Waste Management System

1.Project Overview:

1. What's the Goal?

- **Objective:** The main goal of the waste management system project is to create a streamlined, solution for tracking and managing waste collection efficiently. The system allows users to book waste collection services, monitor the status of their requests, and receive timely updates on the collection process. It does this by tracking requests in a manner similar to delivery tracking in e-commerce platforms, like Flipkart or Amazon, but without GPS functionality.
- **Scope Includes:**
    - User account management (sign-up/login)
    - Waste collection booking interface with options for date, time, waste type, and quantity
    - A real-time or near-real-time tracking system for collection requests
    - Admin and operator dashboards for managing bookings and tracking status updates
- **Scope Excludes:**
    - GPS tracking
    - Extensive user profiling and analytics
    - Integration with external tracking devices

2. Why it is Important?

- **Efficiency in Waste Collection:** By organizing and tracking waste collection requests digitally, the system can reduce manual processing, minimize delays, and optimize collection schedules.
- **Environmental Impact:** A more organized waste management system can help reduce waste mismanagement and pollution by promoting timely waste collection and disposal.
- **Enhanced User Experience:** Users benefit from a more convenient and transparent waste collection service, which helps foster positive environmental practices.
- **Operational Goals:** Ensure a streamlined workflow for both users and administrators, with an intuitive interface, easy tracking updates, and efficient data management.

3. Who's Involved?

- **Project Leader:** Myself, responsible for overall project management, setting deadlines, defining goals, and ensuring all aspects are on track.

- **Team Members:**
    1. Chekka Geetha Sri
    2. Ritesh Kumar Gupta
    3. Paul Maxwell

    Their roles are,

    - Developers
    - Database Administrator (DBA)
    - UI Designer
- **End Users:**
    - The target users of this web application is the every people who need to maintain their wastes and dispose them properly in the manner which does not cause any harm to other neighbours.

2.Requirements Documentation:

1. What Does it Need to Do?

- **Core Features and Functions:**
    - **User Account Management:** Allow users to register, log in, and manage their profiles. User data is securely stored in the system.
    - **Booking System:** Enable users to schedule waste collection by entering details such as date, time, waste type, quantity, and pickup location.
    - **Collection Tracking:** Users can track the status of their collection requests through status updates (e.g., Scheduled, In Progress, Collected).
    - **Admin Dashboard:** Admins can view, manage, and update all bookings. They have the ability to change the status of a booking and manage operational schedules.
    - **Notification System:** Send notifications to users on booking status changes (e.g., booking confirmation, collection reminder, completion notification).
    - **Popup for Booking Updates:** Users can update the booking date and time via a popup for quick and convenient adjustments.

2. How Well Does it Need to Work?

- **Performance:**
    - The system should handle concurrent requests from multiple users without significant lag, especially during booking and tracking interactions.

    - Quick loading times for dashboards and tracking updates to ensure a smooth user experience.
- **Security:**
    - Secure user authentication and data encryption to protect user information and sensitive data.
    - Implement access control for different user roles (e.g., admin vs. regular users) to restrict data access.
    - Data validation and sanitization to prevent SQL injection, cross-site scripting (XSS), and other potential vulnerabilities.
- **Usability:**
    - User-friendly interface with an intuitive layout, especially for tracking and booking.
    - Accessible design for users with varying levels of technical ability, making key features like booking and tracking easy to navigate.
    - Dark and aesthetic theme for a visually appealing and consistent experience.
- **Reliability:**
    - Maintain high uptime and ensure that critical functions (like booking, tracking, and notifications) work reliably under typical usage conditions.
    - Comprehensive error handling to manage unexpected issues gracefully.

3. Why is it Needed?

- **Business Objectives Supported:**
    - **Improved Operational Efficiency:** Streamlines the booking and tracking of waste collection, reducing manual intervention and potential errors.
    - **Enhanced Customer Satisfaction:** By providing real-time status updates and an intuitive booking process, users experience a more transparent and responsive service.
    - **Sustainability Goals:** Helps the company align with environmental goals by promoting organized waste management and responsible disposal practices.
    - **Data-Driven Decision Making:** Collecting and analyzing data on bookings and collections provides insights that can improve service efficiency and customer engagement.
- **Company Alignment:** The project aligns with the company's commitment to environmental sustainability, operational efficiency, and superior customer service.

4. How Will Users Use It?

- **User Stories / Use Cases:**

- **User Story 1:** *As a resident, I want to book a waste collection service by specifying my preferred date, time, waste type, and quantity, so I can easily manage waste disposal.*
- **User Story 2:** *As a resident, I want to view the status of my booking in real time, so I know exactly when my waste will be collected.*
- **User Story 3:** *As an admin, I want to view all upcoming bookings in a dashboard and update their statuses, so I can keep track of operations and ensure timely collections.*
- **User Story 4:** *As a resident, I want to receive notifications when the status of my booking changes, so I stay informed about any updates or delays.*
- **User Story 5:** *As a resident, I want to update my booking's date and time through a popup option, so I can quickly make adjustments without navigating to a different page.*
- **User Story 6:** *As an admin, I want to generate reports on waste collection activities, so I can evaluate performance and identify areas for improvement.*

3.Project Plan:

1. When Will it Be Done?

- **Milestone 1(Week 1-3):**
  • Successful user authentication and registration implementation.
  • Password reset and email verification functionalities.

- **Milestone 2 (Week 4-5):**
  • Successful waste collection request functionality.
  • Implementation of scheduling and status tracking for waste collection.

- **Milestone 3 (Week 6-7):**
  • Admin dashboard completed with request management functionalities.
  • Ability for admins to monitor and assign tasks for collection.

- **Milestone 4(Week 8-10):**
  • Fully functional notification system integrated with email/SMS services.

2. What Do We Need?

- **People:**
  - Project Leader
  - Backend Developer

    - Frontend Developer
    - Tester (To test the proper functioning of the website)
- **Equipment:**
    - Development Laptops/Desktops
    - Internet connection
- **Software:**
    - Development Tools: Python, Django, MySQL, VS Code
    - Frontend Tools: HTML, CSS, JavaScript
    - Testing: Manual testing
    - Deployment Tools: GitHub (version control)
- **Miscellaneous:**
    - Documentation tools (Google Docs)
    - Communication tools (Microsoft Teams)

3. **How Much Will it Cost?**

Since it is a self project, it is of no cost. But if we host this project in any cloud source, then it would cost minimum of rupees 200/month.

4.Architecture and Design:

1. How is it Built?

- **System Architecture:**
    1. The system follows a **Client-Server Architecture** with the following components:
        - **Client Side (Frontend):**
            - Built using HTML, CSS, and JavaScript.
            - Provides a user interface for waste booking, status tracking, and admin dashboards.
            - Includes a popup for booking updates.
        - **Server Side (Backend):**
            - Built using Django.
            - Handles user authentication, booking management, status updates, and notification logic.
            - Communicates with the database to store and retrieve data.
        - **Database:**
            - MySQL database to store user data, bookings, and tracking information.
            - Uses foreign keys to maintain relationships between tables.
        - **Notification System:**

        - Backend triggers email notifications to inform users about status changes.
- **Subsystem Interaction:**
    1. **User Actions:**
        - Users interact with the client-side interface to book waste collection or track its status.
    2. **API Requests:**
        - The client sends HTTP requests to the backend for actions like creating or updating bookings.
    3. **Data Processing:**
        - The backend processes the requests, updates the database, and returns responses.
    4. **Database Interaction:**
        - The backend performs CRUD operations on the MySQL database.
    5. **Notification Trigger:**
        - The backend triggers notifications when booking statuses change.

2. What Does it Look Like?

- **UML Diagrams and Flowcharts:** *(Descriptions for now; diagrams can be generated if needed.)*
    - **Use Case Diagram:**
        - Actors: User (Resident), Admin
        - Use Cases: Register/Login, Book Waste Collection, Track Status, Update Booking, Manage Bookings, View Bookings.
    - **System Flow Diagram:**
        - **Step 1:** User logs into the system.
        - **Step 2:** User books waste collection by submitting required details.
        - **Step 3:** The system saves the booking in the database and notifies the user.
        - **Step 4:** Admin updates booking status through the dashboard.
        - **Step 5:** Status update triggers a notification sent to the user.
- **Wireframe Design:**
    - **Home Page:** Includes options for login, booking waste collection, and tracking.
    - **Booking Form Page:** Form fields for date, time, waste type, quantity, and address.
    - **Admin Dashboard:** Displays a list of all bookings with options to update statuses.
    - **Popup for Updates:** A modal form to edit booking date and time.
    - **Task assign:** The admin will assign the task to the worker, on collecting the waste from the user.

3. How is Data Stored?

- **Database Design:**
  - **ER Diagram Description:**
    - Tables:
      - **User Table:** Stores user information (e.g., name, email, password).
      - **Booking Table:** Contains booking details like date, time, waste type, quantity, user ID (foreign key), and status.
      - **Waste Type Table:** (Optional) Stores predefined waste types for dropdown selection.
    - Relationships:
      - **User ↔ Booking:** One-to-Many (One user can have multiple bookings).
- **ER Diagram (Explanation):**
  - **User Table:**
    - email (Primary Key)
    - name
    - password
  - **Booking Table:**
    - booking_id (Primary Key)
    - user_id (Foreign Key to User Table)
    - date
    - time
    - waste_type
    - quantity
    - status
  - **Waste Type Table:**
    - type_id (Primary Key)
    - type_name
- **Sample ER Diagram:** *(Describe connections; diagram generation follows if requested)*:
  - **User Table → Booking Table:** email → user_id
  - **Waste Type Table → Booking Table:** type_id → waste_type

5.Testing and Quality Assurance:

1. How Will We Ensure It Works?

- **Testing Approach:**
  - **Unit Testing:**

        - Test individual components like models, views, and API endpoints.
        - Example: Verify that the booking creation logic saves data correctly to the database.
    - **Integration Testing:**
        - Test how different modules interact (e.g., booking form submission to email notification).
        - Example: Ensure the status update triggers an email notification to the user.
    - **System Testing:**
        - Test the entire workflow from user login to booking completion.
        - Example: Validate end-to-end functionality of booking waste collection and tracking status.
    - **Performance Testing:**
        - Simulate high traffic and measure system responsiveness.
        - Example: Check if the system can handle 100 concurrent bookings without slowing down.
    - **Security Testing:**
        - Validate that user data is protected and vulnerabilities are mitigated.
        - Example: Test for SQL injection and ensure sensitive data like passwords are hashed.
    - **User Acceptance Testing (UAT):**
        - Allow stakeholders to verify the system meets expectations.
        - Example: Users test ease of booking and tracking waste collection.
- **Tools for Testing:**
    - **Pytest:** For unit and integration tests in Django.

2. What Makes It Complete?

- **Conditions for Completion:**
    - **Functional Requirements Met:**
        - Users can log in, book waste collection, and receive notifications about status updates.
        - Admins can view and manage bookings efficiently.
    - **Non-Functional Requirements Met:**
        - System is responsive, secure, and scalable.
        - Notifications are sent reliably and quickly.
    - **Testing Criteria:**
        - All unit, integration, and system tests pass.
        - Feedback from UAT is implemented, and no critical bugs remain.
    - **Deployment Ready:**
        - The system is deployed and accessible to users without downtime.

3. When Will We Test?

- **Testing Schedule:**
  - **Unit Testing:** During module development (Weeks 2–3).
  - **Integration Testing:** After critical modules are completed (Weeks 4–5).
  - **System Testing:** Before UAT to validate the complete system (Week 6).
  - **UAT:** Involve stakeholders to test the system in real-world scenarios (Week 7).
- **Continuous Testing in CI/CD:**
  - Automated tests (unit and integration) run on every code commit using GitHub Actions or Jenkins.
  - Regression testing is conducted regularly to ensure stability.

4. What If Something Goes Wrong?

- **Issue Reporting:**
  - Use **GitHub Issues** to report and track bugs.
  - Bugs are logged with detailed information, including steps to reproduce, screenshots, and error logs.
- **Issue Tracking:**
  - Assign a priority level (Critical, High, Medium, Low) to each bug.
  - Track the progress of issue resolution using GitHub Projects.
- **Fixing Process:**
  - Developers fix bugs in a separate branch.
  - The fix is tested locally and in staging before merging into the main branch.
- **Testing Fixes:**
  - Perform regression testing to ensure fixes don’t break existing functionality.
  - Confirm stakeholders are satisfied with resolved issues during UAT.

Example Test Cases

1. **Unit Test for Booking Creation:**
   - **Input:** Valid booking details submitted via the form.
   - **Expected Output:** Booking is saved to the database with the correct status.
2. **Integration Test for Status Update Notification:**
   - **Input:** Admin updates the booking status to "Completed."
   - **Expected Output:** An email is sent to the user with the updated status.
3. **System Test for End-to-End Workflow:**
   - **Scenario:** User logs in, books waste collection, and tracks status until completion.
   - **Expected Output:** Each step functions without errors, and notifications are sent at appropriate times.

4. **Performance Test:**
    - **Scenario:** Simulate 500 users booking waste collection simultaneously.
    - **Expected Output:** System remains responsive, and all bookings are processed.
5. **Security Test:**
    - **Scenario:** Attempt SQL injection via login form.
    - **Expected Output:** Input is sanitized, and no unauthorized access occurs.

6.Deployment and Implementation Plan:

1. Where Will It Live?

- **Deployment Environment:**
    - **Cloud-Based:**
        - Use a cloud service provider like **AWS**, **Google Cloud Platform (GCP)**, or **Azure** for hosting.
        - Deploy the application on **AWS EC2** or **GCP Compute Engine**, ensuring scalability and availability.
    - **Database Hosting:**
        - Host the MySQL database using **Amazon RDS**, **Google Cloud SQL**, or an on-premises setup for better control.
    - **Static Files and Media Storage:**
        - Use **AWS S3** or **GCP Storage** to store user-uploaded files and static assets.
- **Future Growth Considerations:**
    - Implement auto-scaling on the cloud to handle increased user traffic.
    - Adopt microservices architecture if the system needs to expand to include additional services like advanced analytics or IoT integration for waste tracking.

---

2. How Will We Get It There?

- **Deployment Strategy:**
    - **Initial Launch (One-Time):**
        - Deploy the system after all testing phases are complete and feedback from stakeholders is addressed.
    - **Phased Rollout:**
        - Release the system in stages:
            1. **Stage 1:** Deploy basic booking and admin dashboard functionalities.

        2. **Stage 2:** Add notifications and advanced tracking features.
    - **CI/CD Pipeline:**
        - Automate the deployment process using **GitHub Actions**, **Jenkins**, or **CircleCI**.
        - Pipeline steps:
            1. Code push triggers automated tests (unit, integration).
            2. On success, the build is deployed to staging for further validation.
            3. After approval, deploy to production.

---

3. What If Something Goes Wrong?

- **Backup Plans:**
    - **Database Backups:**
        - Set up automated backups for the MySQL database using **AWS RDS snapshots** or **GCP SQL backups**.
        - Maintain a retention policy to store backups for a specific period.
    - **Codebase Backups:**
        - Use version control (Git) to manage and revert to previous commits if necessary.
        - Store critical application configurations in a secure environment, such as **AWS Secrets Manager**.
- **Rollback Strategy:**
    - Implement a **blue-green deployment** approach:
        - Keep the current production environment (blue) active.
        - Deploy updates to a staging environment (green).
        - Switch traffic to the green environment only after successful validation.
        - If issues occur, revert traffic to the blue environment.
    - Use feature toggles to enable or disable specific features without redeploying.

---

4. How Will Users Learn to Use It?

- **Training and Onboarding:**
    - **User Documentation:**
        - Provide a detailed user manual with screenshots and step-by-step instructions for key tasks, such as:
            - Creating a booking.
            - Tracking booking status.
            - Updating booking details.

    - **Video Tutorials:**
        - Create short tutorial videos demonstrating how to use the system.
    - **Interactive Onboarding:**
        - Implement an in-app onboarding experience using tools like **Intro.js** or a custom walkthrough to guide users through key features.
    - **Support System:**
        - Set up a helpdesk or email support system to assist users with queries or technical issues.
- **Admin Training:**
    - Conduct live training sessions for admin users to explain dashboard functionalities and system management.
    - Share best practices for monitoring and updating bookings.

Deployment Checklist

1. **Pre-Deployment:**
    - Conduct final rounds of regression and load testing.
    - Prepare backups for the database and application.
2. **Deployment Phase:**
    - Deploy the system to the production environment.
    - Monitor logs and performance metrics to ensure a smooth transition.
3. **Post-Deployment:**
    - Verify that all features are functioning as intended.
    - Gather feedback from users and address any critical issues immediately.

7.Maintenance and Support Plan:

---

1. Who's in Charge of Keeping It Running?

- **Maintenance Team:**
    - **Developers:**
        - Responsible for fixing bugs, implementing updates, and ensuring feature enhancements.
        - Available to address critical issues or escalations.
    - **System Administrators:**
        - Oversee server health, database backups, and performance optimization.

        - Manage cloud infrastructure and security protocols.
    - **Support Team:**
        - Handle user queries, troubleshoot common issues, and escalate unresolved problems to the development team.
    - **Team Structure:**
        - **Lead Developer:** Oversees system functionality and major updates.
        - **Support Engineers:** Address user complaints and system anomalies.

---

2. What Needs to Be Done to Keep It Running?

- **Regular Maintenance Tasks:**
    - **Software Updates:**
        - Apply patches and updates for the framework (Django), libraries, and server dependencies.
        - Update third-party integrations (e.g., payment gateways, notification services).
    - **Bug Fixes:**
        - Resolve user-reported bugs and perform regression testing after fixes.
    - **Performance Monitoring:**
        - Use tools like **New Relic**, **Google Cloud Monitoring**, or **AWS CloudWatch** to monitor system performance and identify bottlenecks.
        - Optimize database queries and server configurations to maintain speed and scalability.
    - **Database Management:**
        - Regularly clean up old records or redundant data to optimize database size and performance.
        - Perform periodic data validation and integrity checks.
    - **Security Audits:**
        - Conduct vulnerability scans and penetration tests quarterly.
        - Ensure compliance with data protection standards like GDPR, if applicable.
    - **User Support:**
        - Resolve issues reported by users and update FAQs or help documents accordingly.

---

3. How Will We Know What Users Think?

- **Feedback Collection:**

    - **In-App Feedback Form:** Allow users to submit suggestions or complaints directly through the platform.
    - **Surveys:** Send periodic surveys via email to gather user opinions on system usability and satisfaction.
    - **Support Interactions:** Track common queries and complaints logged via the helpdesk or email.
- **Feedback Review:**
    - Organize weekly or bi-weekly meetings with the team to review feedback.
    - Prioritize enhancements or fixes based on the severity and frequency of reported issues.
- **Feedback Implementation:**
    - Add features or fixes to a public-facing roadmap to keep users informed about upcoming updates.
    - Use tools like **Trello**, **Asana**, or **JIRA** to manage feedback implementation.

---

4. What Are Our Service Commitments?

- **Service-Level Agreements (SLAs):**
    - **System Uptime:**
        - Commit to 99.9% uptime, ensuring minimal disruptions to the platform.
    - **Response Time for Issues:**
        - **Critical Issues:** (e.g., system down, data loss)
            - Response within 1 hour.
            - Resolution within 4 hours or a workaround provided.
        - **High-Priority Issues:** (e.g., broken functionality, user unable to book)
            - Response within 4 hours.
            - Resolution within 1 business day.
        - **Low-Priority Issues:** (e.g., UI glitches, minor inconveniences)
            - Response within 1 business day.
            - Resolution within 1 week.
    - **Performance Standards:**
        - Page load time should be under 3 seconds for key operations (e.g., booking, status tracking).
- **Incident Management:**
    - Log incidents using tools like **JIRA**, **GitHub Issues**, or **ServiceNow**.
    - Regularly review incident reports to identify trends and prevent future occurrences.

8.Risk Management Plan:

1. What Could Go Wrong?

- **Technical Risks:** System downtime, data breaches, integration failures, scalability issues.
- **Operational Risks:** Incomplete requirements, delayed deployment, user resistance.
- **Financial Risks:** Budget overruns, revenue impact.
- **Environmental Risks:** Natural disasters, power outages.

2. How Can We Prevent Problems?

- Use reliable cloud services, strong security protocols, and fallback mechanisms.
- Conduct stakeholder reviews, follow Agile practices, and allocate contingency budgets.
- Provide user training and optimize resource allocation.

3. What's Our Backup Plan?

- Real-time database backups, rollback mechanisms, disaster recovery protocols, and alternate workflows.

4. Who's Watching for Problems?

- **Risk Management Team:** Includes project manager, tech lead, support lead, and finance manager.
- **Monitoring Tools:** Tools like New Relic, JIRA, and cloud dashboards.

9.Security and Privacy:

1. Data Protection

- **Encryption:** Use SSL/TLS for data in transit and AES-256 for data at rest.
- **Access Controls:** Implement role-based access controls (RBAC) to restrict sensitive data access.
- **Compliance:** Ensure the system adheres to relevant regulations like GDPR for user data protection.

2. Security Measures

- **Firewalls:** Use cloud-based firewalls and web application firewalls (WAF).
- **Intrusion Detection:** Deploy monitoring tools to detect and alert on suspicious activity.
- **Audits:** Conduct regular security audits and vulnerability assessments.

3. Incident Response Plan

- **Detection:** Use real-time monitoring to identify breaches promptly.
- **Containment:** Isolate affected systems to prevent the spread of breaches.
- **Recovery:** Restore systems using backups and patch vulnerabilities.
- **Notification:** Inform stakeholders and users promptly in compliance with regulations.

10.Legal and Compliance:

1. Licensing

- Use open-source software under permissible licenses like MIT or Apache.
- Ensure compliance with proprietary software licensing agreements for tools and libraries.
- Verify hardware licenses or certifications required for deployment environments.

2. Regulatory Compliance

- Follow **GDPR** for user data protection and **ISO 27001** for information security management.
- Adhere to environmental regulations relevant to waste management, if applicable.
- Regularly audit systems to meet compliance standards.

3. Intellectual Property

- Protect project components with copyrights for code and documentation.
- Register trademarks for branding and unique project identifiers.
- Explore patents for innovative features or processes within the system.

11.Environmental Impact Assessment:

1. Sustainability

- **Energy Consumption:** Use cloud services with a strong commitment to renewable energy sources (e.g., AWS, Google Cloud).
- **Waste Generation:** Minimize waste through digital processes and paperless communication. Encourage e-waste recycling practices in hardware usage.

2. Green IT Practices

- **Energy-efficient Hardware:** Use energy-efficient servers, low-power devices, and optimize infrastructure to reduce power consumption.
- **Virtualized Infrastructure:** Leverage cloud-based virtualization for efficient resource use and reduce the need for physical servers, leading to lower energy consumption.

- **Optimized Coding:** Optimize code to improve performance and reduce computational resource usage.

12. User Documentation:

1. User Manuals

- Provide a **comprehensive user manual** that includes step-by-step instructions on how to use the system.
- Include sections for **common tasks**, such as booking requests, tracking waste, updating user details, and using notifications.
- Ensure the manual is available in **printable** and **digital formats** (e.g., PDF).

2. Online Help and Tutorials

- Create an **online FAQ** page addressing common questions and troubleshooting tips.
- Develop **tutorials** (both written and video formats) covering key system functions, such as how to create and manage bookings or track status updates.
- Offer a **video guide** that provides a **visual walkthrough** of the system, including user interface and key features.

3. User Interface Design

- Focus on an **intuitive UI** that ensures users can navigate the system with minimal learning curve.
- Include **clear icons, labels, and tooltips** to guide users through tasks.
- Provide **responsive design** for both desktop and mobile devices, ensuring accessibility for a wide range of users.
- Ensure **accessibility compliance** (e.g., color contrast, keyboard navigation) to support all users.

13.Project Management and Monitoring:

1. **Project Planning:**
    - Create a detailed project roadmap, breaking down tasks into phases (e.g., planning, development, testing, deployment).
    - Assign timelines and milestones for each phase.
    - Allocate resources like developers, tools, and infrastructure based on the project requirements.

    - Use tools like **Trello**, **JIRA**, or **Asana** for task tracking and collaboration.
2. **Risk Management:**
    - **Identify Risks:** Highlight technical, operational, or resource-based risks, such as delayed timelines, insufficient server capacity, or security vulnerabilities.
    - **Mitigation Strategies:** Develop contingency plans for each risk:
        - Backup systems to prevent data loss.
        - Regular progress reviews to identify and address delays early.
        - Training sessions to handle potential skill gaps in the team.
    - Monitor risks throughout the project using risk assessment tools or regular review meetings.

14. Testing and Quality Assurance:

1. **Unit Testing:**
    - Focus on testing individual components or functions, such as:
        - Views in Django (e.g., `dashboard`, `assign_task`).
        - Models for correct data handling (e.g., `Booking`, `User`).
        - Utility functions like sending emails for status updates.
    - Tools: Django's built-in test framework or manual testing.
2. **Integration Testing:**
    - Verify that components work together as expected:
        - Ensure the frontend properly interacts with backend APIs.
        - Test database interactions (e.g., saving/updating records in `booking` or `AdminTask`).
        - Validate email notifications after status updates.
    - Tools: Django test framework for end-to-end checks or manual testing.