Civil Tech Manual

Front end Domain: <https://app.covaiciviltechlab.com/>

Backend Domain: <https://files.covaiciviltechlab.com/dashboard/>

# Apps:

* Account
* General
* Payment

## Account App:

The "account" app is designed to manage various entities related to user accounts, specifically focusing on customers and employees. It provides a structured way to store and retrieve essential information such as personal details, contact information, and addresses.

Key Features:

1. **Customer Management**: The app allows for detailed record-keeping of customer information, including multiple addresses and contact persons. Each customer can be associated with cities, states, and countries, enhancing location tracking and management.
2. **Employee Management**: User accounts for employees include vital details such as personal information, qualifications, and salary. The app supports tracking of employee data with the option to upload signatures and specify roles within the organization.
3. **Audit Trails**: With fields for tracking who created or modified records, the app maintains an audit trail that ensures accountability and transparency in data management.

## General App:

The "general" app serves as a comprehensive module designed to manage essential business resources, including tax configurations, print formats, materials, report templates, tests, and expenses. This app aims to streamline various operational aspects, providing a structured approach to data management.

Key Features:

1. **Tax Management**: Employee can define and manage different tax types, set their percentages, and control their activation status. This ensures compliance and facilitates financial planning.
2. **Print Formats**: The app allows the creation and management of print formats, enabling customization of documents and reports to align with organizational branding.
3. **Material Handling**: Materials can be cataloged with unique names and associated details, along with related templates and logos. This is crucial for organizations that need to manage various materials used in their operations.
4. **Test Management**: The app includes functionality to define tests, including pricing and associated materials.
5. **Expense Tracking**: A dedicated model for managing expenses ensures that all financial outflows are recorded and can be easily referenced for budgeting and analysis.
6. **User Management**: All models include tracking who created or modified records, ensuring accountability, and an audit trail for all changes.

## Payment App:

The "payment" app is a robust module designed to manage all aspects of financial transactions and record-keeping within an organization. It integrates seamlessly with other applications like "account" and "general," allowing for efficient handling of invoices, payments, expenses, and related documentation.

Key Features:

1. **Invoice Management**: The app enables the creation and tracking of invoices linked to customers and sales modes. Users can manage discounts, advance payments, and balances, ensuring accurate financial records.
2. **Payment Tracking**: Various payment modes (cash, cheque, UPI, NEFT, TDS) are supported, allowing for detailed tracking of payment methods and associated information. This helps in maintaining a clear overview of incoming funds.
3. **Test Invoicing**: The app supports invoicing for specific tests, linking them to invoices and capturing necessary details like quantity, pricing, and completion status. This is particularly useful for laboratories and service providers.
4. **Expense Management**: Users can log and categorize expenses, providing insight into financial outflows. This feature helps in budgeting and financial analysis.
5. **File Management**: The ability to upload and associate files with invoices and expenses ensures that all relevant documentation is easily accessible. This includes receipts, invoices, and other supporting documents.
6. **Tax Management**: Integration with the general app allows for the application of different tax structures (CGST, SGST, IGST) on invoices, ensuring compliance with taxation regulations.
7. **Comprehensive Reporting**: The app includes properties that calculate totals, taxes, and other financial metrics, enabling users to generate accurate reports and summaries.
8. **User Accountability**: Each transaction and entry tracks the user responsible for its creation and modification, promoting accountability and transparency.

# Developer Manual:

### Account App

URL Configuration

The API endpoints are defined in urls.py. Below is a summary of the available routes:

| **Endpoint** | **HTTP Method** | **Description** |
| --- | --- | --- |
| /login/ | POST | User login |
| /logout/ | GET | User logout |
| /create\_customer/ | POST | Create a new customer |
| /customer\_list/ | GET | Retrieve a list of customers |
| /edit\_customer/<int:id>/ | GET, PUT, DELETE | Manage a specific customer |
| /create\_employee/ | POST | Create a new employee |
| /employee\_list/ | GET | Retrieve a list of employees |
| /edit\_employee/<int:id>/ | GET, PUT, DELETE | Manage a specific employee |
| /change-password/ | POST | Change user password |
| /create\_city/ | POST | Create a new city |
| /city\_list/ | GET | Retrieve a list of cities |
| /edit\_city/<int:id>/ | GET, PUT, DELETE | Manage a specific city |

View Classes

The views are implemented in views.py using Django REST Framework's APIView. Below are some key view classes:

* **Login\_View**: Handles user authentication and returns a token.
* **Logout**: Deletes the user's authentication token.
* **Create\_Customer**: Allows creation of new customers and fetching related city, state, and country data.
* **Manage\_Customer**: Supports CRUD operations for existing customers.
* **Manage\_Employee**: Supports CRUD operations for employees.
* **Manage\_City**: Handles city-related CRUD operations.
* **ChangePasswordView**: Allows users to change their passwords.

Serializers

Serializers are defined in serializers.py and are used for validating and transforming data. Key serializers include:

* **LoginSerializer**: Validates user credentials.
* **UserSerializer**: Handles user creation and updates.
* **Create\_Customer\_Serializer**: Validates customer data for creation.
* **Customer\_Serializer**: Transforms customer data for responses.
* **ChangePasswordSerializer**: Validates password change requests.

### General App

URL Configuration

The API endpoints are defined in urls.py. Below is a summary of the available routes:

| **Endpoint** | **HTTP Method** | **Description** |
| --- | --- | --- |
| /create\_tax/ | POST | Create a new tax |
| /tax\_list/ | GET | Retrieve a list of taxes |
| /edit\_tax/<int:id>/ | GET, PUT, DELETE | Manage a specific tax |
| /create\_material/ | POST | Create a new material |
| /material\_list/ | GET | Retrieve a list of materials |
| /edit\_material/<int:id>/ | GET, PUT, DELETE | Manage a specific material |
| /create\_report\_template/ | POST | Create a new report template |
| /report\_template\_list/ | GET | Retrieve a list of report templates |
| /edit\_report\_template/<int:id>/ | GET, PUT, DELETE | Manage a specific report template |
| /create\_test/ | POST | Create a new test |
| /test\_list/ | GET | Retrieve a list of tests |
| /edit\_test/<int:id>/ | GET, PUT, DELETE | Manage a specific test |
| /create\_expense/ | POST | Create a new expense |
| /expense\_list/ | GET | Retrieve a list of expenses |
| /edit\_expense/<int:id>/ | GET, PUT, DELETE | Manage a specific expense |

View Classes

The views are implemented in views.py using Django REST Framework's APIView. Below are some key view classes:

* **Manage\_Tax**: Handles CRUD operations for taxes.
* **Manage\_Material**: Manages materials, including retrieval and creation.
* **Create\_Report\_Template**: Manages report template creation.
* **Create\_Test**: Handles the creation and listing of tests.
* **Manage\_Expense**: Supports CRUD operations for expenses.

**Serializers**

Serializers are defined in serializers.py and are used for validating and transforming data. Key serializers include:

* **Create\_Tax\_Serializer**: Validates data for creating a new tax.
* **Tax\_Serializer**: Transforms tax data for API responses.
* **Create\_Material\_Serializer**: Validates material creation data.
* **Material\_Serializer**: Transforms material data for API responses.
* **Create\_ReportTemplate\_Serializer**: Validates data for creating report templates.
* **Report\_Template\_Serializer**: Transforms report template data for responses.

### Payment App

URL Configuration

The API endpoints are defined in urls.py. Below is a summary of the available routes:

| **Endpoint** | **HTTP Method** | **Description** |
| --- | --- | --- |
| /create\_expense/ | POST | Create a new expense |
| /expense\_list/ | GET | Retrieve a list of expenses |
| /edit\_expense/<int:id>/ | GET, PUT, DELETE | Manage a specific expense |
| /create\_invoice/ | POST | Create a new invoice |
| /invoice\_list/ | GET | Retrieve a list of invoices |
| /edit\_invoice/<int:id>/ | GET, PUT, DELETE | Manage a specific invoice |
| /create\_sales\_mode/ | POST | Create a new sales mode |
| /sales\_mode\_list/ | GET | Retrieve a list of sales modes |
| /edit\_sales\_mode/<int:id>/ | GET, PUT, DELETE | Manage a specific sales mode |
| /create\_receipt/ | POST | Create a new receipt |
| /receipt\_list/ | GET | Retrieve a list of receipts |
| /edit\_receipt/<int:id>/ | GET, PUT, DELETE | Manage a specific receipt |

**View Classes**

The views are implemented in views.py using Django REST Framework's APIView. Below are some key view classes:

* **Manage\_Expense**: Handles CRUD operations for expenses.
* **Manage\_Invoice**: Manages invoices, including retrieval and creation.
* **Manage\_Sales\_Mode**: Handles operations related to sales modes.
* **Manage\_Receipt**: Supports CRUD operations for receipts.

**Serializers**

Serializers are defined in serializers.py and are used for validating and transforming data. Key serializers include:

* **Create\_Expense\_Entry\_Serializer**: Validates data for creating a new expense.
* **Expense\_Entry\_Serializer**: Transforms expense data for API responses.
* **Create\_Invoice\_Serializer**: Validates data for creating a new invoice.
* **Invoice\_Serializer**: Transforms invoice data for API responses.
* **Receipt\_Serializer**: Validates and transforms receipt data.

How Invoice Works

1. **Invoice Number**: Each invoice must generate a unique sequential number.
2. **Invoice QR Code**: When creating an invoice, generate a QR code for it and save the code as part of the invoice.
3. **Invoice Test**:

* **Template Creation**: Start by copying the material template. Add headers (including the civil logo), test order details, customer information, date, testing location, tester's name, and project name. Save this modified template as the test invoice template in the report\_template field.
* **QR Code Generation**: Create a QR code for the test report and store the image on the server. Record the QR code's location in the invoice\_image field.

1. **Template Editing**: Update the invoice test template to indicate whether the test was completed and the name of the person who conducted it.
2. **Printing the Invoice Test Template**: The template can be printed with or without headers. The headerless option removes all headers from the printed version.
3. **Invoice Payments**: Support for multiple payment entries, with each payment recorded individually.
4. **Invoice Taxes**: Ability to assign multiple tax rates to an invoice.
5. **Invoice Discounts**: Discounts can be applied to the invoice.
6. **Invoice Completion**: After the test and payment are finalized, the invoice completion status can be updated.
7. **Invoice Printing**: Use the invoice print serializer to send all relevant invoice data to the front end for printing. The front end will handle the presentation of the invoice.

Report

* Sales Report
* Expense Report
* Expense File Report
* Invoice File Report
* Test Report

Dashboard

**Customer Statistics**:

* Display total customer count.
* Show the count of customers created this month.

**Invoice Payment Section**:

* Present financial year invoice data in a chart.
* Include a separate chart for this month's invoice data.
* List recent invoices in a table format.

**Expense Payment Section**:

* Display financial year expense data in a chart.
* Include a chart for this month's expense data.
* Present recent expenses in a table format.