Overview

This document outlines the setup, testing, and usage of the Construction Management System API built using Django. The API facilitates project management, resource management, document management, site monitoring, and safety management.

Table of Contents

1. Testing the API

Using Postman

1. Download and install Postman.
2. Create a new request.
   - Enter the URL for the API endpoint (e.g., `http://127.0.0.1:8000/api/projects/`).
   - Select the HTTP method (GET, POST, PUT, DELETE).
   - Send the request and observe the response.

Using cURL

Use the command line to test the API with the following examples:

- **GET All Projects:**

```
curl -X GET http://127.0.0.1:8000/api/projects/
```

- **Create a New Project:**

```
curl -X POST http://127.0.0.1:8000/api/projects/ -H "Content-Type:
application/json" -d '{"name": "New Project", "location": "Location A",
"budget": 100000.00, "timeline": "2024-10-01"}'
```

- **Update a Project:**

```
curl -X PUT http://127.0.0.1:8000/api/projects/{project_id}/ -H
"Content-Type: application/json" -d '{"name": "Updated Project",
"location": "Location B", "budget": 150000.00, "timeline": "2024-12-
01"}'
```

- **Delete a Project:**

```
curl -X DELETE http://127.0.0.1:8000/api/projects/{project_id}/
```

---

2. API Endpoints

Project Endpoints

1. **GET** `/api/projects/`: Retrieve all projects.
2. **POST** `/api/projects/`: Create a new project.
3. **PUT** `/api/projects/{project_id}/`: Update an existing project.
4. **DELETE** `/api/projects/{project_id}/`: Delete a project.

Task Endpoints

1. **GET** `/api/tasks/`: Retrieve all tasks.
2. **POST** `/api/tasks/`: Create a new task.
3. **PUT** `/api/tasks/{task_id}/`: Update an existing task.
4. **DELETE** `/api/tasks/{task_id}/`: Delete a task.

Material Endpoints

1. **GET** `/api/materials/`: Retrieve all materials.
2. **POST** `/api/materials/`: Create a new material.
3. **PUT** `/api/materials/{material_id}/`: Update an existing material.
4. **DELETE** `/api/materials/{material_id}/`: Delete a material.

Worker Endpoints

1. **GET** `/api/workers/`: Retrieve all workers.
2. **POST** `/api/workers/`: Create a new worker.
3. **PUT** `/api/workers/{worker_id}/`: Update an existing worker.
4. **DELETE** `/api/workers/{worker_id}/`: Delete a worker.

Equipment Endpoints

1. **GET** `/api/equipment/`: Retrieve all equipment.
2. **POST** `/api/equipment/`: Create new equipment.
3. **PUT** `/api/equipment/{equipment_id}/`: Update existing equipment.
4. **DELETE** `/api/equipment/{equipment_id}/`: Delete equipment.

Document Endpoints

1. **GET** `/api/documents/`: Retrieve all documents.
2. **POST** `/api/documents/`: Upload a new document.
3. **PUT** `/api/documents/{document_id}/`: Update a document.
4. **DELETE** `/api/documents/{document_id}/`: Delete a document.

Daily Log Endpoints

1. **GET** `/api/daily-logs/`: Retrieve all daily logs.
2. **POST** `/api/daily-logs/`: Create a new daily log.
3. **PUT** `/api/daily-logs/{log_id}/`: Update an existing daily log.
4. **DELETE** `/api/daily-logs/{log_id}/`: Delete a daily log.

Incident Endpoints

1. **GET** `/api/incidents/`: Retrieve all incidents.
2. **POST** `/api/incidents/`: Report a new incident.
3. **PUT** `/api/incidents/{incident_id}/`: Update an incident.
4. **DELETE** `/api/incidents/{incident_id}/`: Delete an incident.

---

3. Troubleshooting

1. **Server Errors:** Check the terminal for error messages.
2. **Request Failures:** Verify the URL and method.
3. **Response Issues:** Ensure that your request body is correctly formatted.

---

4. Models

Project Model

- **Attributes:**
  - `id`: Unique identifier (AutoField)
  - `name`: Project name (CharField)

    - `location`: Project location (CharField)
    - `budget`: Estimated budget (DecimalField)
    - `timeline`: Project duration (DateField)
    - `start_date`: Actual start date (DateField)
    - `end_date`: Actual end date (DateField)
    - `status`: Current status (CharField with choices)
    - `created_at`: Creation timestamp (DateTimeField)
    - `updated_at`: Last update timestamp (DateTimeField)

Relationships:

- One project can have many tasks, workers, materials, documents, daily logs, and incidents.

Task Model

- **Attributes:**
    - `id`: Unique identifier (AutoField)
    - `project`: Foreign key to Project model (ForeignKey)
    - `name`: Task name (CharField)
    - `description`: Task description (TextField)
    - `status`: Task status (CharField with choices)
    - `assigned_worker`: Foreign key to Worker model (ForeignKey)
    - `due_date`: Task deadline (DateField)
    - `created_at`: Creation timestamp (DateTimeField)
    - `updated_at`: Last update timestamp (DateTimeField)

Material Model

- **Attributes:**
    - `id`: Unique identifier (AutoField)
    - `project`: Foreign key to Project model (ForeignKey)
    - `type`: Type of material (CharField)
    - `quantity`: Required quantity (IntegerField)
    - `unit`: Unit of measurement (CharField)
    - `status`: Availability status (CharField with choices)
    - `created_at`: Entry timestamp (DateTimeField)

Worker Model

- **Attributes:**
    - `id`: Unique identifier (AutoField)
    - `name`: Worker name (CharField)
    - `role`: Worker role (CharField)
    - `contact_info`: Contact details (CharField)
    - `project`: Foreign key to Project model (ForeignKey)

    - `created_at`: Entry timestamp (DateTimeField)

Equipment Model

- **Attributes:**
    - `id`: Unique identifier (AutoField)
    - `type`: Type of equipment (CharField)
    - `status`: Current status (CharField with choices)
    - `project`: Foreign key to Project model (ForeignKey)
    - `created_at`: Entry timestamp (DateTimeField)

Document Model

- **Attributes:**
    - `id`: Unique identifier (AutoField)
    - `title`: Document title (CharField)
    - `upload`: File upload field (FileField)
    - `project`: Foreign key to Project model (ForeignKey)
    - `created_at`: Upload timestamp (DateTimeField)

---

5. Views

Project Views

- **List View:** Display all projects.
- **Detail View:** Show detailed information about a specific project.
- **Create View:** Provide a form to create a new project.
- **Update View:** Allow users to edit project details.
- **Delete View:** Confirm and delete a project.

Task Views

- Similar structure as Project views.

Material Views

- Manage materials linked to projects.

Worker Views

- Manage worker information.

Equipment Views

6.1 User Model

- **Default User Model:** Utilize Django's built-in User model (`django.contrib.auth.models.User`), which includes fields such as:
    - `username`
    - `password`
    - `email`
    - `first_name`
    - `last_name`
- **Custom User Model (optional):** If additional fields are required, extend the `AbstractUser` model to create a custom user model.

6.2 Configuring User Management in the Admin Panel

- **Admin Registration:** Register the User model in the admin panel to manage users directly.
- **Admin Customization:**
    - Define list displays, filters, and search fields for easier user management.
    - Use the `UserAdmin` class for enhanced functionality, allowing inline management of user groups and permissions.

6.3 User Authentication Views

1. **User Registration View:**
    - **Functionality:** Create a view for new user registration.
    - **Form:** Collect necessary user information and validate it.
    - **Post-registration:** Redirect to the login page or automatically log the user in.
2. **User Login View:**
    - **Functionality:** Implement a login view that authenticates users.
    - **Django Built-in:** Use `LoginView` for simplicity.
    - **Redirect:** Upon successful login, redirect users to a dashboard or homepage.
3. **User Logout View:**
    - **Functionality:** Use `LogoutView` to handle user logout.
    - **Outcome:** Clear the user session and redirect to the login page.
4. **Profile Management View:**
    - **Functionality:** Allow users to manage their profiles.
    - **Details:** Users can update information such as email, password, and personal details.
    - **Validation:** Ensure form validation to maintain data integrity.

6.4 Permissions and Access Control

- **User Permissions:** Utilize Django’s permissions framework to control user access to different parts of the application.
- **Role-based Access Control:**
    - Assign permissions to users or groups based on their roles (e.g., project managers, workers).

- **Group Management:**
    - Create groups to simplify permission management. Users can be assigned to one or more groups.

6.5 Password Management

1. **Password Reset View:**
    - **Functionality:** Implement a view for users to reset their password if forgotten.
    - **Process:** Send a password reset email with a link to set a new password.
2. **Password Change View:**
    - **Functionality:** Allow logged-in users to change their passwords securely.

6.6 User Registration Workflow

- **Workflow Steps:**
    1. Form submission by the admin or user.
    2. Data validation.
    3. User creation and confirmation.
    4. Optionally, send a welcome email.

6.7 Testing Authentication Features

- Thoroughly test all authentication features:
    - Valid and invalid credentials.
    - Duplicate usernames.
    - Successful and unsuccessful login/logout behavior.

7. Authentication

7.1 User Model

- **Default User Model:** Utilize Django's built-in User model (`django.contrib.auth.models.User`), which includes fields such as:
    - `username`
    - `password`
    - `email`
    - `first_name`
    - `last_name`
- **Custom User Model (optional):** If additional fields are required, extend the `AbstractUser` model to create a custom user model.

7.2 Configuring User Management in the Admin Panel

- **Admin Registration:** Register the User model in the admin panel to manage users directly.
- **Admin Customization:**

    - Define list displays, filters, and search fields for easier user management.
    - Use the `UserAdmin` class for enhanced functionality, allowing inline management of user groups and permissions.

7.3 User Authentication Views

1. **User Registration View:**
    - **Functionality:** Create a view for new user registration.
    - **Form:** Collect necessary user information and validate it.
    - **Post-registration:** Redirect to the login page or automatically log the user in.
2. **User Login View:**
    - **Functionality:** Implement a login view that authenticates users.
    - **Django Built-in:** Use `LoginView` for simplicity.
    - **Redirect:** Upon successful login, redirect users to a dashboard or homepage.
3. **User Logout View:**
    - **Functionality:** Use `LogoutView` to handle user logout.
    - **Outcome:** Clear the user session and redirect to the login page.
4. **Profile Management View:**
    - **Functionality:** Allow users to manage their profiles.
    - **Details:** Users can update information such as email, password, and personal details.
    - **Validation:** Ensure form validation to maintain data integrity.

7.4 Permissions and Access Control

- **User Permissions:** Utilize Django’s permissions framework to control user access to different parts of the application.
- **Role-based Access Control:**
    - Assign permissions to users or groups based on their roles (e.g., project managers, workers).
- **Group Management:**
    - Create groups to simplify permission management. Users can be assigned to one or more groups.

7.5 Password Management

1. **Password Reset View:**
    - **Functionality:** Implement a view for users to reset their password if forgotten.
    - **Process:** Send a password reset email with a link to set a new password.
2. **Password Change View:**
    - **Functionality:** Allow logged-in users to change their passwords securely.

7.6 User Registration Workflow

- **Workflow Steps:**
    1. Form submission by the admin or user.

2. Data validation.
3. User creation and confirmation.
4. Optionally, send a welcome email.

7.7 Testing Authentication Features

- Thoroughly test all authentication features:
    - Valid and invalid credentials.
    - Duplicate usernames.
    - Successful and unsuccessful login/logout behavior.