# Instructions For Project

### Project Evergreen: Basic Instruction

**Team:**

- **Abhay:** Lead on RoomScape AI & GreenScore (Primarily Frontend/Mobile App)
- **Tejaswini & Mayank:** Leads on Waste-Knot (Primarily Backend, Data, and AI Logic)

---

### Step 0: Common Ground (Everyone Does This First!)

1. **Setup a Git Repository:** Use GitHub or GitLab. Create a `main` branch.
    - Abhay, create a branch called `feature/user-experience`.
    - Tejaswini and Mayank, create a branch called `feature/backend-logic`.
2. **Agree on the API:** This is the most important step. Below is the official starting point for this API specification. Use this as your contract. If you need to change or add something, discuss it as a team first!
    - **Do this first! It will save you days of headaches.**

---

### Part 1: Instructions for Abhay (RoomScape AI & GreenScore)

Your focus is everything the main shopper sees and interacts with in the mobile app.

#### Phase 1: The GreenScore Feature

1. **Build the UI:** Using **React Native**, create the basic UI components. Don't worry about real data yet.
    - Create a "Product Details" screen. Add a placeholder for the GreenScore (e.g., a circle with the number "85").
    - Create a "User Profile" screen. Add a placeholder for the user's total score and a section for badges.
2. **Make it Work with Fake Data:** Create a local JSON file in your app with a few sample products and a sample user. Make your UI read from this fake data.
3. **Connect to the Real API:** Once Tejaswini and Mayank have the basic API endpoints ready, swap out your fake data for real `fetch()` calls to their server.

#### Phase 2: The RoomScape AI Feature

1. **Research AR in React Native:** Look into libraries like `react-native-vision-camera` combined with a 3D rendering library like `react-native-three.js`.

2. **Hello, Cube!:** Your first goal is to **place a simple 3D cube on a flat surface** using the phone's camera.
3. **Find 3D Models:** Go to an open-source 3D model site like **Sketchfab** and download a few free furniture models in `.glTF` format.
4. **Place Furniture:** Modify your "Hello, Cube!" code to load and place a 3D model of a chair instead of a cube. The URL for the model will come from the API.
5. **Build the UI:** Create the UI around the AR view (buttons to browse items, etc.).

**Key API Endpoints You Will Need to Use:**

- **For Authentication:**
  - `POST /api/auth/login`: To log the user in.
  - `POST /api/auth/register`: To create a new user account.
- **For User Data:**
  - `GET /api/auth/me`: To get the logged-in user's profile, including their `cumulative_green_score`.
- **For Product Data:**
  - `GET /api/products`: To get a list of products to show in a list or grid.
  - `GET /api/products/:id`: To get all the details for one specific product, including its `green_score`, `green_score_details`, and the crucial `model_3d_url` for RoomScape.

---

### Part 2: Instructions for Tejaswini & Mayank (Waste-Knot)

Your focus is the "brain" of the project: the server, the database, and the logic for donations.

**Phase 1: Build the Core API and Database**

1. **Setup the Backend:** Use **Node.js** with the **Express.js** framework. This will be your server.
2. **Create the Database Tables:** Connect your Express app to a **PostgreSQL** database. Use a library like **Sequelize** or **Prisma** to make this easier. Run the following SQL commands to create your tables. This is the foundation for everything.

```sql
-- Users Table: Stores login info for everyone. The 'role' is critical for permissions.
CREATE TYPE user_role AS ENUM ('shopper', 'store_manager', 'charity_rep', 'admin');

CREATE TABLE Users (
    id SERIAL PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,
    full_name VARCHAR(255),
```

```
    role user_role NOT NULL DEFAULT 'shopper',
    cumulative_green_score INT NOT NULL DEFAULT 0,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT aurootc(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT aurootc()
);

-- Products Table: Central catalog for all items.
CREATE TABLE Products (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    price NUMERIC(10, 2) NOT NULL,
    image_url TEXT,
    green_score INT CHECK (green_score >= 0 AND green_score <= 100),
    green_score_details JSONB, -- e.g., {'packaging': 10, 'sourcing': 8,
'carbon_footprint': 7}
    model_3d_url TEXT, -- Link to the .glTF file for RoomScape
    category VARCHAR(100),
    inventory_quantity INT NOT NULL DEFAULT 0,
    expiry_date DATE -- This will be NULL for non-perishable items
);

-- Charities Table: Stores info about partner organizations.
CREATE TABLE Charities (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    address TEXT,
    contact_email VARCHAR(255),
    is_verified BOOLEAN NOT NULL DEFAULT false, -- An admin must approve
them
    user_id INT UNIQUE REFERENCES Users(id) ON DELETE SET NULL -- A
charity is managed by one user
);

-- Donation_Manifests Table: The core of the Waste-Knot workflow.
CREATE TYPE manifest_status AS ENUM ('pending_approval', 'available',
'claimed', 'completed', 'cancelled');

CREATE TABLE Donation_Manifests (
    id SERIAL PRIMARY KEY,
    status manifest_status NOT NULL DEFAULT 'pending_approval',
    created_by_user_id INT REFERENCES Users(id),
    claimed_by_charity_id INT REFERENCES Charities(id),
```

```
    scheduled_pickup_time TIMESTAMP WITH TIME ZONE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT aurootc(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT aurootc()
);

-- Manifest_Items Table: Linking table showing which products are in
which manifest.
CREATE TABLE Manifest_Items (
    id SERIAL PRIMARY KEY,
    manifest_id INT NOT NULL REFERENCES Donation_Manifests(id) ON DELETE
CASCADE,
    product_id INT NOT NULL REFERENCES Products(id),
    quantity INT NOT NULL CHECK (quantity > 0),
    UNIQUE (manifest_id, product_id)
);
```

3. **Build the API Endpoints:** Create the following API routes in Express. Use middleware to protect routes and check user roles (e.g., `isStoreManager`). Start by making them return fake data to unblock Abhay, then connect them to your database.

   **Group 1: Authentication & User Routes**

   - **`POST /api/auth/register`** (Body: `{ email, password, fullName, role }`)
   - **`POST /api/auth/login`** (Body: `{ email, password }`)
   - **`GET /api/auth/me`** (Protected: Requires JWT)

   **Group 2: Products & GreenScore Routes**

   - **`GET /api/products`** (Supports query params like `?category=furniture`)
   - **`GET /api/products/:id`**

   **Group 3: Waste-Knot Donation Routes**

   - **For Store Managers:**
     - `GET /api/donations/pending` (Protected: Store Manager role)
     - `POST /api/donations/:manifestId/approve` (Protected: Store Manager role)
   - **For Charity Reps:**
     - `GET /api/donations/available` (Protected: Charity Rep role)
     - `POST /api/donations/:manifestId/claim` (Protected: Charity Rep role)
   - **General:**
     - `GET /api/donations/:manifestId` (Protected)

   **Group 4: Admin Routes (For you to manage data)**

   - `POST /api/admin/products` (Protected: Admin role)

- `PUT /api/admin/products/:id` (Protected: Admin role)
- `POST /api/admin/charities/:id/verify` (Protected: Admin role)

4. **Populate with Seed Data:** Create a script that fills your new database tables with some sample products, users, and charities so you have data to test with.

**Phase 2: The Waste-Knot Logic ("The AI")**

1. **The "AI" Model (Rule-Based Algorithm):** Write a function in your Node.js backend that:
    1. Queries the `Products` table.
    2. Finds items where `expiryDate` is near and `inventory_quantity` is high.
    3. Groups these items and creates a new entry in the `Donation_Manifests` and `Manifest_Items` tables with a `status` of 'pending_approval'.
2. **Automate It:** Use a library like `node-cron` to schedule your function to run automatically once every day.
3. **Build the Manager & Charity Views:** Create two simple web pages using a templating engine like **EJS** in Express.
    - **Manager Page:** Fetches from `/api/donations/pending` and has an "Approve" button that calls the approve endpoint.
    - **Charity Page:** Fetches from `/api/donations/available` and has a "Claim" button that calls the claim endpoint.

---

### Work Division for Backend Team (Tejaswini & Mayank)

**Overall Team Goal:** Build a robust, secure, and functional backend server that provides the necessary API for the mobile app and powers the entire Waste-Knot system.

**Guiding Principles:**

- **Pair Program on Setup:** Work together on the initial project setup to ensure you both understand the foundation.
- **Own Your Modules:** Each person is the primary owner of their assigned modules. You are responsible for the code, but you should still review each other's work.
- **Communicate Constantly:** Talk daily. Mayank's work depends on Tejaswini's, and vice-versa. Use pull requests for code reviews.

---

### Phase 1: Foundation (Work Together)

**Goal:** Get the project skeleton up and running.
**Tasks (Do these together):**

1. **Initialize the Node.js/Express Project:** Set up the basic server file (`index.js` or `server.js`), install Express, and create the basic folder structure (`/routes`, `/controllers`, `/models`, etc.).

2. **Database Setup:**
    - Create the PostgreSQL database.
    - Connect the Express application to the database using an ORM like **Sequelize** or **Prisma**.
    - **Write the migration/schema files for ALL the tables together.** This is critical. You both need to agree on the exact structure of the `Users`, `Products`, `Charities`, `Donation_Manifests`, and `Manifest_Items` tables before splitting up.
3. **Authentication Scaffolding:** Set up the basic JWT strategy (e.g., using `passport.js` or a custom middleware). Create the empty functions for `register`, `login`, and the middleware to check for a valid token.

---

### Tejaswini's Primary Responsibilities: The "Public-Facing" Backend & Core Logic

**Module Ownership: User & Product API and the "AI" Donation Creation Logic.**
Your focus is on providing the data Abhay needs for the mobile app and creating the core logic that identifies waste.

**Specific Tasks:**

1. **Implement Authentication & User Endpoints:**
    - Flesh out the logic for `POST /api/auth/register` and `POST /api/auth/login`. This involves hashing passwords and generating JWTs.
    - Implement the `GET /api/auth/me` endpoint to return the profile of the logged-in user.
2. **Implement Product Endpoints:**
    - Build the `GET /api/products` endpoint, including logic for filtering by category and pagination.
    - Build the `GET /api/products/:id` endpoint to return detailed information for a single product. This is a high-priority task to unblock Abhay.
3. **Develop the "AI" Waste Identification Engine:**
    - Write the core function that queries the `Products` table to find items nearing expiry.
    - This function will then create new entries in the `Donation_Manifests` and `Manifest_Items` tables with the status set to `pending_approval`.
    - Use `node-cron` to schedule this function to run automatically every night. You are the "brain" behind the donation creation process.
4. **Admin Endpoints for Products:**
    - Build the `POST /api/admin/products` and `PUT /api/admin/products/:id` routes for managing the product catalog.

---

### Mayank's Primary Responsibilities: The "Internal-Facing" Backend & Workflow Management

**Module Ownership: Donation Workflow Management & The Web Dashboards.**
Your focus is on the entire lifecycle of a donation *after* Tejaswini's script has created it. You will manage the state changes (from pending to approved to claimed) and build the simple web interfaces for store managers and charities.

**Specific Tasks:**

1. **Implement Donation Workflow Endpoints:**
   - Build the `GET /api/donations/pending` endpoint for store managers. It should return a list of manifests waiting for approval.
   - Implement the `POST /api/donations/:manifestId/approve` endpoint. This will update a manifest's status from `pending_approval` to `available`.
   - Build the `GET /api/donations/available` endpoint for charities.
   - Implement the `POST /api/donations/:manifestId/claim` endpoint. This is a critical step that updates the status to `claimed` and links the manifest to the `charity_id`.
   - Build the general `GET /api/donations/:manifestId` endpoint to show the details of any single donation.
2. **Build the Web Dashboards:**
   - Using a templating engine like **EJS**, create a simple, secure web interface.
   - **Store Manager Dashboard:** A page that requires a `store_manager` login. It will fetch data from your `/api/donations/pending` endpoint and display a list with an "Approve" button for each item.
   - **Charity Dashboard:** A page that requires a `charity_rep` login. It will fetch data from your `/api/donations/available` endpoint and display a list with a "Claim" button for each item.
3. **Admin Endpoints for Charities:**
   - Build the `POST /api/admin/charities/:id/verify` route, which allows an admin to approve a new charity so they can start claiming donations.

---

### Summary of Work Division:

| Task Area | Tejaswini (Primary Owner) | Mayank (Primary Owner) |
|---|---|---|
| **Core Logic** | Creates new donations from surplus inventory (The "AI"). | Manages the status and workflow of existing donations. |
| **API for Abhay (Mobile)** | Provides all User and Product data. | (Indirectly) Provides the data for donation history if needed. |
| **API for Web Dashboards** | (Indirectly) Creates the data that Mayank's dashboards will use. | Provides all endpoints needed for the Manager and Charity dashboards. |
| **User Interfaces** | None (API only). | Builds the EJS web dashboards for managers and charities. |

| Task Area | Tejaswini (Primary Owner) | Mayank (Primary Owner) |
| --- | --- | --- |
| **Database Tables** | Focuses on writing to `Donation_Manifests` and `Manifest_Items`. | Focuses on reading from and updating `Donation_Manifests`. |
| **Admin Functions** | Manages Products. | Manages Charities. |

This division ensures that you both have significant, self-contained modules to work on, minimizing blockers while ensuring your work fits together perfectly.