Design and Implementation of Home Services Web Application

Aly Elsaka, Hamza Hassan, Abdelrahman Mohamed

January 9, 2025

January 9, 2025

Contents

1 Introduction

1.1 Purpose

The purpose of this project is to develop a comprehensive home service platform that connects clients with service providers, ensuring a seamless and user-friendly experience. This website will enable clients to browse available services, submit service requests, and manage ongoing tasks, while service providers can efficiently handle and track their assignments. The platform also offers an administrative interface for monitoring and managing the system.

1.2 Scope

1.2.1 Core Functionalities

- User authentication system with role-based access for clients, workers, and admins.
- Service request creation and management for clients, including status tracking.
- Workers can view, accept, or reject assigned tasks.
- Admin dashboard for user management and system oversight.

1.2.2 Advanced Features

- Real-time notifications to keep users updated on request statuses.
- Dynamic filtering of services by location, type, and availability to enhance search efficiency.
- A feedback system allowing clients to rate and review completed tasks.

1.2.3 Scalability and Compatibility

- The platform is designed to handle a growing user base and increasing service requests.
- Ensures cross-platform accessibility with responsive design for mobile and desktop users.

2 Requirements

2.1 Functional Requirements

1. **FR-001: Client, Admin, and Worker Authentication**
    - Clients must register and log in to book services and manage their bookings.
    - Admins must add workers, reject or approve workers.
    - Workers must register and log in to update their availability and view bookings.
    - Post-Condition: The user is logged in, and their role-based dashboard is accessible.

- Justification: Ensures secure access and personalized experiences for all users while clearly defining roles.

2. **FR-002: Service Search**
   - Clients must be able to browse and search for available services, including:
     - Electrician
     - Gardener
     - Plumber
     - Cleaning Services
   - Post-Condition: The system displays the results matching the user's query or suggests alternatives.
   - Justification: Core functionality to allow clients to find the desired home services easily.

3. **FR-003: Service Booking**
   - The system must allow clients to book a service provider based on availability.
   - Post-Condition: The booking details are stored, and a confirmation is sent to both the client and provider.
   - Justification: Critical for enabling the primary purpose of the system: scheduling services.

4. **FR-004: Provider Availability Management**
   - Workers must be able to update their availability through the system.
   - Admins must add workers to the system after verifying their IDs and government-issued documents.
   - Post-Condition: Worker availability is updated and visible to clients.
   - Justification: Enhances the system's reliability by ensuring real-time data accuracy and provider legitimacy.

5. **FR-005: Booking Confirmation**
   - The system must send booking confirmations via email or SMS to clients.
   - Post-Condition: Clients and providers receive confirmation of the booking details.
   - Justification: Keeps clients informed and provides documentation of their booking.

6. **FR-006: Service Reviews**
   - Clients should be able to view reviews and ratings for service providers.
   - Post-Condition: The user gains insight into the provider's quality and credibility.
   - Justification: Encourages trust and better decision-making for clients.

7. **FR-007: Multiple Payment Options**

- The system should support multiple payment options, including credit/debit cards and digital wallets.
- Post-Condition: The payment is successfully processed, and the booking is confirmed.
- Justification: Makes the system more client-friendly and accessible to a larger audience.

8. **FR-008: Booking Management**
   - Clients must be able to reschedule or cancel bookings through the platform.
   - Post-Condition: Booking details are updated or removed, and notifications are sent to relevant parties.
   - Justification: Adds flexibility and improves client satisfaction.
9. **FR-009: Admin Dashboard**
   - The system must include an admin dashboard for:
     - Managing client and worker accounts.
     - Adding or removing services.
     - Monitoring platform activity and generating reports.
   - Post-Condition: Changes made by the admin are applied and logged for audit purposes.
   - Justification: Essential for operational control and oversight by administrators.
10. **FR-010: Cancel Booking**
    - Clients must have the option to cancel a booking after scheduling a service.
    - Post-Condition: The booking is canceled, and the client receives confirmation and refund details if applicable.
    - Justification: Provides flexibility and accommodates changes in client plans.

2.2 Non-Functional Requirements

1. **NFR-001: Performance Testing**
   - The platform must ensure an average page load time under 3 seconds.
   - Justification: Provides a responsive user experience, critical for client retention.
2. **NFR-002: Multilingual Support**
   - The system must support interfaces in the top three languages: English, Spanish, and French.
   - Justification: Enhances accessibility and usability for a diverse audience.
3. **NFR-003: Cross-Platform Compatibility**

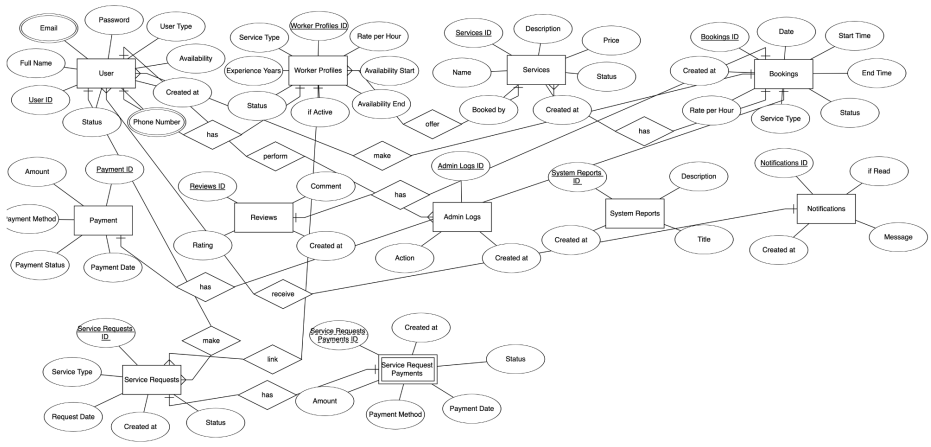
- The platform must be compatible with desktop browsers, including Firefox, Chrome, and Edge.
- Justification: Provides seamless access across devices, catering to modern user behavior.

2.3 Won't Haves

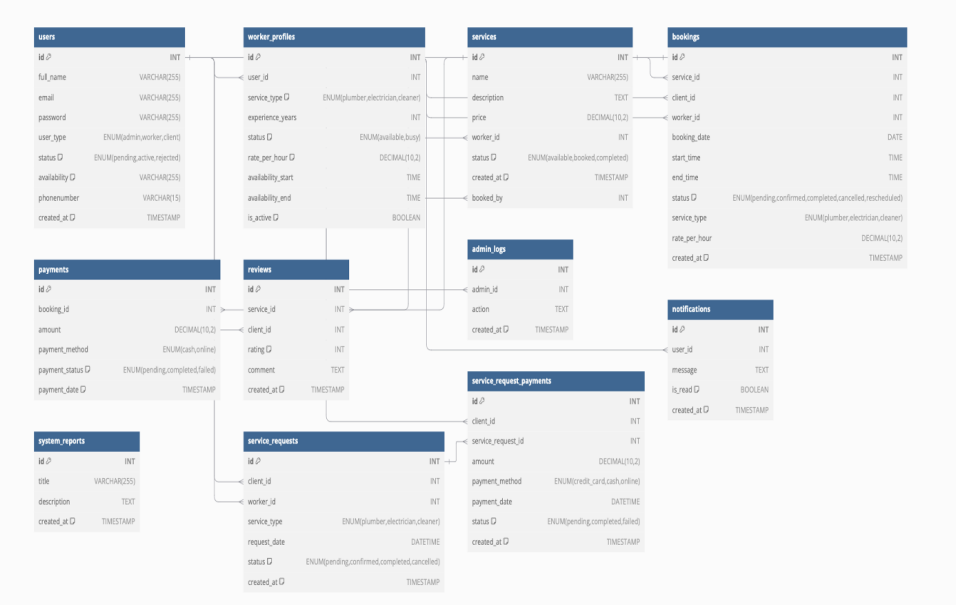
- Advanced AI Chatbot
- Offline Booking System
- Custom Pricing for Services
- Service Delivery Time Guarantees

3 System Design

3.1 Entity Relationship Diagram



3.2 Relational Model



4 Implementation

4.1 SQL Queries

Below are the SQL queries used to create the database schema. These queries are stored in a file named `queries.sql` and are displayed here for reference.

```sql
CREATE DATABASE IF NOT EXISTS home_services;
USE home_services;

-- Users Table
CREATE TABLE users (
    id INT PRIMARY KEY AUTO_INCREMENT,
    full_name VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    user_type ENUM('admin', 'worker', 'client') NOT NULL,
    status ENUM('pending', 'active', 'rejected') DEFAULT 'pending',
    availability VARCHAR(255) DEFAULT 'Available',
    phonenumber VARCHAR(15),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Worker Profiles Table
CREATE TABLE worker_profiles (
    id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT,
    service_type ENUM('plumber', 'electrician', 'cleaner') NOT NULL DEFAULT 'plumber',
    experience_years INT,
```

```
    status ENUM('available', 'busy') DEFAULT 'available',
    rate_per_hour DECIMAL(10, 2) DEFAULT 0.00,
    availability_start TIME,
    availability_end TIME,
    is_active TINYINT(1) DEFAULT 1,
    FOREIGN KEY (user_id) REFERENCES users(id)
);

CREATE TABLE services (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    price DECIMAL(10, 2) NOT NULL,
    worker_id INT,
    status ENUM('available', 'booked', 'completed') DEFAULT '
       available',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    booked_by INT,
    FOREIGN KEY (worker_id) REFERENCES users(id),
    FOREIGN KEY (booked_by) REFERENCES users(id)
);

-- Bookings Table
CREATE TABLE bookings (
    id INT PRIMARY KEY AUTO_INCREMENT,
    service_id INT NOT NULL,
    client_id INT NOT NULL,
    worker_id INT NOT NULL,
    booking_date DATE NOT NULL,
    start_time TIME NOT NULL,
    end_time TIME NOT NULL,
    status ENUM('pending', 'confirmed', 'completed', 'cancelled',
        'rescheduled') DEFAULT 'pending',
    service_type ENUM('plumber', 'electrician', 'cleaner'),
    rate_per_hour DECIMAL(10, 2),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (service_id) REFERENCES services(id),
    FOREIGN KEY (client_id) REFERENCES users(id),
    FOREIGN KEY (worker_id) REFERENCES users(id)
);

-- Payments Table
CREATE TABLE payments (
    id INT PRIMARY KEY AUTO_INCREMENT,
    booking_id INT NOT NULL,
    amount DECIMAL(10, 2) NOT NULL,
    payment_method ENUM('cash', 'online') NOT NULL,
    payment_status ENUM('pending', 'completed', 'failed') DEFAULT
        'pending',
    payment_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (booking_id) REFERENCES bookings(id)
```

```sql
);

-- Reviews Table
CREATE TABLE reviews (
    id INT PRIMARY KEY AUTO_INCREMENT,
    service_id INT NOT NULL,
    client_id INT NOT NULL,
    rating INT NOT NULL CHECK (rating BETWEEN 1 AND 5),
    comment TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (service_id) REFERENCES services(id),
    FOREIGN KEY (client_id) REFERENCES users(id)
);

-- Admin Logs Table
CREATE TABLE admin_logs (
    id INT PRIMARY KEY AUTO_INCREMENT,
    admin_id INT NOT NULL,
    action TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (admin_id) REFERENCES users(id)
);

-- Notifications Table
CREATE TABLE notifications (
    id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT NOT NULL,
    message TEXT NOT NULL,
    is_read BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id)
);

-- System Reports Table
CREATE TABLE system_reports (
    id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Service Requests Table
CREATE TABLE service_requests (
    id INT PRIMARY KEY AUTO_INCREMENT,
    client_id INT NOT NULL,
    worker_id INT NOT NULL,
    service_type ENUM('plumber', 'electrician', 'cleaner') NOT
       NULL,
    request_date DATETIME NOT NULL,
    status ENUM('pending', 'confirmed', 'completed', 'cancelled')
        DEFAULT 'pending',
```

```sql
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (client_id) REFERENCES users(id),
    FOREIGN KEY (worker_id) REFERENCES users(id)
);

-- Payments Table (for service requests)
CREATE TABLE service_request_payments (
    id INT PRIMARY KEY AUTO_INCREMENT,
    client_id INT NOT NULL,
    service_request_id INT NOT NULL,
    amount DECIMAL(10, 2) NOT NULL,
    payment_method ENUM('credit_card', 'cash', 'online') NOT NULL,
    payment_date DATETIME NOT NULL,
    status ENUM('pending', 'completed', 'failed') DEFAULT 'pending',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (client_id) REFERENCES users(id),
    FOREIGN KEY (service_request_id) REFERENCES service_requests(id)
);

-- ================================================
-- SQL Queries from server.js
-- ================================================

-- Worker Booking Route
SELECT * FROM worker_profiles WHERE user_id = ?;
SELECT * FROM services WHERE id = ?;
INSERT INTO bookings (service_id, client_id, worker_id, booking_date, start_time, end_time, status, service_type, rate_per_hour) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?);

-- Login Route
SELECT * FROM users WHERE email = ?;

-- User Registration Route
SELECT * FROM users WHERE email = ?;
INSERT INTO users (full_name, email, password, user_type, phonenumber, status) VALUES (?, ?, ?, ?, ?, ?);
INSERT INTO worker_profiles (user_id, service_type, experience_years) VALUES (?, ?, ?);

-- Fetch Logged-in User's Details
SELECT id, full_name AS name, email, user_type FROM users WHERE id = ?;

-- Fetch Available Workers
SELECT users.id, users.full_name, users.email, worker_profiles.service_type, worker_profiles.rate_per_hour, worker_profiles.availability_start, worker_profiles.availability_end
```

```sql
FROM users
JOIN worker_profiles ON users.id = worker_profiles.user_id
WHERE users.user_type = 'worker' AND users.status = 'active';

-- Fetch Available Services
SELECT services.id, services.name, services.price, users.
   full_name AS worker_name
FROM services
JOIN users ON services.worker_id = users.id
WHERE services.status = 'available';

-- Fetch Client Bookings
SELECT bookings.id, services.name AS service, bookings.
   booking_date AS date, bookings.start_time, bookings.end_time,
   bookings.rate_per_hour AS price
FROM bookings
JOIN services ON bookings.service_id = services.id
WHERE bookings.client_id = ?;

-- Password Reset Endpoint
SELECT * FROM users WHERE email = ? AND phonenumber = ?;
UPDATE users SET password = ? WHERE email = ?;

-- Cancel a Booking
SELECT * FROM bookings WHERE id = ? AND worker_id = ?;
DELETE FROM bookings WHERE id = ?;

-- Fetch Client Payments
SELECT * FROM payments WHERE service_request_id IN (SELECT id
   FROM service_requests WHERE client_id = ?);

-- Fetch Client Notifications
SELECT * FROM notifications WHERE user_id = ? ORDER BY created_at
    DESC;

-- Fetch Reviews
SELECT reviews.id, services.name AS service, reviews.rating,
   reviews.comment, reviews.created_at
FROM reviews
JOIN bookings ON reviews.booking_id = bookings.id
JOIN services ON bookings.service_id = services.id
WHERE bookings.client_id = ?;

-- Admin Endpoint to Approve Workers
UPDATE users SET status = "active" WHERE id = ?;

-- Reject a Worker
DELETE FROM worker_profiles WHERE user_id = ?;
DELETE FROM bookings WHERE worker_id = ?;
DELETE FROM users WHERE id = ?;

```

```sql
-- Update Worker Availability
UPDATE worker_profiles SET availability_start = ?,
   availability_end = ? WHERE user_id = ?;

-- Update Worker Rate Per Hour
UPDATE worker_profiles SET rate_per_hour = ? WHERE user_id = ?;

-- Update Worker Active Status
UPDATE worker_profiles SET is_active = ? WHERE user_id = ?;

-- Fetch Pending Workers
SELECT users.id, users.full_name, users.email, worker_profiles.
   service_type, worker_profiles.experience_years
FROM users
JOIN worker_profiles ON users.id = worker_profiles.user_id
WHERE users.user_type = 'worker' AND users.status = 'pending';

-- Fetch All Services
SELECT services.id, services.name, services.price, users.
   full_name AS worker_name
FROM services
JOIN users ON services.worker_id = users.id;

-- Delete a Service
DELETE FROM services WHERE id = ?;

-- Fetch Admin Reports
SELECT * FROM system_reports;

-- ================================================
-- SQL Queries from api.js
-- ================================================

-- Fetch Services
SELECT id, name, price AS rate_per_hour FROM services;

-- Fetch Bookings Based on User Role (Client)
SELECT bookings.id, services.name AS service, bookings.
   booking_date AS date, bookings.start_time, bookings.end_time,
   bookings.rate_per_hour, bookings.total_price
FROM bookings
JOIN services ON bookings.service_id = services.id
WHERE bookings.client_id = ?;

-- Fetch Bookings Based on User Role (Worker)
SELECT bookings.id, services.name AS service, bookings.
   booking_date AS date, bookings.start_time, bookings.end_time,
   bookings.rate_per_hour, bookings.total_price
FROM bookings
JOIN services ON bookings.service_id = services.id
WHERE bookings.worker_id = ?;
```

```
250
251 -- Create a New Service
252 INSERT INTO services (name, price, worker_id, availability_start,
        availability_end, is_active, status) VALUES (?, ?, ?, ?, ?,
    ?, ?);
253
254 -- Fetch Available Services
255 SELECT services.id, services.name, services.price AS
    rate_per_hour, users.full_name AS worker_name
256 FROM services
257 JOIN users ON services.worker_id = users.id
258 WHERE services.status = 'available';
259
260 -- Book a Service
261 SELECT * FROM services WHERE id = ?;
262 INSERT INTO bookings (client_id, service_id, booking_date,
    start_time, end_time, rate_per_hour, status) VALUES (?, ?, ?,
    ?, ?, ?, ?);
263 UPDATE services SET status = ? WHERE id = ?;
264
265 -- Fetch Client Bookings
266 SELECT bookings.id, services.name AS service, bookings.
    booking_date AS date, bookings.start_time, bookings.end_time,
    bookings.rate_per_hour, bookings.total_price, users.
    phonenumber AS worker_phone
267 FROM bookings
268 JOIN services ON bookings.service_id = services.id
269 JOIN users ON bookings.worker_id = users.id
270 WHERE bookings.client_id = ?;
271
272 -- Cancel a Booking
273 SELECT * FROM bookings WHERE id = ? AND client_id = ?;
274 DELETE FROM bookings WHERE id = ?;
275
276 -- Reschedule a Booking
277 SELECT * FROM bookings WHERE id = ? AND client_id = ?;
278 UPDATE bookings SET booking_date = ?, start_time = ? WHERE id =
    ?;
279
280 -- Fetch Pending Workers for Admin Approval
281 SELECT users.id, users.full_name, users.email, users.phonenumber,
        worker_profiles.service_type, worker_profiles.
    experience_years
282 FROM worker_profiles
283 JOIN users ON worker_profiles.user_id = users.id
284 WHERE users.status = 'pending';
285
286 -- Approve a Worker
287 UPDATE users SET status = ? WHERE id = ?;
288
289 -- Reject a Worker
```

```sql
DELETE FROM worker_profiles WHERE user_id = ?;
DELETE FROM users WHERE id = ?;

-- Fetch Admin Reports
SELECT id, title, description, created_at
FROM system_reports
ORDER BY created_at DESC;

-- Add a New Service with Worker's Service Type
SELECT service_type FROM worker_profiles WHERE user_id = ?;
INSERT INTO services (name, description, price, worker_id,
   availability_start, availability_end, is_active, status)
   VALUES (?, ?, ?, ?, ?, ?, ?, ?);

-- Fetch Worker's Services
SELECT id, name, price, availability_start, availability_end,
   status
FROM services
WHERE worker_id = ?;

-- Fetch Worker ID Based on User ID
SELECT id FROM users WHERE id = ? AND user_type = "worker";

-- Automatically Create a Worker Profile
SELECT * FROM worker_profiles WHERE user_id = ?;
INSERT INTO worker_profiles (user_id, service_type,
   experience_years, status, rate_per_hour, is_active) VALUES (?,
    ?, ?, ?, ?, ?);

-- Fetch Worker's Profile
SELECT users.id, users.full_name, users.email, users.phonenumber,
    worker_profiles.service_type
FROM users
JOIN worker_profiles ON users.id = worker_profiles.user_id
WHERE users.id = ?;
```

5 Risk Plan

5.1 Risk Plan Diagram

Risk	Likelihood	Effect	Mitigation Plan
Missing field validation	High	Users may enter invalid data	Add simple checks for email and password
Broken links	Medium	Users can't navigate the site	Test all links to ensure they work
Page not loading	Medium	Users can't access the website	Check the code for errors in file paths
Weak passwords	High	Accounts may be hacked	Ask users to create longer passwords
Server crashes	Low	Website becomes unavailable	Restart the server if it stops working

6 GUI Development Using Prompt Engineering

6.1 Stage 1: Initial Prototype

6.1.1 Prompt

"I am a software engineering student working on a project to create a basic home service website. The website should include the following pages: Home, Login, Signup, Service Requests, and an Admin Dashboard. It should have simple navigation elements, such as a top navigation bar and footer, and placeholders for service categories and descriptions. The layout should include clear call-to-action buttons like 'Request Service,' 'Login,' and 'Sign Up.' Focus on a clean and user-friendly design. My requirements are attached in the provided file, and all of them need to be satisfied. Start by outlining the necessary files and their names. The project must be developed using Visual Studio Code, with MySQL as the database."

6.1.2 Techniques Used

1. **Clear and Actionable Instruction**: The prompt explicitly states the task ("create a basic website") and lists key elements, such as required pages and navigation structure.

2. **Task Breakdown**: Each core feature is detailed (e.g., Home, Login, Signup, Admin Dashboard) to provide clear implementation guidelines.

3. **User-Centric Focus**: The design emphasizes usability through clear call-to-action buttons and a user-friendly layout.

4. **Practical Constraints**: The prompt specifies the development environment (Visual Studio Code) and database choice (MySQL), aligning with the project requirements.

5. **Testable Outcome**: Deliverables like navigation, placeholders, and buttons are easily testable for functionality and completeness.

—

6.2 Stage 2: Adding Basic Interactivity

6.2.1 Prompt

"Introduce interactive features like a searchable list of service providers. Implement filters for location, service type, and availability. Ensure that the results dynamically update based on user input without reloading the page. Design a responsive interface to ensure usability on mobile and desktop."

6.2.2 Techniques Used

1. **Incremental Complexity**: Adds interactivity by introducing dynamic updates and search functionality.
2. **Explicit Functional Requirements**: Specifies filters and real-time updates to ensure clear functionality.
3. **Responsive Design**: Stresses design compatibility with multiple screen sizes.
4. **UI/UX Elements**: Introduces dropdowns and search bars for intuitive use.
5. **Testable Outcome**: The success can be tested by trying the filters and ensuring responsive behavior.

—

6.3 Stage 3: User Authentication System

6.3.1 Prompt

"Create a user authentication system with login, registration, and role-based access control for clients, workers, and admins. Upon login, users should be directed to role-specific dashboards (e.g., clients to service request history, workers to task lists, and admins to user management). Ensure session management for login persistence."

6.3.2 Techniques Used

1. **Task Decomposition**: Breaks down authentication into manageable tasks (login, registration, role-based access).
2. **Role-Specific Instructions**: Clearly specifies what each user role should experience post-login.
3. **Technical Context**: Highlights session management for persistence.
4. **Logical Flow**: Guides user experience from login to role-specific dashboards.

5. **Testable Outcome**: Easily testable by simulating login for each role and verifying dashboard access.

—

6.4 Stage 4: Service Request Workflow

6.4.1 Prompt

"Enable clients to create service requests by selecting a service type, specifying location and time, and leaving optional instructions. Allow workers to view and accept/reject service requests through their dashboard. Notify clients about the status of their requests in real-time."

6.4.2 Techniques Used

1. **Feature-Specific Clarity**: Details the exact steps for service request creation and management.
2. **Role-Specific Design**: Outlines separate workflows for clients and workers.
3. **Real-Time Interaction**: Emphasizes real-time notifications for immediate updates.
4. **Actionable Design**: Clear user actions like 'accept' or 'reject' for workers.
5. **Testable Outcome**: Verify the workflow by creating requests and monitoring status changes.

—

6.5 Stage 5: Admin Panel

6.5.1 Prompt

"Design an admin panel to manage users, monitor service requests, and generate system reports. Include features like user role assignment, approval/rejection of new workers, and viewing platform analytics (e.g., completed tasks, user activity)."

6.5.2 Techniques Used

1. **Task Prioritization**: Focuses on critical admin functionalities (user management, analytics).
2. **Role-Specific Clarity**: Highlights administrative oversight and control.
3. **Data Visualization**: Encourages use of charts and graphs for analytics.
4. **Management Tools**: Introduces approval workflows for worker onboarding.
5. **Testable Outcome**: Testable by verifying user role assignment and analytics.

—

6.6 Stage 6: Payment Integration

6.6.1 Prompt

"Integrate secure payment processing for service transactions. Allow clients to pay through various methods (e.g., credit card, digital wallets). Notify workers of successful payments. Store payment records securely for future reference."

6.6.2 Techniques Used

1. **Secure Implementation**: Highlights the importance of secure payment gateways.
2. **User-Centric Design**: Focuses on seamless payment options for clients.
3. **Real-Time Updates**: Ensures workers are notified of payments instantly.
4. **Data Retention**: Includes secure storage of payment history.
5. **Testable Outcome**: Validate by testing payment flows with mock transactions.

—

6.7 Stage 7: Feedback and Rating System

6.7.1 Prompt

"Implement a feedback system allowing clients to rate workers and leave comments after task completion. Display aggregated ratings on worker profiles. Notify workers of new feedback."

6.7.2 Techniques Used

1. **User Engagement**: Encourages client-worker interaction through ratings.
2. **Data Aggregation**: Summarizes ratings for quick assessment.
3. **Notifications**: Keeps workers informed of client opinions.
4. **Transparency**: Builds trust by showing ratings publicly.
5. **Testable Outcome**: Test by submitting feedback and checking its reflection on profiles.

—

6.8 Stage 8: Finalizing Design

6.8.1 Prompt

"Polish the website's design with a cohesive color scheme, clean typography, and user-friendly animations. Add hover effects on buttons, improve spacing, and ensure all elements align properly across pages. Test for accessibility, responsiveness, and cross-browser compatibility."

6.8.2 Techniques Used

1. **Aesthetic Refinement**: Focuses on visual consistency and branding.
2. **UX Enhancements**: Encourages smooth interactions with animations and hover effects.
3. **Accessibility**: Promotes inclusive design practices.
4. **Cross-Device Functionality**: Ensures responsiveness on various devices.
5. **Testable Outcome**: Verify through design audits and cross-browser tests.

—

7 Testing and Validation

7.1 Test Plan

7.1.1 Objectives

- Verify the functionality of user registration, login, booking management, admin service management, and "Forgot Password."
- Ensure integration between modules such as booking management and service management.
- Validate both functional and non-functional requirements (e.g., performance, security).

7.1.2 Scope

- **In Scope**:
  - User registration, login, and "Forgot Password."
  - Booking creation, cancellation, and rescheduling by clients.
  - Admin functionalities such as adding and deleting services.
  - Integration testing for core workflows.
  - Non-functional testing, including performance and security validation.
- **Out of Scope**:
  - Third-party integrations, such as email notifications.
  - Testing on platforms other than Windows and Chrome.

7.2 Test Strategy

7.2.1 Manual Testing

- **Purpose**: Focused on usability testing, exploratory testing, and scenarios that require human judgment.
- **Application**:
  - Testing scenarios such as user registration, login, booking creation, and admin functionalities.
  - Validating error handling for edge cases, such as invalid data submissions.

7.2.2 Automation Testing

- **Purpose**: Ensures repeatability and efficiency, especially for regression and integration testing.
- **Application**:
  - Automating workflows for login, registration, and "Forgot Password."
  - Validating integration workflows, including booking operations and service management.

7.3 Test Environment

- **Operating System**: Windows 10
- **Browser**: Chrome 116
- **Application Version**: 1.0
- **Tools**:
  - Automation Framework: Python scripts using Selenium.
  - Defect Reporting: Excel-based defect tracking.

7.4 Test Deliverables

- Test cases document.
- Defect report.
- Test execution summary.
- Final test report.

7.5 Test Cases

7.5.1 Manual Test Cases

Manual testing will focus on critical user flows, including:

- **TC_001**: Verify that a registered user can log in successfully.
- **TC_002**: Verify that a new user can register successfully.
- **TC_003**: Validate the "Forgot Password" functionality.
- **TC_004**: Verify that a client can reschedule a booking.
- **TC_005**: Confirm that a client cannot cancel a completed booking.
- **TC_006**: Verify that an admin can add a new service with valid inputs.
- **TC_007**: Ensure appropriate error messages for invalid inputs when adding services.
- **TC_008**: Validate compliance with all functional requirements, such as login and booking management.
- **TC_009**: Ensure the application meets non-functional requirements, including performance and security.

7.5.2 Automation Testing

The following scenarios have been automated:

- **Login Automation**:
  - Automated script validates login functionality with valid credentials.
  - Status: Success
- **Registration Automation**:
  - Automated script confirms successful registration with valid data.
  - Status: Success
- **Forgot Password Automation**:
  - Automated script tests password reset functionality.
  - Status: Failure

7.6 Defect Management

7.6.1 Critical Defects

- **DEF_003 (Critical)**: Forgot password functionality fails.
  - Description: The password reset link is not sent to the registered email.
  - Impact: Prevents users from resetting their passwords, blocking account access.
  - Action: Immediate resolution required.

7.6.2 High-Priority Defects

- **DEF_001 (High)**: Admin cannot add a new service.
  - Description: System shows a "Failed to fetch" error when adding services.
  - Impact: Prevents the addition of new services, disrupting catalog management.
  - Action: Prioritize fixing validation logic for service addition.
- **DEF_002 (High)**: Admin cannot delete a service.
  - Description: Similar to DEF_001, the system fails with a "Failed to fetch" error when deleting services.
  - Impact: Prevents proper maintenance of the service catalog.
  - Action: Investigate and resolve promptly.

7.7 Test Execution

- **Manual Testing**:
  - Validate all critical workflows manually.
  - Document and report any defects encountered during the test cycle.
- **Automation Testing**:
  - Execute automated scripts for login, registration, and "Forgot Password."
  - Validate results and report issues.
- **Defect Retesting**:
  - Retest resolved defects to confirm fixes.
  - Conduct regression testing to ensure no new issues are introduced.

8 Future Enhancements

- Incorporate AI for Intelligent Matching: Use AI algorithms to match clients with workers based on skills, availability, and location for more personalized service recommendations.
- Advanced Analytics for Admin Dashboards: Provide insights into user activity, booking trends, and system performance to help administrators make data-driven decisions.
- Enable Multilingual Support: Add support for multiple languages to make the platform accessible to a broader audience globally.
- Introduce Mobile Application Support: Develop a mobile app to extend platform accessibility and improve user convenience.
- Enhance Security Features: Implement advanced encryption and two-factor authentication for enhanced user data protection.

9 Conclusion

This project successfully delivers a comprehensive platform for managing home services, combining robust backend functionality with an intuitive user interface. The platform facilitates seamless interactions between clients, workers, and administrators, meeting core functional requirements effectively. Future enhancements will focus on scalability, security, and personalization, ensuring the platform remains user-centric and adaptive to evolving needs.

10 GitHub Repository

The source code for this project is available on GitHub:
`https://github.com/alyelsaka/HomeService.git`