This is the documentation for the backend design of Meeting Space.

1. Virtual Environment, Framework, Programming Language and Database

The backend uses Pipenv v2018.11.26 in place of the usually used virtualenv (venv), Pipenv configures the venv and combines pip with the virtual environment. Therefore, *pipenv* should be used in place of Pip. Pipfile.lock is to be used to ensure the dependency declarations from the last tested combination. The project is developed on Django 3.0.3 and the rest APIs use Django Rest Framework 3.11.

Postgresql v12.2 has been used as the database for the backend. As a filter backend for Celery 4.4.1, Redis 3.4.1 has been used. This is further explained below. Please note that postgresql has a dependency on psycopg2 v2.8.4 which has been ensured through Pipfile.lock. All the packages that have been used(except for Redis) were the latest version during the time of development.

2. Initial Setup

Postgres database maintains a hierarchy level through *database -> schema -> tables.* The backend has been designed so that data of the entities are stored separately in the form of schemas. The schema name is currently determined by the input of the superadmin, in this case, the people who are reading this documentation right now. To complete the schema name, schema is appended to the back of the name as entered by the superadmin during the creation of the company. This has been described below.

Please note that python v3.8, pipenv v2018.11.26, postgres v12.2 and psycopg2 v2.8.4 must be present in the server, so that no dependencies issues arise. Postgresql server must also be started, a database named *meeting* (The rest of the document will use the *meeting* as the database for the application) should be created. During development, an user *meeting* has been created with a password that has not been included in the documentation for security reasons. All the authentication credentials can be changed by the reader of this document as seen fit. A script with the aforementioned credentials will be provided to the reader of this document unless otherwise mentioned.

Change the directory to (*/newMeetingSpace*), hereby referred to as the root directory. The Pipfile and Pipfile.lock should already be present. In the root directory, create a virtual environment with the command *pipenv shell*. To install from the Pipfile.lock file, enter *pipenv install* in the same directory. This will install all the required packages. This may take upto 5 minutes, please be patient. After the installation, run the django migration command *python manage.py migrate_schemas --shared* which will initialize tables in the database. The usual command of *python manage.py migrate* has been replaced with the aforementioned command. The reason will be briefly described below. After the database has been initialized, start the python shell with the command *python manage.py shell* and create a primary entity. Please note, the primary entity will reside in the public schema (the default schema of postgresql), and will act as the fallback schema during runtime. The tables present in the public schema will be different from other schemas. All the schemas will be created and maintained by the database user *meeting.* The above steps will be provided in the form of a script along with a readme.txt when submitting this documentation. Please follow the steps in the readme.txt to run the script.

Python decouple is used to ensure the security in the django settings file. A .env/.ini and an .env.example/.ini.example file will be provided in the project which should be filled by the reader of this document during production.

A superuser will be created after the aforementioned script has been run. The credentials have not been mentioned in this document for security reasons. Hereby, any use of the word *superuser* should be understood as the user created by the script.

3. About django-tenant-schemas

The backend is developed by making an extensive use of django-tenant-schemas. Django tenant schemas ships with preconfigured database routing. In simple terms, database routing means, based on the request, django-tenant-schemas will determine which schema should be referenced.

On default configuration django-tenant-schemas uses subdomains (abcd.example.com, xyz.example.com), to determine the schema. We have changed the default configuration to use the request header instead. A request if it contains the header X-DTS-SCHEMA, the value in the header will be used to determine the respective schema. If the schema does not exist, *public* schema will be used as the fallback. This configuration resides in middleware.py inside the /meetingSpace folder.

Please note, any changes to middleware.py must be reflected on the frontend.

To read more on django-tenant-schemas, please visit:
https://django-tenant-schemas.readthedocs.io/en/latest/install.html

4. Authentication

We have used token-based authentication signed by the secret key in settings.py. Specifically, we have used JWT authentication. When a user credentials match, an access token and a refresh token is generated and returned to the frontend along with the expiry time of the access token.

Following auth0 practices, any subsequent request that requires authorization must have the access token present in the header of the request. An access token will be valid for 15 minutes and a refresh token will be valid for 15 days. These settings can be changed as required in settings.py (/meetingSpace).

A new access token can be generated using the refresh token. More on this in the **API Endpoints** section.

5. Groups

There are three access levels in the system. Superadmin, Company Admin and Employee. As such the access levels have been maintained through the use of groups and the data is stored in *users_user_groups* and *users_user_belongs_to*. Whenever a user is created, they are categorised in one of the access levels. The group/s a user belongs to is determined from the payload in the access token. To create a company admin, the user must be a Superadmin and to create an employee, the user must be a company admin. Based on the groups, we have derived three django permissions. *IsStaffUser, IsCompanyAdmin, IsEmployee* for Superadmin, Company admin and employee respectively. Any reference to these three terms, hereafter in the document must be understood as such.

Please note: A company admin is also an employee of the company and will be added to both groups. However, an employee will be present only in the employee group.

6. API Endpoints

The endpoints may already have been covered in the android/iOS/web documents presented with this document. However, this section is primarily intended for future developers as such this section should be referred to, whenever the inner workings of the system is needed.

Note: All the folders are inside the root folder, which is *newMeetingSpace*. All the APIs have also been documented through the use of swagger which can be viewed by visiting */swagger.*

Common

These are endpoints that are not restricted to any particular permission/access levels.

/auth/v1/signin
Methods: POST
Authentication: Not required

The endpoint for signing in to the application. If the credentials are correct, this endpoint returns the *access_token, refresh_token, expiry_time*, *schema_name.*

/auth/v1/change_password
Methods: POST
Authentication: Required
Permission: Any

This endpoint takes the old password and a new password. If the old password matches the one in the database, password is changed and status code 204 is returned.

/auth/v1/reset_password
Methods: POST
Authentication: Not required

This endpoint requires an email. If the user exists on the database and if the user is active, a password reset link will be sent to the email, which can be used to reset the password.

The password reset link cannot be used more than once.

/auth/v1/refresh_token
Methods: POST
Authentication: Not required

This endpoint requires the refresh token obtained along with the access token during login. If the refresh token is valid, a new pair of access token and refresh token is returned.

The refresh token, according to current settings, is valid for 15 days.

Superadmin
All the URLs for the superadmin are located inside */api/v1/superApi/urls.py*.

/super/createCompany/
Methods: POST
Authentication: Required
Permission: IsStaffUser

As the URL suggests, this endpoint creates a company. One of the required data fields is *short_name*. In the database, if all the constraints are met, a new organization is added to *organization_organization* in the *public* schema. A new schema with the name *short_name* + schema is created. The schema name must be unique as such the *short_name* received in the request must also be unique. Furthermore, an email for the admin of the company is also required in the request body. After the schema has been created, the email is added to *users_user* table and the schema name that the email belongs to is stored in *temp_name*. An email with the verification link is sent to the email, which can be used to reset the password. The verification link will expire after one use.
In short, all the organizations are stored in the *organization_organization* table and all the users are stored in *users_user* table.

/super/viewCompany/
Methods: GET
Authentication: Required
Permission: IsStaffUser

This endpoint lists all the organizations in the system. Along with the organizations, their details (*subscription_expiry, subscription_start_date, is_active*) is also exposed from this endpoint.

/super/statusCompany/<string:short_name>/
Methods: GET, PUT
Authentication: Required
Permission: IsStaffUser

A PUT request to this endpoint can be used to change if the company is active, is on trial or to extend the subscription_expiry date. Similarly, a GET request to this endpoint returns the active status, on trial status and the subscription expiry date.

/super/dashboard/
Methods: GET
Authentication: Required
Permission: IsStaffUser
Query Parameters: Limit

This endpoint is specific to the needs of the frontend. It returns data required by the superadmin for business purposes. For example, currently this endpoint returns:

- Total number of companies
- Total number of active companies
- Total number of companies that are on trial
- Average time taken to host a meeting
- Average users per meeting

This endpoint is specific to the needs of the business and should be modified as per the need.

The query parameter limit is optional. It can be used to limit the number of results returned from the endpoint.

Company Admin
These endpoints are specific to a company admin.

/admin/dashboard/
Methods: GET
Authentication: Required
Permission: IsCompanyAdmin
Query Parameter: Limit, Year

This endpoint returns data relating to the company. The response should be modified as needed. Currently, a part of the response is:

- Rooms with most meetings held
- Least used rooms
- Number of ongoing meetings
- Meetings per month

There are two query parameters, limit and year, both are optional. Limit can be used to limit the number of results in the response. Year can be used to retrieve data of a particular year. If a year is not passed as a parameter, data relating to the current year is returned.

/auth/v1/add_employee/
Methods: POST
Authentication: Required
Permission: IsCompanyAdmin

This endpoint takes an email as input. If the email is valid, a verification link is sent to the respective email. Furthermore, the email is added to the respective company schema.

/auth/v1/allusers/
Methods: GET
Authentication: Required
Permission: IsCompanyAdmin

This endpoint returns all the users in the company.

/auth/v1/user/<int:pk>/
Methods: GET, PUT
Authentication: Required
Permission: IsCompanyAdmin

This endpoint returns all the users on GET request and allows the admin to change the active/inactive status through PUT request.

/organization/profile/
Methods: GET, PUT
Authentication: Required
Permission: IsCompanyAdmin

On GET request the details of the respective organization is returned. The said detail can be updated through PUT request.

/property/
Methods: GET, POST
Authentication: Required
Permission: IsCompanyAdmin

This endpoint allows the user to add property to a schema through a POST request. During GET request, an **optional** query parameter *?status*= can be passed, whose value must be either *available* or *unavailable.* If *available* is passed as the query parameter all buildings that have not been SHUT DOWN are returned in the response and vice versa when *unavailable* is passed as the query parameter. A list of all buildings are returned if the optional query parameter is not used.

/property/<int:pk>/
Methods: GET, PUT, PATCH
Authentication: Required
Permission: IsCompanyAdmin

While the above endpoint returned the list of all buildings in a given schema, this endpoint returns the details of a particular building based on the ID passed in the URL. This URL can also be used to update/edit the information of any building by using the PUT or PATCH request.

/property/department/
Methods: GET, POST
Authentication: Required
Permission: IsCompanyAdmin

This endpoint returns the list of all departments on GET request. New departments can be added via the POST request.

/property/department/<int:pk>/

Methods: GET, PUT, DELETE
Authentication: Required
Permission: IsCompanyAdmin

This endpoint allows the company admin to retrieve and update the details of any department. Furthermore, a department can also be deleted by sending a DELETE request and the primary key of the respective department.

/property/department/members/<int:pk>/
Methods: GET
Authentication: Required
Permission: IsCompanyAdmin

This endpoint returns the list of all the members that are part of a given department. The department is determined by the primary key passed in the URL.

/property/room/
Methods: POST
Authentication: Required
Permission: IsCompanyAdmin

This endpoint can be used to add rooms to a specific property. The property id must be passed in the body along with other required attributes.

/property/room/update/<int:pk>/
Methods: PUT
Authentication: Required
Permission: IsCompanyAdmin

This endpoint can be used to update the details of any room. The room is determined by the primary key passed in the URL.

All the endpoints have been started with */v1/* for version control.