Business Context

Rosla Technologies specializes in **green energy solutions**, **solar panel installation and maintenance**, **EV charging**, and **smart home management**. They want a **digital solution** that allows users to **calculate their carbon footprint, get educational tips, and schedule consultations**.

The goal is to create a **user-friendly platform** that enhances customer engagement and helps users make **sustainable energy decisions**.

Functional Requirements (What the system should do)

1. **User Registration & Authentication**
    - Users can create an account and log in securely.
    - Admin can manage user accounts.
2. **Carbon Footprint Calculator**
    - Users input data about energy usage and transportation.
    - System calculates estimated carbon footprint and suggests tips.
3. **Energy Tracking Tool**
    - Users can log their energy consumption.
    - System provides a **visual representation** (charts, graphs).
4. **Consultation Scheduling**
    - Users can book appointments with Rosla Technologies.
    - System sends email/SMS reminders.
5. **Accessibility Features**
    - Adjustable **font size, contrast settings** for visually impaired users.

Non-Functional Requirements (System qualities)

1. **Scalability** – Must handle increasing users without lag.
2. **Security** – Secure login with **password hashing and HTTPS encryption**.
3. **Performance** – Load pages in under **2 seconds**.
4. **Cross-platform** – Work on **mobile, tablet, and desktop**.

Here’s the functional and non-functional requirements table:

Functional Requirements

Requirement	Priority	Justification
User Registration	High	Allows users to manage consultation data.
Accessibility Features	Medium	Improves usability for users with disabilities.
Carbon Footprint Calculator	High	Essential feature to help users track their impact.
Energy Tracking Tool	High	Provides insights into energy consumption.
Consultation Scheduling	Medium	Allows users to book appointments easily.
Educational Content	Low	Provides tips and essays on reducing carbon footprint.

Non-Functional Requirements

Requirement	Priority	Justification
Security & Data Privacy	High	Ensures user data is protected (GDPR compliant).
Scalability	Medium	Supports an increasing number of users over time.
Performance	High	Ensures fast response times and smooth functionality.
Reliability	High	Ensures system uptime and prevents frequent downtimes.
Usability	Medium	Ensures the application is easy to navigate for all users.
Compliance	High	Adheres to legal and industry standards like WCAG.

Problem Decomposition

Main Problems to Solve

1. **Carbon Footprint Calculation**
    - How to collect relevant user data?
    - What formula to use for accurate results?
2. **Consultation Scheduling**

    - How to handle **time slots, availability**, and avoid double-booking?
3. **User Data Management**

    - How to **securely store** personal information?
    - How to allow **data modification** without security risks?

Key Performance Indicators (KPIs) & User Acceptance Criteria

KPI	Target
Page Load Speed	**<2s**
Consultation Bookings	**100% functional scheduling**
Energy Tracking Accuracy	**>95% accuracy**
Accessibility Score	**WCAG 2.1 Compliance**

User Acceptance Criteria

- Users should be able to **register and log in** smoothly.
- Carbon footprint calculator should provide **realistic** estimates.
- Consultation scheduling should work **without conflicts**.
- Accessibility features should make the site **usable for all**.

Proposed Solution

- **Tech Stack:**

    - **Frontend**: React.js, HTML, CSS
    - **Backend**: Django (Python)
    - **Database**: PostgreSQL
- **How It Meets Client Needs:**

    - Provides a **user-friendly dashboard**.
    - Secure **data management**.
    - Helps customers **reduce their carbon footprint**.

Design Documentation

1. Website Hierarchy Diagram

pgsql
CopyEdit

```
Home
 ├── Carbon Footprint Calculator
 ├── Energy Tracker
 ├── Schedule Consultation
 ├── Login/Register
 ├── User Dashboard
 ├── Accessibility Settings
```

---

2. Entity Relationship Diagram (ERD)

Entities:

- **User (UserID, Name, Email, Password, EnergyData)**
- **Consultation (ConsultationID, UserID, Date, Time, Status)**
- **EnergyData (EntryID, UserID, Usage, Timestamp)**

---

3. Data Requirements & Data Dictionary

Field Name	Data Type	Description
UserID	INT (PK)	Unique User Identifier
Name	VARCHAR(255)	User’s Full Name
Email	VARCHAR(255)	User’s Email Address
Password	VARCHAR(255)	Encrypted Password
EnergyUsage	FLOAT	User’s Energy Consumption

---

4. Data Flow Diagram

Level 0 DFD (High-Level View)

pgsql
CopyEdit

```
User → [Login/Register] → Dashboard → Energy Tracking / Consultation / Carbon Calculator
```

---

5. Use Case Diagram

Actors: **User, Admin**
Use Cases:

- Register & Login
- Calculate Carbon Footprint
- Track Energy Usage
- Schedule Consultation
- Manage Accessibility

Frontend & Backend Needs

Frontend

- **Login/Register UI**
- **Dashboard**
- **Graphs for energy tracking**
- **Accessibility settings (contrast, font size)**

Backend

- **User authentication**
- **Energy data processing**
- **Consultation scheduling logic**
- **API for frontend to retrieve data**

Python Pseudocode (2 Examples)

1. Carbon Footprint Calculation

python
CopyEdit

```
def calculate_carbon_footprint(electricity_usage, transportation_miles, waste_generated):
    footprint = (electricity_usage * 0.5) + (transportation_miles * 0.2) + (waste_generated * 0.3)
    return footprint
```

2. Consultation Booking System

python
CopyEdit

```python
def book_consultation(user_id, date, time):
    if check_availability(date, time):
        save_booking(user_id, date, time)
        return "Booking Confirmed"
    else:
        return "Time Slot Unavailable"
```

Flowcharts (2 Examples)

1. Carbon Footprint Calculation

pgsql
CopyEdit

```
Start
   ↓
Input: Energy Usage, Transportation, Waste
   ↓
Compute Carbon Footprint
   ↓
Display Result
   ↓
End
```

2. Consultation Booking

pgsql
CopyEdit

```
Start
   ↓
User Selects Date & Time
   ↓
Check Availability?
    ├── Yes → Confirm Booking
    └── No → Show Error
   ↓
End
```

Testing Strategy (Tabular Format)

Test Case	Test Type	Expected Outcome
User Login	Black Box	User logs in successfully
Carbon Footprint Calculator	White Box	Correct calculation output
Consultation Booking	Integration	No double-booking
Accessibility Settings	UI Test	Font size and contrast adjust

Regulatory Guidelines & Risk Mitigation

Legal & Compliance

- **GDPR Compliant**: Encrypt user data, allow data deletion.
- **Accessibility Laws**: Ensure WCAG 2.1 compliance.

Risk Mitigation

Risk	Mitigation Strategy
Data Breach	Encrypt user data & use strong authentication
High Traffic Load	Optimize database queries, use caching
Booking Overlaps	Implement a real-time availability checker

This is a **detailed breakdown** of the proposed **digital solution** for Rosla Technologies. Let me know if you need anything **further explained or adjusted!** 🚀

You said:

go into a bit of detail with test strategy

ChatGPT said:

Test Strategy for Rosla Technologies Digital Solution

The test strategy ensures that all components of the digital solution function correctly, meet performance expectations, and comply with security and accessibility standards. It includes different types of testing, a structured testing approach, and specific test cases.

Testing Approach

Testing will be performed using a mix of **manual and automated testing**. The strategy follows a **bottom-up approach**, testing smaller components before integrating them into the full system.

1. **Unit Testing** (Testing individual components)
   - Ensures that each function (e.g., carbon footprint calculation, consultation booking) works correctly.
2. **Integration Testing** (Testing how components work together)
   - Ensures that the frontend can communicate with the backend and database correctly.
   - Example: Verifying that booked consultation slots are properly saved and retrieved.
3. **System Testing** (Testing the full application)
   - Ensures that the entire system meets the functional and non-functional requirements.
   - Checks if a user can register, log in, book consultations, and track energy usage.
4. **User Acceptance Testing (UAT)** (Testing from a user's perspective)
   - Ensures that the system is **easy to use** and that it **meets business needs**.
   - Example: Users should be able to **adjust font size** without affecting functionality.
5. **Performance Testing** (Evaluating system speed and load handling)
   - **Load Testing**: Ensures the website can handle multiple users at once.
   - **Stress Testing**: Tests how the system behaves under extreme conditions.
6. **Security Testing** (Ensuring data protection)
   - Tests login security, **password hashing**, and encryption.
   - Ensures protection against **SQL injection, XSS, and CSRF attacks**.
7. **Accessibility Testing** (Ensuring compliance with WCAG 2.1)
   - Tests if users with disabilities can **navigate the system properly**.
   - Ensures compatibility with **screen readers** and **color contrast settings**.

Test Cases & Expected Results (Tabular Format)

Test Case	Test Type	Description	Expected Outcome

User Registration	Functional (Black Box)	Verify that a user can create an account with valid credentials.	User account is created, and email confirmation is sent.
User Login	Security	Check if incorrect passwords prevent login and if correct credentials allow access.	Incorrect password is rejected, correct password grants access.
Password Encryption	Security	Ensure that passwords are stored in hashed format.	Passwords are hashed and not stored in plain text.
Carbon Footprint Calculator	Functional (White Box)	Verify that the calculation formula provides the correct carbon footprint estimate.	Correct values are displayed based on user input.
Consultation Booking	Integration	Check if a user can book an appointment without overlapping existing bookings.	Available slots allow booking, unavailable slots show an error message.
Consultation Rescheduling	Functional	Verify if a user can change the date/time of a consultation.	Booking updates correctly without duplicating entries.
Consultation Cancellation	Functional	Ensure a user can cancel an existing booking.	Appointment is removed from the database.
Data Validation on Forms	Functional (Black Box)	Check if fields (e.g., email, phone number) accept only valid data.	System prevents invalid entries and provides error messages.
Energy Tracking System	Integration	Verify if users can log their energy usage daily and retrieve historical data.	Entries are saved correctly and displayed in user dashboards.
Energy Graph Rendering	UI	Ensure that charts display the correct energy consumption trends.	Graph updates correctly based on user data.
Mobile Compatibility	UI	Test if the platform is fully responsive on mobile devices.	All elements are properly displayed on mobile screens.
Accessibility: Font Resizing	UI	Check if users can increase or decrease font size.	Text scales correctly without breaking layout.

Accessibility: Contrast Mode	UI	Test if high-contrast mode works for visually impaired users.	Dark mode/high-contrast mode is applied properly.
Database Security	Security	Ensure that unauthorized users cannot access database records.	Only authorized users can access their own data.
SQL Injection Protection	Security	Attempt to inject SQL commands in input fields.	System prevents SQL injection attacks.
XSS Protection	Security	Insert a script into an input field to check if the system allows execution.	The system sanitizes inputs and prevents script execution.
Stress Test (1000 users at once)	Performance	Check how the system handles a large number of concurrent users.	System remains stable with minimal lag.
Load Test (Peak Usage)	Performance	Test performance under peak traffic conditions.	System remains functional without crashes.
API Response Time	Performance	Measure API response times for key requests (login, booking, etc.).	API responds within 500ms for standard requests.
Session Timeout	Security	Ensure that inactive users are logged out after a certain time.	Users are logged out after 15 minutes of inactivity.

Testing Tools

To efficiently test the application, the following tools will be used:

- **Selenium** – Automated UI testing
- **JUnit/PyTest** – Unit testing for backend logic
- **OWASP ZAP** – Security testing tool
- **JMeter** – Load and stress testing
- **Lighthouse** – Accessibility and performance testing

Testing Execution Plan

Phase 1: Unit Testing

- Developers test **individual functions** (e.g., login, booking).
- **Automated tests** for backend logic.

Phase 2: Integration Testing

- Test **database interactions**.
- Ensure frontend and backend communicate correctly.

Phase 3: System Testing

- Full end-to-end testing of **user workflows**.

Phase 4: Performance & Security Testing

- Simulate **heavy traffic**.
- Conduct **penetration testing** for vulnerabilities.

Phase 5: User Acceptance Testing (UAT)

- Involve **real users** to test usability and satisfaction.
- Gather **feedback and fix issues**.