointment Database: Denotes the data store for appointment information.



Level 1 Diagram with description (Booking Processes):

- This level builds upon the "Booking System" procedure from Level 0.
- Check Service Availability: Checks if the selected service is available at the requested time.
- Retrieve Service Details: Fetches or acquires detailed information about the selected service or desired time.
- Update Appointment Database: Saves the confirmed appointment information.
- Update User Database: Updates or records the user's history with the appointment.
- The level one diagram also shows the flow of data to and from each of the databases.
- The level one diagram is included within a subgraph name "Booking Processes" to show that these processes are part of the broader booking system. **(Zakaria et al., 2019)**



b. Database Design (Data Types and Sizes)

The system uses **CSV files** (or SQLite) for data storage.

Table: Users

Column Name	Data Type	Size	Description
UserID	INTEGER	-	Unique ID for users
Name	TEXT	100	User's full name
Email	TEXT	255	User's email
Phone	TEXT	15	Contact number
Password	TEXT	255	Hashed password

Table: Appointments

Column Name	Data Type	Size	Description
AppointmentID	INTEGER	-	Unique ID for booking
UserID	INTEGER	-	Links to Users table
ServiceType	TEXT	100	Type of service booked
Date	DATE	-	Appointment date
Time	TIME	-	Appointment time
Status	TEXT	50	(Pending, Confirmed, Cancelled)

Table: Services

Column Name	Data Type	Size	Description
ServiceID	INTEGER	-	Unique ID for service
ServiceName	TEXT	100	Name of service
Duration	INTEGER	-	Time required in minutes
Price	REAL	-	Service cost

3. Use case diagram

Use case diagram provides clear overview of the booking, focusing on the different steps that involves in booking any service. This helps ensuring that all the stakeholders (users, admin and developer) have the clear understanding of the booking process.

This use case diagram represents the appointment booking system with three main actors.

User

Admin

System

Use cases:

1. **User (Customer)-**

Signs up/Signs in

User must verify their credentials (email/password) while signing in. If login fails, system will extend to show a login error.

Search and select service

Users can browse and choose the service they want the booking for.

Check date and time slots:

User can see the available appointment slots.

Make appointment –

User can book an appointment

Reschedule appointment –

If user wants to change an existing booking, they can reschedule instead of booking a new one.

Cancel appointment –

If needed, users can cancel their appointment anytime they want.

2. **Admin –**

Manage availability –

Admin can add, modify and remove the time slots according to the need to manage the schedule.

View appointments –

Admin can see all scheduled appointments.

Approve/Reject appointment –

After viewing appointments, admin has the option to either approve or reject them.

3. **System –**

Authenticate users –

System will ensure security and correct user authentication.

Suggest best date/time slots-

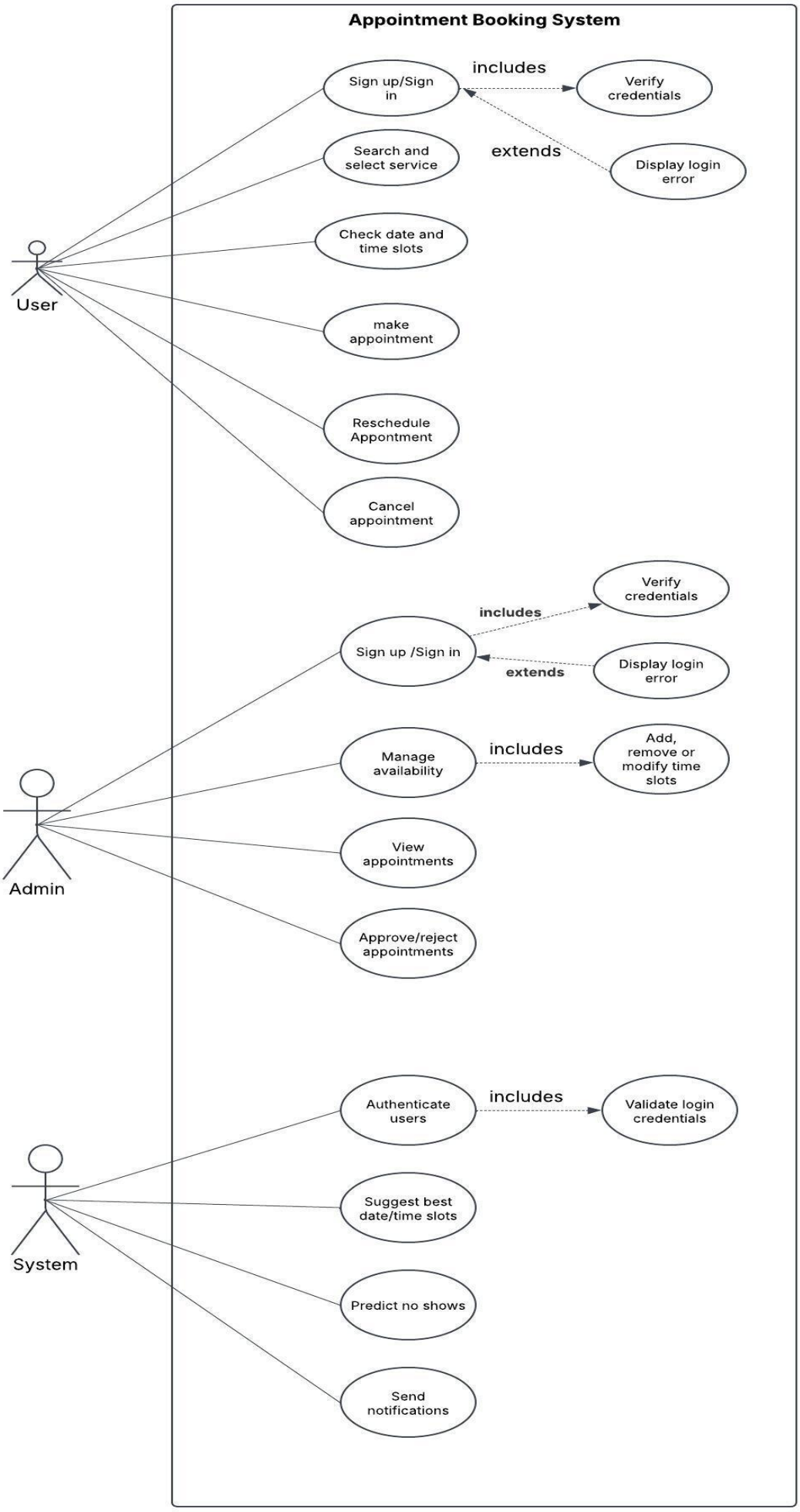
The system analyses past bookings and suggests the most preferrable time slots.

Predict no shows –

The system can anticipate potential cancellations or no shows.

Send notifications –

The system sends alerts about upcoming appointment cancellation and rescheduling updates.


Appointment Booking System
Sign up/Sign in
includes
Verify credentials
extends
Display login error
Search and select service
Check date and time slots
make appointment
Reschedule Appontment
Cancel appointment
User
Verify credentials
includes
Sign up /Sign in
extends
Display login error
Manage availability
includes
Add, remove or modify time slots
View appointments
Approve/reject appointments
Admin
Authenticate users
includes
Validate login credentials
Suggest best date/time slots
Predict no shows
Send notifications
System

4. Non-Functional Requirements:

Look and feel requirements – The system should be responsive to ensure seamless user experience. It should respond quickly across various browsers as well as operating systems.

Usability and humanity requirements – The system should be intuitive in a way that users would be able to use the system by themselves without any instructions.

Performance requirements – The system should be capable of handling at least 500 active users without affecting the performance of the system.

Operational requirements – The system must be available almost all the time. Backup should be done regularly to make sure no data is lost even if something goes wrong.

Maintainability and support – The system should be regularly updated and be able to fixate the bug.

Security requirements – User's data must be protected. Multi factor authentication should be enforced for admins. Regular conduction of security audits and penetration testing must be done to find out if there are any vulnerabilities.

Cultural and political requirements – System should support multiple languages to accommodate diverse user bases. Time zones and date formats should be configurable.

Legal requirements – System must comply with GDPR, Australian Privacy act and other data protection regulations. Privacy policy should be implemented, and user's records must be kept in an encrypted format for their security. **(Booch et al., 1998)**

5. Navigation and user interface design –

User flow:

A. Customer navigation flow-

Sign up/ sign in

Users load the sign in/sign up page.

If user is new, they can sign up providing email and password.

If user already has an account, they can sign in and verify their credentials.

If login fails, error message will pop up.

Search and select service

Users can search for available services on the homepage/dashboard and select the one that they prefer.

Check date and time slots

After the service is selected, users can see the available appointment slots.

Make an appointment

Users can make the booking after which confirmation screen appears with the booking details.

Reschedule appointment

Users can view their appointment history and choose an existing appointment to reschedule.

Cancel appointment

Users can cancel the booked appointment too if they need.

B. Business Admin

Login and Dashboard access -

Admin can sign up/sign in using their credentials.
They have access to the admin dashboard.

Manage availability
Admin can add, remove and modify time slots.

View appointments
All scheduled appointments are displayed on dashboard, which can be viewed by admin.

Approve/Reject appointments
After viewing the appointment, admin can approve or reject appointments.

C. System

Authenticate users
System verifies login credentials.

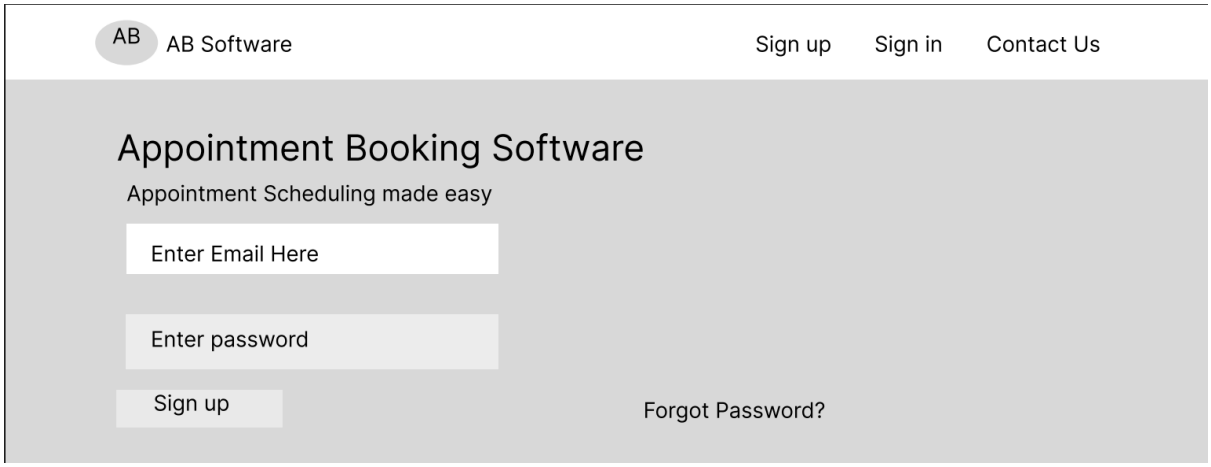
Suggest best date/time slots
System highlights optimal time slots based on previous data.
Predict no shows
The system detects potential no shows based on user's past booking history.

Send notification
System sends SMS or emails about appointment confirmation, or anything related to the appointment booked.

User Interface Design
Sign up/Sign in –

- There will be fields for entering email and password.

- There will be forgot password option as well which will lead to reset password and log in again into the account.



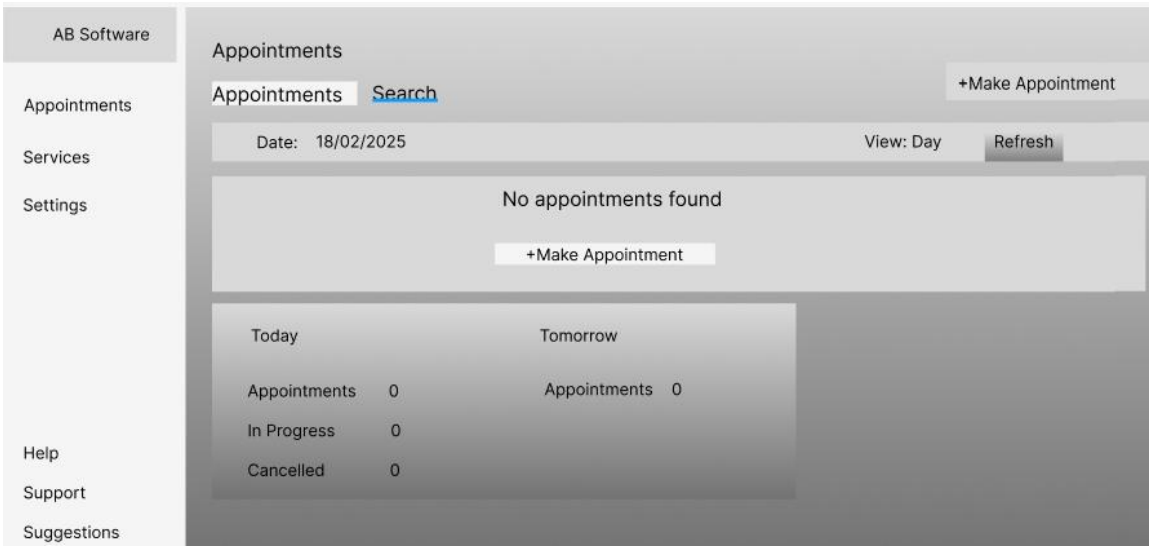
User dashboard –

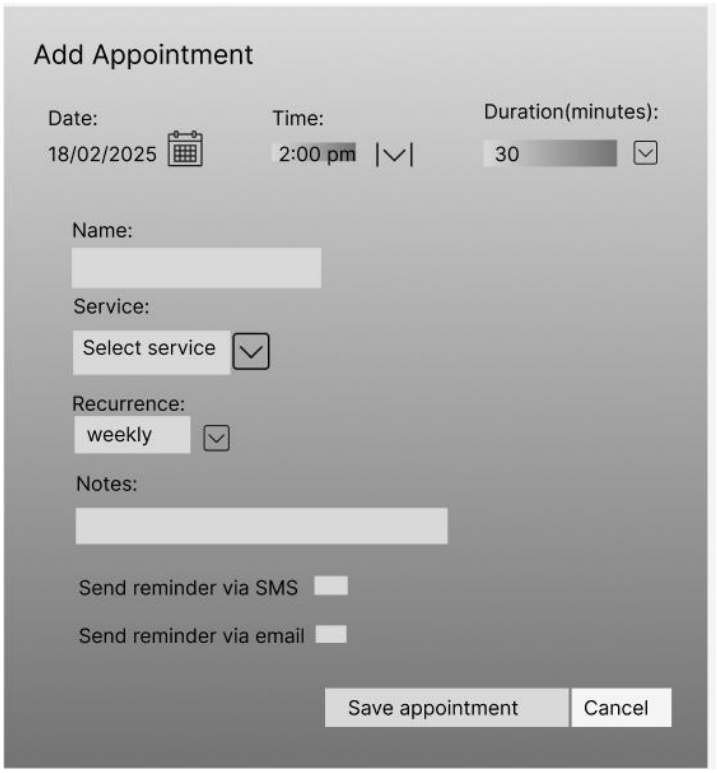
- Displays list of upcoming appointments if there is any.
- If not, there will be a button that says make appointment.

Appointment Booking page -

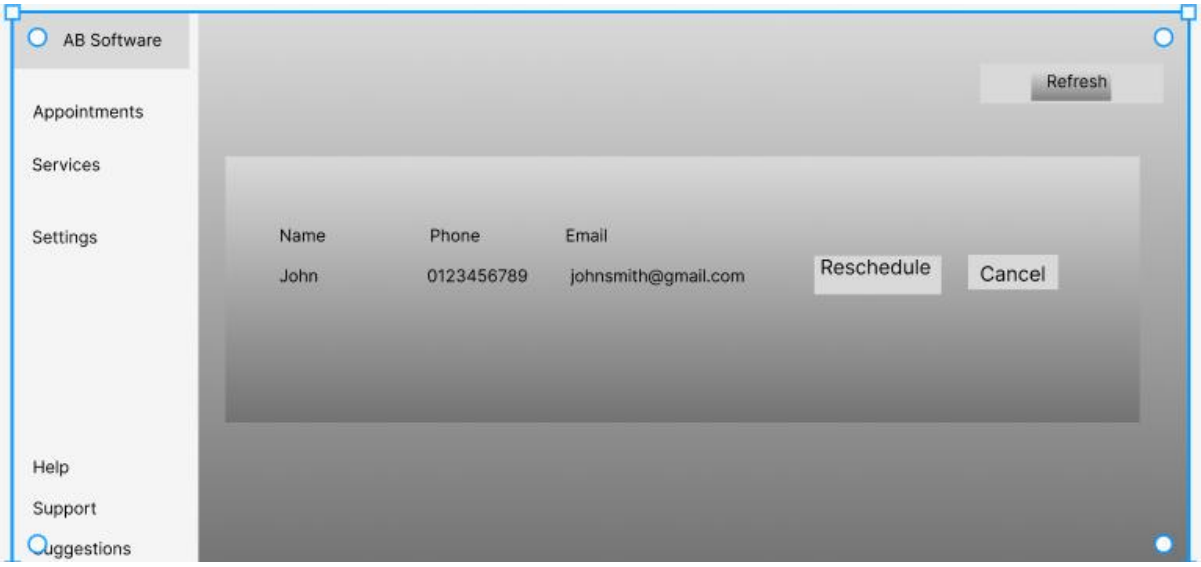
- Has service selection dropdown
- Calendar view to pick up date
- Available time slots displayed
- Confirmation message after booking. **(Zhao et al., 2017b)**

- User can also choose how they would like to receive reminders for the booking.





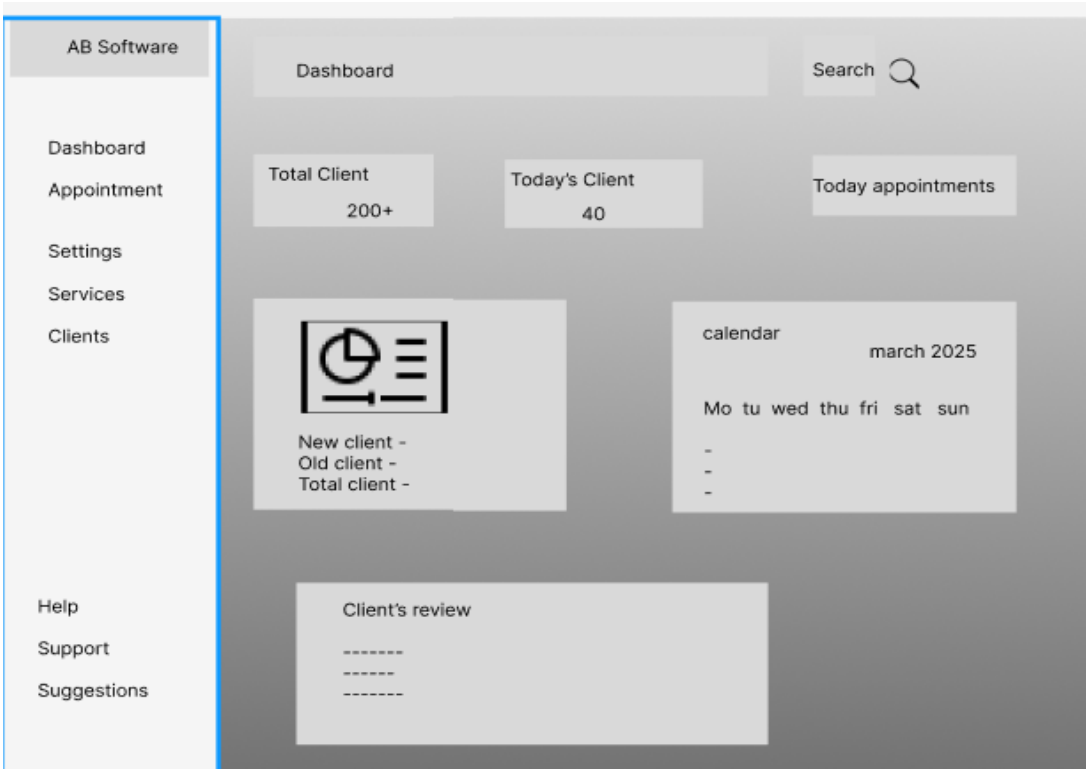
- After the booking, user can review their upcoming appointments, they will always have the option to reschedule and cancel.
- If they reschedule, they will be redirected to the booking page and to cancel it, they can simply click the cancel button. **(Zhao et al., 2017)**

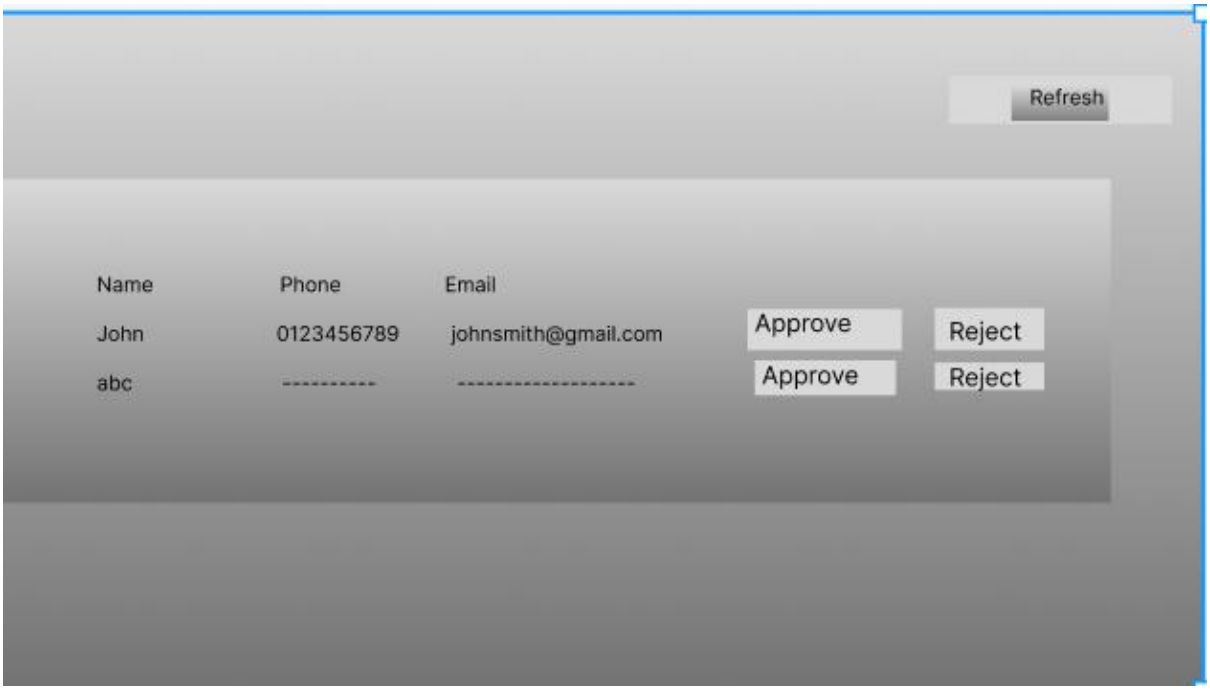


Admin Dashboard –

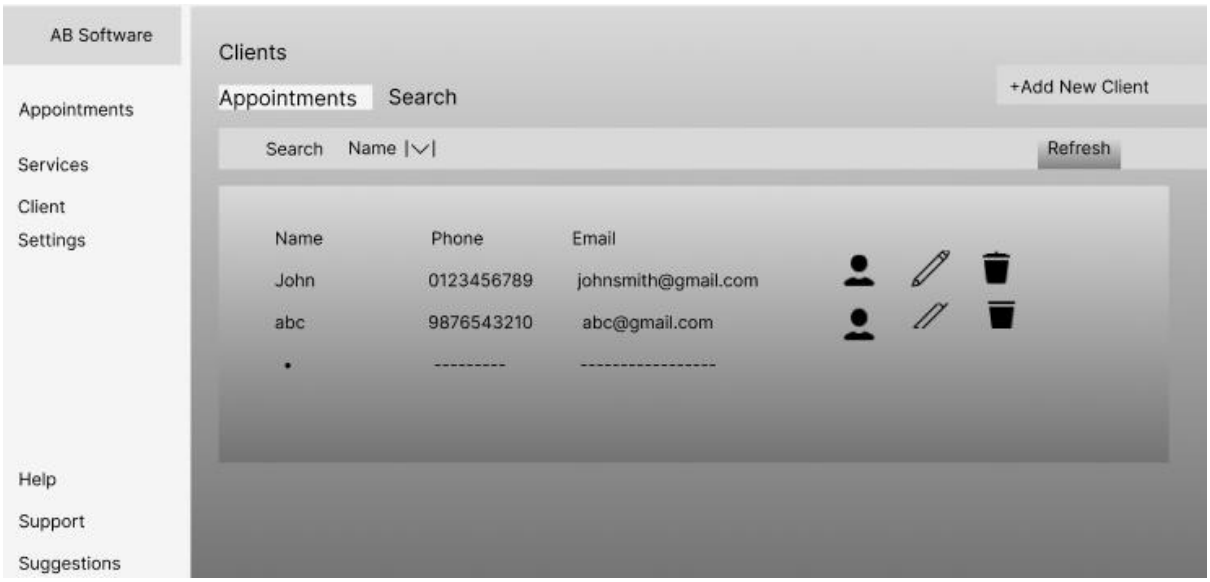
- Admin can view all the booked appointments.

- **Total clients** – Displays number of clients till date.
- **Today's clients** – Displays number of clients for the day.
- **Today's appointment** – Likely shows scheduled bookings for the day.
- **Calendar** – Displays upcoming appointments or available slots.
- Admin can view and manage clients feedback too.





- Admin can view all the scheduled appointments.
- Also, they can view all the client's information including contact details and booking history.
- They can edit and delete client's information if necessary.
- Admin can use approve or reject button to either approve or reject the appointment based on the availability or other factors. **(for, 2025)**



6. Estimated cost breakdown

Cost Category	Description	Estimated Cost
Development Costs	Includes coding, debugging, and implementation by team members	$0
Hosting & Domain	Server for deployment, domain registration	$0
Database Management	Database setup and management	$0
UI/UX Design	Designing user interface and wireframes	$0
Security & Compliance	Implementing basic security measures like SSL and access control	$0
Testing & QA	Testing the system for functionality, security, and performance	$0
Documentation	user guides, and training materials	$0

7) Work Breakdown Structure (WBS)

1.1 Launch of the Project
Specify the goals and scope of the project.

Determine the team's roles and stakeholders.

Make a preliminary risk assessment.

1.2 Conditions Getting Together
Compile the requirements, including functional and non-functional.

Make user stories and use case diagrams.

Complete the system architecture.

1.3 Create the database schema (data flow diagrams, ER diagrams).

Make UI/UX designs and wireframes.

1.4 Progress
Create a development environment.

Create the essential modules:

User verification (login, registration).

Reservation, cancellation, and rescheduling of appointments.

administrative features (create reports, schedule appointments).

Include third-party services (notifications, payment gateway, etc.).

1.5 Testing Individual module unit testing.

System component integration testing.

testing user acceptance (UAT) with interested parties.

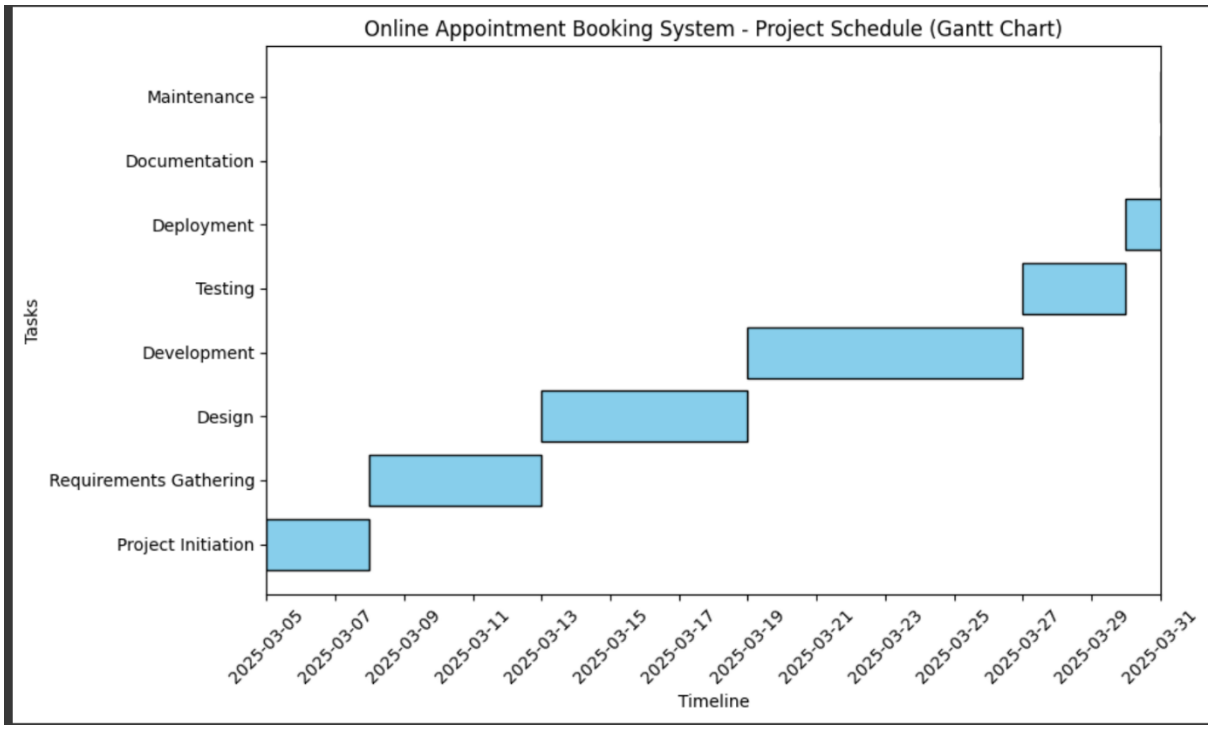
1.6 Implementation
Create the production environment.

Install the program on a live server.

Perform last-minute testing and bug repairs.

Gannt chart

Work breakdown structure and a project schedule (as a Gantt Chart)



Explanation of project schedule

3.1 Project start-up
During this phase, the project's goals, scope, and stakeholders are defined. It guarantees that everyone is on the same page and lays the groundwork for the project.

3.2
The team collects both functional as well as nonfunctional needs in this phase. User stories and use case diagrams are made to make sure the demands of all parties involved are met.

3.3
The design step entails developing the system architecture, wireframes, and database structure (ER diagrams, DFDs). Before development starts, this stage makes sure the system is thoroughly planned.

3.4
The system's core modules are coded during the longest period, development. This covers

admin features, appointment scheduling, and user authentication. During this stage, third-party services like notifications and payment gateways are also integrated.

3.5
Testing guarantees that the system satisfies the requirements and is error-free. During this stage, user acceptance testing (UAT), integration testing, and unit testing are carried out.

3.6 Deployment
To make sure everything functions as planned in the production setting, the system gets uploaded to a live server and undergoes final testing.

3.7 Documentation
For the project to be successful in the long run, documentation is essential. During this stage, technical documentation, admin manuals, and user manuals are created.

3.8 Continuous Maintenance
The system moves into the maintenance stage following deployment. This entails keeping an eye on performance, addressing bugs, and offering assistance to users. During this phase, additional updates and feature improvements are being planned.

How schedule relates to the current Project

4.1 Explicit Task Decomposition
By segmenting the project in manageable tasks, the WBS makes sure that every facet of the system—such as admin features, appointment scheduling, and user authentication—is addressed.

4.2 Managing Timelines
The team can stay on course and achieve deadlines by using the Gantt Chart, which gives each task a clear timeframe. For instance, the 21 days allotted for the development phase are enough to construct and integrate all of the essential modules.

4.3 Handling Dependencies
Task dependencies are evident in the timetable. For instance, testing cannot start until development is complete, and development never start unless the design process is finished.

4.4 Allocation of Resources
Assigning assets (team members, tools, etc.) to particular tasks is made easier by the timetable. For instance, during the design phase, frontend developers will concentrate on UI/UX design, while backend developers would work on developing APIs and setting up databases.

4.5 Reduction of Risk
The timetable enables proactive risk mitigation by segmenting the project into smaller activities

and setting explicit deadlines. This helps identify potential hazards, such as development delays.

8. Key Issues and Solutions

i) Time Management & Workload Distribution

Issue:

- With only four team members, an uneven workload distribution may occur.
- College assignments and exams might affect availability, causing delays.

Solution:

- Assign tasks based on each member's strengths (e.g., frontend, backend, database, project management).
- Utilize a shared calendar to monitor progress and set deadlines.
- Conduct weekly check-ins to ensure everyone stays on track. **(Salas et al., 2008)**
-

ii) Technical Challenges

Issue:

- Some team members may have limited experience with specific technologies (e.g., database management, payment integration).
- Debugging and troubleshooting could take longer due to a lack of expertise.

Solution:

- Encourage early learning and upskilling before development begins.
- Utilize online resources (tutorials, forums) and seek guidance from professors when needed.
- Keep the system straightforward and avoid overly complex features. **(Ali, 2017)**
-

iii) Budget Constraints

Issue:

- Limited financial resources due to the nature of a college project.
- Paid tools, hosting services, or third-party integrations may exceed the budget.

Solution:

- Opt for open-source tools and free-tier cloud services for development and hosting.
- Avoid unnecessary paid plugins or features unless essential.
- Focus on core functionality rather than additional enhancements.

iv). Deployment & Cutover Issues

Issue:

- Moving from development to deployment may cause problems (e.g., server errors, database migration failures).
- Testers or professors might encounter difficulties accessing the system.

Solution:

- Conduct a trial deployment in a test environment before the final rollout.
- Maintain backup copies of all project files to prevent data loss.
- Assign a designated team member to oversee deployment and troubleshooting.

v) Security & Data Privacy Risks

Issue:

- The project may involve user data, requiring protection against unauthorized access.
- Weak authentication systems could lead to security vulnerabilities.

Solution:

- Implement basic encryption for user passwords and sensitive information.
- Use role-based access control (e.g., separate admin and user permissions).
- Adhere to best practices in data handling, even for a college-level project.

vi) Team Communication & Coordination

Issue:

- Miscommunication can lead to duplicated work or missing features.
- Differences in opinions may cause conflicts regarding project direction.

Solution:

- Use collaboration tools (e.g., Trello, Google Docs) to track tasks and updates.
- Hold regular short meetings to address concerns and align progress.
- Resolve conflicts through team discussions and voting on key decisions. **(Ala & Chen, 2022)**

9. Potential Human Factors in an Online Booking System & Our Approach

As a team, we recognize the importance of human factors in ensuring the usability and success of our online booking system. We have identified key challenges and developed strategies to address them effectively.

1. User Experience & Usability

Issue: Users may find navigation difficult, struggle with a complex interface, or face unclear instructions.

Solution:

- Design a clean, intuitive interface with clear icons and labels.
- Conduct usability testing with peers to gather feedback and make improvements.
- Provide step-by-step assistance through tooltips or an FAQ section.

2. Technical Literacy

Issue: Not all users are tech-savvy, which may lead to errors or confusion during the booking process.

Solution:

- Develop a user-friendly, mobile-responsive design.
- Keep the booking process simple with minimal steps.
- Offer a help section or chatbot to address common inquiries. **(Nielsen, 1993)**

3. Human Error (Incorrect Bookings)

Issue: Users might enter incorrect details or make accidental bookings.

Solution:

- Implement a confirmation step before finalizing bookings.
- Provide options to easily modify or cancel reservations.
- Use input validation to prevent errors like invalid dates or duplicate entries.

4. Accessibility & Inclusivity

Issue: Users with disabilities may face difficulties using the system.

Solution:

- Use accessible fonts, color contrast, and user-friendly design principles.
- Ensure keyboard navigation compatibility for better accessibility.
- Add alternative text for images and buttons to aid visually impaired users.

5. Trust & Security Concerns

Issue: Users may hesitate to provide personal or payment details due to security concerns.

Solution:

- Implement secure authentication methods, such as email verification and passwords.
- Use SSL encryption to protect user data.
- Clearly display privacy policies and terms of service to build trust. **(Giannoccaro et al., 2020)**

Managing Human Factors as a Team

1. **Dividing Responsibilities:**
    - **Frontend Developer** ensures a simple and accessible user interface.
    - **Backend Developer** handles data validation and security.
    - **Database Manager** ensures accurate storage and retrieval of user data.
    - **Project Manager** gathers user feedback and coordinates improvements.
2. **User Testing:** Conduct testing with classmates and teachers to identify usability issues.
3. **Iterative Improvement:** Make enhancements based on user feedback before final deployment.

10. Potential Privacy Risks and Solutions for the Online Appointment Booking System

Since our system includes user data (names, emails, phone numbers, and appointment details), privacy and security are vital issues. Below are the primary risks and our strategies to address them.

1. Unauthorized Access to User Data

🔴 Risk: If user data (appointments, contact details) is not secured properly, unauthorized users may gain access.

✅ Solution:

Implement password hashing (e.g., using bcrypt) to store passwords securely.

Use role-based access control, so only authorized admins can manage bookings.

Restrict access using session management (users must log in to book appointments).

2. Data Leakage or Exposure

🔴 Risk: Sensitive user data could be revealed because of inadequate storage or transfer (e.g., storing plain-text passwords).

✅ Solution:

Utilize encrypted storage for confidential data.

When information travels over the internet, make sure it's scrambled so nobody can read it along the way.

Only keep the absolute minimum number of personal details you need, so there's less chance of that information getting stolen. **(*Database Design for Mere Mortals ® Third Edition*, n.d.)**

3. Lack of User Consent & Data Misuse

🔴 Risk: Users may not be aware of how their data is used, leading to privacy concerns.

✅ Solution:

Clearly define a Privacy Policy stating how user data is used.

Require user consent before collecting or storing data.

Allow users to delete their account and data if they wish.

4. Data Breaches & Cyber Attacks

🔴 Risk: If the system is not secured, attackers might steal data through SQL Injection, Cross-Site Scripting (XSS), or brute-force attacks.

✅ Solution:

Use input validation to prevent SQL Injection.

Regularly update software to patch security vulnerabilities.

5. Insider Threats (Malicious Admins or Employees)

🔴 Risk: Admins may misuse their access to view or modify sensitive user data.

✅ Solution:

Log all admin activities (audit logs).

Implement two-factor authentication (2FA) for admin accounts if required.

Limit admin access to only necessary data.

11) Discuss any possible professional ethical issues in your project.

Data security and privacy
Ethical Concern: Names, phone numbers, email addresses, and appointment information are among the sensitive user data that the system gathers and keeps. Inappropriate handling of this data may result in data breaches or privacy violations.

Strategies for Mitigation:
Encrypt critical information when it's in motion and when it's at rest.

RBAC, or role-based access control, should be put in place to guarantee that only individuals with permission can access private information.

Privacy Policy: Get users' express consent and explain in detail the way user data is going to be used and stored.

Frequent Audits: To find and address vulnerabilities, perform frequent penetration tests and security audits.

2. Knowledgeable Consent
Ethical Concern: Users might not give their informed consent if they don't completely grasp how their information is being utilized.

Strategies for Mitigation:
Openness: Clearly and succinctly describe gathering data, storage, and use in your privacy policy.

Consent Forms: Before using the system, users must expressly accept the terms of service or privacy statement.

Give consumers the option to remove their login information and data at any moment, or to opt away from data gathering altogether.

3. Inclusivity and Accessibility
Ethical Concern: Discrimination or exclusion may result from the system's inability to accommodate all users, even those with disabilities.

Strategies for Mitigation:
Accessibility Standards: To guarantee that the system is useable by individuals with impairments, adhere to accessibility principles like WCAG (Web Content Accessibility principles).

Usability testing should be done with a variety of user groups, including people with disabilities.

Inclusive Design: In order to render the system more inclusive, use keyboard navigation, color contrasts, and accessible typefaces.

4. Fairness and Bias

Ethical Concern: By giving some users preference over other according to algorithms or data, the system may unintentionally inject bias into appointment scheduling.

Strategies for Mitigation:
Equitable Algorithms: Make sure scheduling or recommendation algorithms are impartial and transparent.

User input: Give people the opportunity to voice concerns about bias or fairness and respond to them right away.

Frequent Reviews: To guarantee equity, periodically examine the data and algorithms of the system.

5. Accountability and Transparency
Ethical Concern: A lack of accountability and transparency may result from users' ignorance of the system's operation.

Strategies for Mitigation:
Unambiguous Documentation: Clearly explain the system's operation, including the scheduling and management of appointments.

User Support: Provide a chatbot or help area to support users with any queries or problems.

Audit Trails: To guarantee accountability, keep track of every action administrators and users do.

Inclusion of

i) Contribution table
ii) Meeting minutes
iii) Clear version control of project documentation including source codes.

1) Contribution table

Group Member	Contribution	Details of contribution
Vedant Jadhav	Project schedule, ethical issues, references , meeting time and additional documentation	Planned project schedule, Gannt chart, Work breakdown structure. Analysed ethical considerations in the project. Managed final

		documentation, version control, source codes and made sure that report comply with report requirements. Managed meeting times of group as well as with professor. Managed possible ethical issues in the project. Provided with 12 scholarly references.
Amit Gupta	system development approach, functional requirements, privacy risks	Researched and justified the system development approach chosen. Functional requirements, data types and sizes are defined. Potential privacy risks and their proposed solutions are identified.
Sharmila Timalsina	Use case diagram, UI design, Nonfunctional requirements	Use case diagram is created and detailed explanation about them is provided. System navigation, and UI is designed, using wireframes. Analysed various non-functional requirements.
Lalit Maharjan	Project cost estimate, risk analysis, human factors	Project cost is estimated and created budget breakdown. Project risks and their solutions are identified. Human factors in system design are addressed and mitigation strategies are suggested.

ii) Meeting minutes

Group meeting

Date : 17th March

Time : 6 pm – 6:30 pm

Items discussed : Tasks allocation to all team members .

Schedule next meeting for discussion .

Meeting with professor

Date : 18th March

Time : 2:30 pm – 3 pm

Items discussed : Project status

Should have a group meeting at least twice a week .

Discussion about next meeting via zoom (Thursday)

Group meeting

Date : 19th March

Time : 6 pm – 6:30 pm

Items discussed : Final checking of assessment 2 before submission.

Some minor changes to be made

iii) Clear version control of project documentation including source codes in the appropriate order.

Database Setup (SQLite)

import sqlite3

```
# Connect to SQLite database (or create it if it doesn't exist)
conn = sqlite3.connect('appointment_system.db')
cursor = conn.cursor()

# Create Users table
cursor.execute('''
CREATE TABLE IF NOT EXISTS Users (
    UserID INTEGER PRIMARY KEY AUTOINCREMENT,
    Name TEXT NOT NULL,
    Email TEXT NOT NULL UNIQUE,
    Phone TEXT NOT NULL,
    Password TEXT NOT NULL
)
''')

# Create Appointments table
cursor.execute('''
CREATE TABLE IF NOT EXISTS Appointments (
    AppointmentID INTEGER PRIMARY KEY AUTOINCREMENT,
    UserID INTEGER,
    ServiceType TEXT NOT NULL,
    Date DATE NOT NULL,
    Time TIME NOT NULL,
    Status TEXT DEFAULT 'Pending',
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
)
''')

# Create Services table
cursor.execute('''
```

```
CREATE TABLE IF NOT EXISTS Services (
    ServiceID INTEGER PRIMARY KEY AUTOINCREMENT,
    ServiceName TEXT NOT NULL,
    Duration INTEGER NOT NULL,
    Price REAL NOT NULL
)
''')

# Commit changes and close connection
conn.commit()
conn.close()
```

User Authentication (Flask Backend)

```
from flask import Flask, request, jsonify, session
import sqlite3
from werkzeug.security import generate_password_hash, check_password_hash

app = Flask(__name__)
app.secret_key = 'your_secret_key'

# User Registration
@app.route('/register', methods=['POST'])
def register():
    data = request.get_json()
    name = data['name']
    email = data['email']
    phone = data['phone']
    password = generate_password_hash(data['password'])

    conn = sqlite3.connect('appointment_system.db')
    cursor = conn.cursor()
    cursor.execute('INSERT INTO Users (Name, Email, Phone, Password) VALUES (?, ?, ?, ?)',
```

```
            (name, email, phone, password))
    conn.commit()
    conn.close()
    return jsonify({"message": "User registered successfully"}), 201

# User Login
@app.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    email = data['email']
    password = data['password']

    conn = sqlite3.connect('appointment_system.db')
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM Users WHERE Email = ?', (email,))
    user = cursor.fetchone()
    conn.close()

    if user and check_password_hash(user[4], password):
        session['user_id'] = user[0]
        return jsonify({"message": "Login successful"}), 200
    else:
        return jsonify({"message": "Invalid credentials"}), 401

# User Logout
@app.route('/logout', methods=['POST'])
def logout():
    session.pop('user_id', None)
    return jsonify({"message": "Logged out successfully"}), 200
```

3. Appointment Booking (Flask Backend)

```
# Book Appointment
@app.route('/book', methods=['POST'])
def book_appointment():
    if 'user_id' not in session:
        return jsonify({"message": "Unauthorized"}), 401

    data = request.get_json()
    user_id = session['user_id']
    service_type = data['service_type']
    date = data['date']
    time = data['time']

    conn = sqlite3.connect('appointment_system.db')
    cursor = conn.cursor()
    cursor.execute('INSERT INTO Appointments (UserID, ServiceType, Date, Time) VALUES (?, ?, ?, ?)',
                   (user_id, service_type, date, time))
    conn.commit()
    conn.close()
    return jsonify({"message": "Appointment booked successfully"}), 201

# Reschedule Appointment
@app.route('/reschedule/<int:appointment_id>', methods=['PUT'])
def reschedule_appointment(appointment_id):
    if 'user_id' not in session:
        return jsonify({"message": "Unauthorized"}), 401

    data = request.get_json()
    new_date = data['date']
    new_time = data['time']
```

```
    conn = sqlite3.connect('appointment_system.db')
    cursor = conn.cursor()
    cursor.execute('UPDATE Appointments SET Date = ?, Time = ? WHERE AppointmentID = ?',
                   (new_date, new_time, appointment_id))
    conn.commit()
    conn.close()
    return jsonify({"message": "Appointment rescheduled successfully"}), 200

# Cancel Appointment
@app.route('/cancel/<int:appointment_id>', methods=['DELETE'])
def cancel_appointment(appointment_id):
    if 'user_id' not in session:
        return jsonify({"message": "Unauthorized"}), 401

    conn = sqlite3.connect('appointment_system.db')
    cursor = conn.cursor()
    cursor.execute('DELETE FROM Appointments WHERE AppointmentID = ?', (appointment_id,))
    conn.commit()
    conn.close()
    return jsonify({"message": "Appointment cancelled successfully"}), 200
```

4. Admin Functionality (Flask Backend)

```
# Admin Login (Assuming admin has a separate table)
@app.route('/admin/login', methods=['POST'])
def admin_login():
    data = request.get_json()
    email = data['email']
    password = data['password']
```

```
    conn = sqlite3.connect('appointment_system.db')
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM Admins WHERE Email = ?', (email,))
    admin = cursor.fetchone()
    conn.close()

    if admin and check_password_hash(admin[3], password):
        session['admin_id'] = admin[0]
        return jsonify({"message": "Admin login successful"}), 200
    else:
        return jsonify({"message": "Invalid credentials"}), 401

# View All Appointments
@app.route('/admin/appointments', methods=['GET'])
def view_appointments():
    if 'admin_id' not in session:
        return jsonify({"message": "Unauthorized"}), 401

    conn = sqlite3.connect('appointment_system.db')
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM Appointments')
    appointments = cursor.fetchall()
    conn.close()
    return jsonify({"appointments": appointments}), 200

# Approve/Reject Appointment
@app.route('/admin/appointments/<int:appointment_id>', methods=['PUT'])
def approve_reject_appointment(appointment_id):
    if 'admin_id' not in session:
        return jsonify({"message": "Unauthorized"}), 401
```

```
    data = request.get_json()
    status = data['status']

    conn = sqlite3.connect('appointment_system.db')
    cursor = conn.cursor()
    cursor.execute('UPDATE Appointments SET Status = ? WHERE AppointmentID = ?', (status, appointment_id))
    conn.commit()
    conn.close()
    return jsonify({"message": f"Appointment {status} successfully"}), 200
```

5. Security Measures

```
# Password Hashing (Already implemented in User Authentication)
# Input Validation (Example for SQL Injection Prevention)
def validate_input(input_data):
    if any(char in input_data for char in [';', '--', '/*', '*/']):
        return False
    return True

# Session Management (Already implemented in User Authentication)
```

6. Frontend (HTML/CSS/JavaScript)

```
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <h1>Login</h1>
    <form id="loginForm">
```

```
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required>
    <label for="password">Password:</label>
    <input type="password" id="password" name="password" required>
    <button type="submit">Login</button>
  </form>
  <script src="login.js"></script>
</body>
</html>
```

Login Page (JavaScript)

```
document.getElementById('loginForm').addEventListener('submit', function(event) {
  event.preventDefault();
  const email = document.getElementById('email').value;
  const password = document.getElementById('password').value;

  fetch('/login', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ email, password })
  })
  .then(response => response.json())
  .then(data => {
    if (data.message === "Login successful") {
      window.location.href = '/dashboard';
    } else {
      alert(data.message);
```

```
    }
  });
});
```

7. Running the Application

```
export FLASK_APP=app.py
flask run
```

References : References are added to the specific paragraphs (Harvard style).