Neeley Business

Open Planner
Developer's Manual

Version 1.0

Revision History

Date	Version	Description	Author
22/02/2023	0.1	Initial Draft	Alex Roa
03/04/2023	1.0	Final Draft	Thuong Hoang

Table of Contents

Developer’s Manual

1. Introduction

The purpose of this document is to explain how the Open Planner application is developed. It will explain the deployment of the Front-End and Back-End on AWS, API documentation, and the database schema. This document is intended to be used by future developers of Open Planner who will maintain and correct the functions of the system.

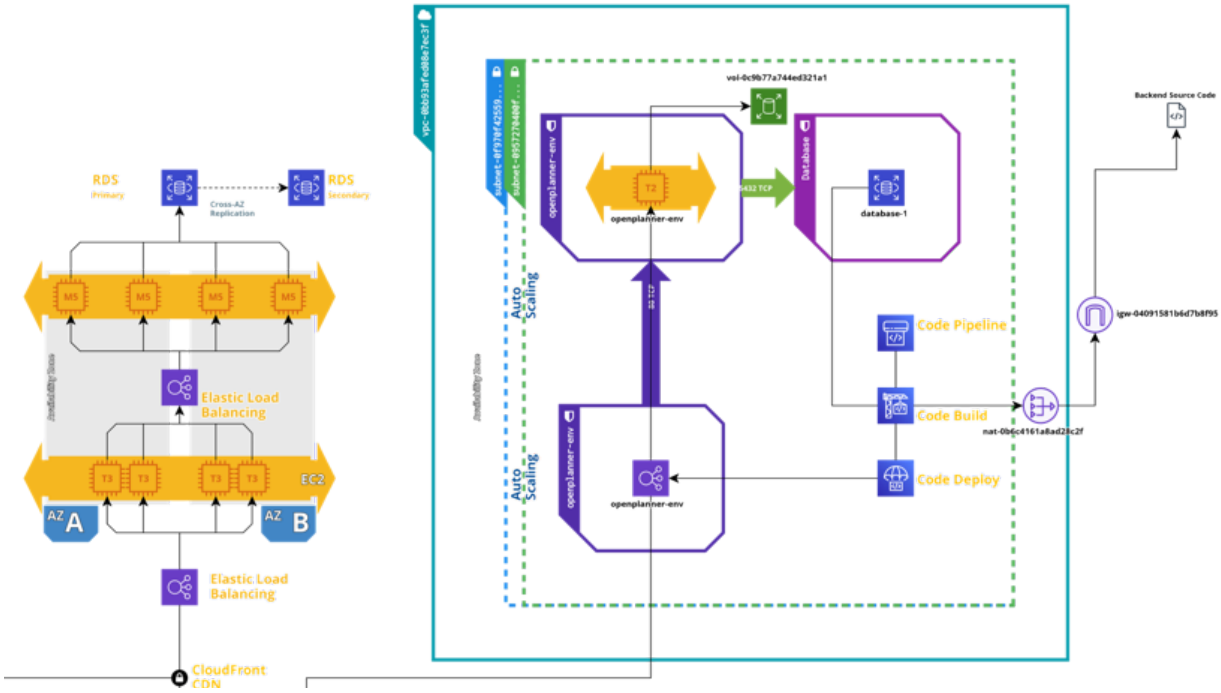
2. Deployment

2.1 Front-end Architecture

Open Planner uses AWS amplify to optimize the delivery of frontend assets that is sourced from the frontend Github repository. AWS amplify sources the code from the Open Planner front-end Github repository to display the frontend assets on the Open Planner domain, open-planner.com. When users enter the open-planner.com domain, AWS cloud front is used to redirect users to the AWS amplify app to display the contents of the Open Planner front-end assets.

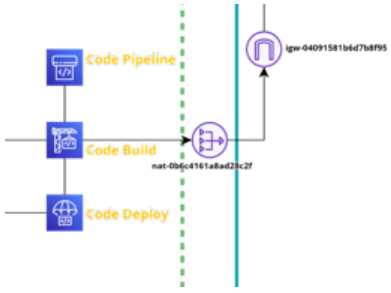
2.2 Back-end Architecture

Open Planner uses a variety of AWS resources to deploy the back-end contents. AWS Elastic Beanstalk is one of the many resources that is used to deploy the back-end contents. Elastic Beanstalk creates an EC2 instance, which runs on Linux with Python version 3.8. This EC2 instance is connected to a relational database, RDS, which has the engine of PostgreSQL. All the resources that Elastic Beanstalk creates is auto scaling, which will increase the EC2 instances or RDS instances as needed as more users are using the application.



2.3 CI/CD Pipeline

Open Planners uses AWS built-in CodePipeline as a CI/CD pipeline for automatic deployments. CodePipeline sources the Open Planner back-end source code from Github. The source code runs the build phase to make changes to the database as necessary and checks for errors within the source code. The build phase also installs all the Python dependencies needed to run the back-end. To allow this, CodeBuild is connected to a NAT Gateway which is connected to an internet gateway. This allows CodeBuild to access the source code along with the internet to download all the dependencies. Once CodeBuild succeeds, CodeDeploy delivers all the source code to the Elastic Beanstalk environment to allow access to the database.



2.4 RDS

Open Planner uses an RDS as the resource to store all of the application's data using PostgreSQL as the engine. The database may be accessible through PgAdmin which can be downloaded at https://www.pgadmin.org/download/

Once this is downloaded and PgAdmin is set up, register a server with the application by right-clicking the servers and going to register and clicking on the server. A popup will show up and go to the connection tab. Once on the connection tab, paste the database name:

database-1.cmarjxwknbvu.us-east-1.rds.amazonaws.com into the hostname/address textfield. Set the Port as 5432, and the Username as postgres. Input openplanner28 as the password. You can click save to create a server connection to the deployed database. Once the connection is stable, the database can be managed.

2.5 EC2

To connect to the EC2 instance, go to the AWS management console at https://**aws**.amazon.com/account/sign-up. Login with the username openplanner, password OpenPlanner28, and account ID 991655202121. Search EC2 in the search bar and click on Instances on the left navigation bar. Click on the instance ID of the EC2 instance that needs to be connected to. Once there, you can click on connect to get to the EC2 instance. You will see the terminal of the Linux computer. The backend virtual environment can be activated by doing source /var/app/${folder_here}/bin/activate. After the environment is activated, cd to /var/app/current and then you can do python manage.py to make changes as needed.

3. API Documentation

This section describes the API endpoints that are available in the backend. These endpoints are located on the domain: www.api.open-planner.com. The headers for most of these endpoints consists of the following:

{

"Content-Type": "application/json",

Accept: "application/json"

}

3.1 User Management

API endpoints in this section entails user registration, login, logout, and getting the current user.

ENDPOINTS

/auth/users/

Creates a user (POST)

Request fields:

email	String
username	String
password	String
re_password	String
is_superuser	boolean

Example:

EX-1: Student Account - {email: example@OP.com, username: student1, password: TestPassword, re_password: TestPassword, is_superuser: False}

EX-2: Admin Account - {email: admin@OP.com, username: admin, password: AdminPass, re_password: AdminPass, is_superuser: True}

Response with 201 status:

{

"username": "${username}",

"first_name": "",

"last_name": "",

"is_superuser": "${is_superuser}",

"email": "${email}",

"id": "${uid}"

}

username and email is a string, is_superuser is a boolean, uid is an integer

Response with status 400:

{

"username": ["${error_message}"],

"email": ["${error_message}"],

"password": ["${error_message}"],

"re_password": ["${error_message}"]

}

The fields will show up depending on which fields have errors.

/auth/token/login/

Logs user in (POST)

Request fields:

email	String
password	String

Response with Status 200:

{

"auth_token": "${token}"

}

token is the token retrieved from the Django backend

Response with Status 400:

{

"non_field_errors": ["Unable to log in with provided credentials."]

}

/auth/token/logout/

Logout user (POST)

Headers:

{

Authorization: "token ${authToken}"

}

Here authToken is a variable that is located in localStorage which is obtained from the user logging in.

Response Status: 200 (Success)

Response with 401 status (Error):

{

"detail": "Invalid token."

}

/auth/users/me/

Retrieve current user's information (GET)

Headers:

{

"Content-Type": "application/json",

Accept: "application/json",

Authorization: "token ${authToken}"

}

Response with 200 status:

{

"username": "${username}",

"first_name": "",

"last_name": "",

"is_superuser": "${is_superuser}",

"id": "${uid}",

"email": "${email}"

}

username and email is a string, is_superuser is a boolean, uid is an integer

If there is an error, it will return a 401 status.

3.2 Event Management

API calls in this section pertain to uploading syllabi to extract events, retrieving all events for a given user, getting a specific event, adding a new event, deleting an event, and updating an event.

ENDPOINTS

/auth/upload

Retrieve events from syllabus (POST)

Headers:

{

"Content-Disposition": "attachment; filename=${file_name}",

filename: ${filename},

Authorization: "token ${authToken}"

}

Request fields:

file	Binary representation of PDF file

Response with status 200:

"PENDING SYLLABUS CREATED" - syllabus couldn't be parsed and is sent to Admin

"FILE EXISTS" - syllabus has previously been parsed

"If it doesnt exist and can be parsed" - syllabus is a new file and has been parsed successfully

If there is an error, it will return a 400 status.

/auth/events/

Retrieve all user events (GET)

Headers:

```
{
   Authorization: “token ${authToken}”
}
```

Response with status 200:

```
[
   {
      “id”: ${eid},
      “start”: “${event_start_datetime}”,
      “end”: “${event_end_datetime}”,
      “title”: “${event_title}”,
      “body”: “${event_body}”,
      “category”: “time”,
      “user_fk”: ${uid},
      “calendar_fk”: ${cid}
   },
   …
]
```

Add an event (POST)

Headers:

```
{
   Authorization: “token ${authToken}”
}
```

Request fields:

start	String in format of YYYY-MM-DDThh:mm:ss
end	String in format of YYYY-MM-DDThh:mm:ss
title	String
body	String
Category	“time”
user_fk	Integer
calendar_fk	Integer

Response with 201 status:

```
{
   “id”: ${eid},
   “start”: “${event_start_datetime}”,
   “end”: “${event_end_datetime}”,
```

“title”: “${event_title}”,
“body”: “${event_body}”,
“category”: “time”,
“user_fk”: ${uid},
“calendar_fk”: ${cid}
}

Response with 400 status:

{
“start”: [“This field is required.”],
“end”:[“This field is required.”],
“title”: [“This field is required.”],
“body”: [“This field is required.”],
“category”: [“This field is required.”],
“user_fk”: [“This field is required.”],
“calendar_fk”: [“This field is required.”]
}

This will show the fields that have an error while making the request.

/auth/events/${eid}/

Retrieve event data (GET)

Headers:

{
Authorization: “token ${authToken}”
}

Response with status 200:

{
“id”: ${eid},
“start”: “${event_start_datetime}”,
“end”: “${event_end_datetime}”,
“title”: “${event_title}”,
“body”: “${event_body}”,
“category”: “time”,
“user_fk”: ${uid},
“calendar_fk”: ${cid}
}

Modifies an event information (PUT)

Request fields:

start	String in format of YYYY-MM-DDThh:mm:ss
end	String in format of YYYY-MM-DDThh:mm:ss
title	String
body	String

Category	"time"
user_fk	Integer
calendar_fk	Integer

Response with 200 status:

```
{
    "id": ${eid},
    "start": "${event_start_datetime}",
    "end": "${event_end_datetime}",
    "title": "${event_title}",
    "body": "${event_body}",
    "category": "time",
    "user_fk": ${uid},
    "calendar_fk": ${cid}
}
```

Response with 400 status:

```
{
    "start": ["This field is required."],
    "end":["This field is required."],
    "title": ["This field is required."],
    "body": ["This field is required."],
    "category": ["This field is required."],
    "user_fk": ["This field is required."],
    "calendar_fk": ["This field is required."]
}
```

This will show the fields that have an error while making the request.

Remove a user's event (DELETE)
Headers:

```
{
    Authorization: "token ${authToken}"
}
```

3.3 Calendar Management

API endpoints in this section include getting a specific calendar associated with a user, editing a calendar, getting all syllabi associated with a user

ENDPOINTS

/auth/user-calendar/${cid}
Retrieves a user's calendars (GET)
Headers:

```
{
    Authorization: "token ${authToken}"
}
```

Response with 200 status:

```
{
    "id": ${cid},
    "name": "${calendar_name}",
    "color": "${calendar_hex_color}"
}
```

Modify a user's calendar (PUT)
Headers:

```
{
    Authorization: "token ${authToken}",
    "Content-Type": "application/json"
}
```

Request fields:

id	integer
name	String
color	Hex code in string format

Response with 200 status:

```
{
    "id": ${cid},
    "name": "${calendar_name}",
    "color": "${calendar_hex_color}"
}
```

Response with 400 status:

```
{
    "id": ["This field is required."],
    "name":["This field is required."],
    "color": ["This field is required."]
}
```

This will show the fields that have an error while making the request.

/auth/syllabus-delete/${sid}/
Delete a user's calendar (DELETE)
Headers:

```
{
    Authorization: "token ${authToken}",
    "Content-Type": "application/json"
}
```

/auth/syllabus-delete-all/
Delete all instances of user's calendars (DELETE)
Headers:

{
Authorization: "token ${authToken}",
"Content-Type": "application/json"
}

/auth/syllabus/
Retrieves all user's calendars (GET)
Headers:

{
Authorization: "token ${authToken}"
}
Response with 200 status:

```
[
    {
        "id": ${cid},
        "user_fk": ${uid},
        "syllabus_fk": {
            "id": ${sid},
            "file": "${binary_file_of_syllabus}",
            "hashed": "${hash_of_file}",
            "status": "${status_of_syllabus}"
        },
        "calendar_fk": {
            "id": ${cid},
            "name": "${calendar_name}",
            "color": "${calendar_hex_color}"
        }
    }
    …
]
```

3.3 Admin Management

API endpoints include retrieving pending syllabi for Admin users to process, getting specific PDF files uploaded to the database, and updating the events for a pending syllabi.

ENDPOINT
/auth/pending/
Retrieve all pending syllabi (GET)
Headers:

{
Authorization: "token ${authToken}"
}
Response with status 200:

```
[
```

```
    {
        "id": ${sid},
        "file": "${binary_file_of_syllabus}",
        "hashed": "${hash_of_file}",
        "status": "${status_of_file}"
    },
    …
]
```

/auth/file
Retrieve the syllabus binary file (POST)
Headers:

```
{
    "Content-Type": "application/json"
}
```

Request fields:

file_path	String

Response with 400 status:

```
{
    "error": "File not found"
}
```

Response with 200 status should include the entire PDF file that is viewable with a PDF Viewer

/auth/completed/
Retrieve all completed syllabi (GET)
Headers:

```
{
    Authorization: "token ${authToken}"
}
```

Response with status 200:

```
[
    {
        "id": ${sid},
        "file": "${binary_file_of_syllabus}",
        "hashed": "${hash_of_file}",
        "status": "${status_of_file}"
    },
    …
]
```

/auth/parsedEvent/${sid}
Add events to a pending syllabus (POST)
Headers:

{
Authorization: “token ${authToken}”,
“Content-Type”: “application/json”
}
Request fields:
The following fields should be in an array to add a collective amount of events

start	String in format of YYYY-MM-DDThh:mm:ss
end	String in format of YYYY-MM-DDThh:mm:ss
title	String
body	String
syllabus_fk	Integer
category	String

Response with 200 status:

```
[
    {
        “id”: ${eid},
        “start”: “${start_datetime}”,
        “end”: “${end_datetime}”,
        “title”: “${title}”,
        “category”: “time”,
        “syllabus_fk”: ${sid}
    }
    …
]
```

Response with 400 status:

```
{
    “start”: [“This field is required.”],
    “end”:[“This field is required.”],
    “title”: [“This field is required.”],
    “category”: [“This field is required.”],
    “syllabus_fk”: [“This field is required.”]
}
```

This will show the fields that have an error while making the request.

/auth/superusers/
Retrieve all Admin users (GET)
Headers:

{
Authorization: “token ${authToken}”
}
Response with 200 status:

```
[
    {
        “username”: “${username}”,
        “first_name”: “${first_name}”,
        “last_name”: “${last_name}”,
        “is_superuser”: “${is_superuser}”,
        “id”: ${uid},
        “email”: “${u_email}”
    },
    …
]
```

/auth/superusers/${uid}/
Delete an Admin User (DELETE)
Headers:

```
{
    Authorization: “token ${authToken}”
}
```

4. Database Schema

This section goes over the schema of the database.