

## Table Of Content

<b>Table Of Content</b>	<b>1</b>
<b>Tool and technology used</b>	<b>2</b>
<b>Raw Table</b>	<b>4</b>
<b>Processed Table ER diagram</b>	<b>6</b>
<b>API data upload</b>	<b>9</b>
<b>Bulk Data Upload</b>	<b>15</b>
<b>Visualization</b>	<b>15</b>
<b>Application Screenshot</b>	<b>19</b>
<b>Gantt Chart</b>	<b>22</b>

## Tool and technology used

1. Python Programming Language
2. Django Framework
3. JWT authentication (Simple JWT library in python)
4. Draw.io
5. Google Sheet
6. Google Docs
7. JavaScript for visualization (Chart.js and D3.js)
8. PostgreSQL(Database)

1. **Python Programming Language:** Python is a high-level, interpreted programming language known for its readability and simplicity. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is widely used for web development, data analysis, artificial intelligence, scientific computing, and more.
2. **Django Framework:** Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It follows the Model-View-Controller (MVC) architectural pattern and provides an ORM for database interactions, a templating engine for rendering views, and a robust set of tools for managing web applications.

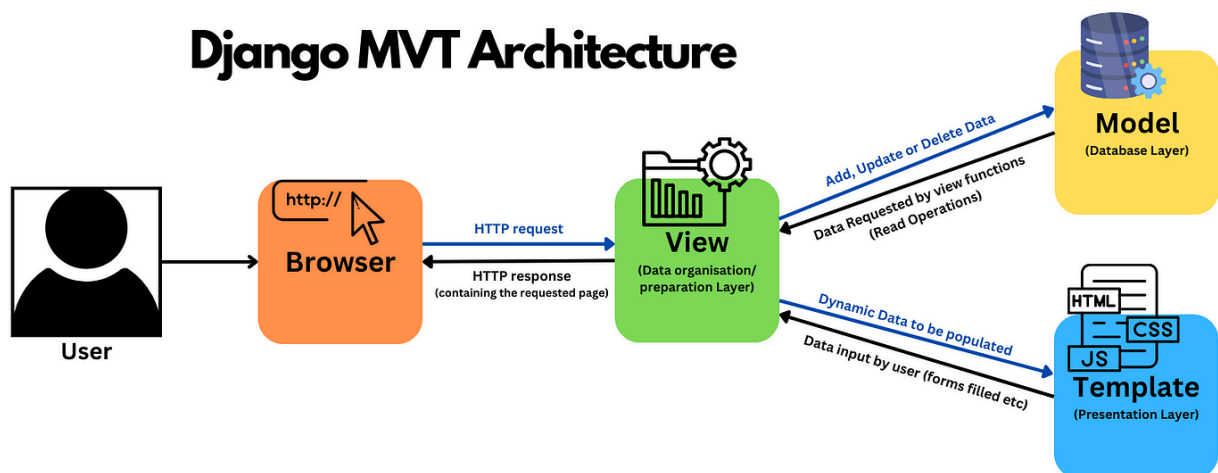


Fig: Django Framework Flow

3. **JWT Authentication (Simple JWT Library in Python):** JSON Web Token (JWT) is an open standard for securely transmitting information between parties as a JSON object. Simple JWT is a Python library that provides a JSON Web Token authentication backend for Django. It allows for stateless, token-based authentication, making it suitable for RESTful APIs.

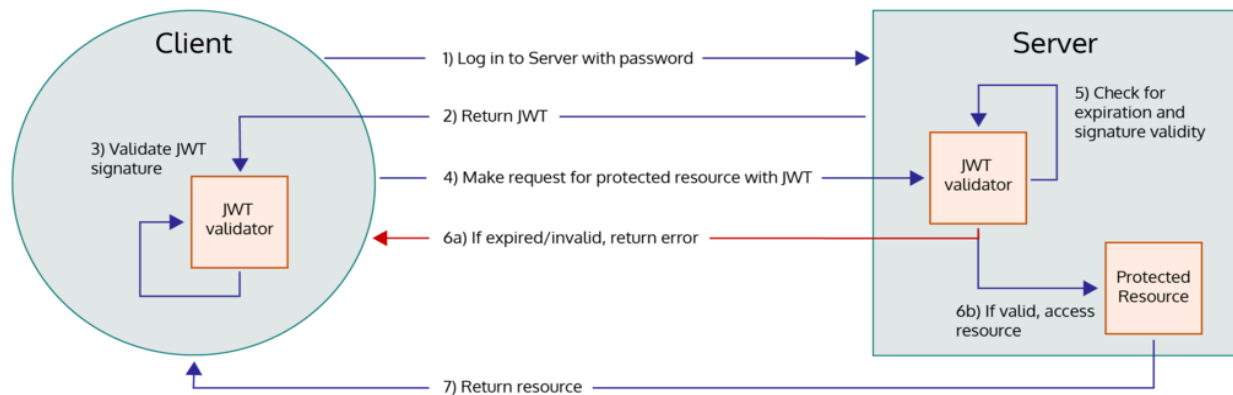


Fig: JWT Authentication

4. **Draw.io:** Draw.io is a web-based diagramming tool that allows users to create flowcharts, network diagrams, UML diagrams, and other types of visual representations. It offers a wide range of shapes, icons, and templates, and supports real-time collaboration.
5. **Google Sheet:** Google Sheets is a web-based spreadsheet application provided by Google. It allows users to create, edit, and collaborate on spreadsheets online. It supports functions, formulas, charts, and various data visualization tools, and can be integrated with other Google services and APIs.
6. **Google Docs:** Google Docs is a web-based word processing application provided by Google. It allows users to create, edit, and collaborate on documents online. Google Docs supports real-time collaboration, rich text formatting, and integration with other Google services and third-party applications.
7. **JavaScript for Visualization (Chart.js and D3.js):**
  - **Chart.js:** Chart.js is a simple, yet flexible JavaScript charting library. It provides easy-to-use options for creating various types of charts, such as bar, line, pie, and doughnut charts. Chart.js is known for its simplicity and ease of integration with web applications.
  - **D3.js:** D3.js (Data-Driven Documents) is a powerful JavaScript library for creating dynamic and interactive data visualizations in web browsers. It uses web standards such as SVG, HTML, and CSS to bring data to life. D3.js is highly customizable and allows developers to create complex visualizations with precision.
8. **PostgreSQL(Database)**

PostgreSQL is a powerful, open-source object-relational database system known for its reliability, feature robustness, and performance. It is used in this project to handle data storage, complex queries. The project leverages PostgreSQL for its relational database needs, ensuring efficient and scalable data management.

## Raw Table

In the below raw table the data from API or the bulk upload is fetched and stored as it is.

raw_employeeleave
id_s
userId
empld
teamManagerId
designationId
designationName
firstName
middleName
lastName
email
isHr
isSupervisor
leaveIssuerId
currentLeaveIssuerId
issuerFirstName
issuerMiddleName
issuerLastName
currentLeaveIssuerEmail
departmentDescription
startDate
endDate
leaveDays
reason
leaveStatus
status
responseRemarks
leaveTypeId
leaveType
defaultDays
transferableDays
isConsecutive
fiscalId
fiscalStartDate
fiscalEndDate
fiscalIsCurrent
createdAt
updatedAt
isAutomated
isConverted
allocations
createdat_db

Fig:Raw table

The above raw table is created from the Django Model.

```
class EmployeeLeave(models.Model):
    id_s = models.IntegerField(null=True)
    userId = models.IntegerField(null=True)
    empId = models.CharField(max_length=10, null=True)
    teamManagerId = models.IntegerField(null=True)
    designationId = models.IntegerField(null=True)
    designationName = models.CharField(max_length=100, null=True)
    firstName = models.CharField(max_length=100, null=True)
    middleName = models.CharField(max_length=100, null=True)
    lastName = models.CharField(max_length=100, null=True)
    email = models.EmailField(null=True)
    isHr = models.BooleanField(null=True)
    isSupervisor = models.BooleanField(null=True)
    leaveIssuerId = models.IntegerField(null=True)
    currentLeaveIssuerId = models.IntegerField(null=True)
    issuerFirstName = models.CharField(max_length=100, null=True)
    issuerMiddleName = models.CharField(max_length=100, null=True)
    issuerLastName = models.CharField(max_length=100, null=True)
    currentLeaveIssuerEmail = models.EmailField(null=True)
    departmentDescription = models.TextField(null=True)
    startDate = models.DateField(null=True)
    endDate = models.DateField(null=True)
    leaveDays = models.IntegerField(null=True)
    reason = models.TextField(null=True)
    leaveStatus = models.CharField(max_length=20, null=True)
    status = models.CharField(max_length=20, null=True)
    responseRemarks = models.TextField(null=True)
    leaveTypeId = models.IntegerField(null=True)
    leaveType = models.CharField(max_length=50, null=True)
    defaultDays = models.IntegerField(null=True)
    transferableDays = models.IntegerField(null=True)
    isConsecutive = models.BooleanField(null=True)
    fiscalId = models.IntegerField(null=True)
    fiscalStartDate = models.DateTimeField(null=True)
    fiscalEndDate = models.DateTimeField(null=True)
    fiscalIsCurrent = models.BooleanField(null=True)
    createdAt = models.DateTimeField(null=True)
    updatedAt = models.DateTimeField(null=True)
    isAutomated = models.BooleanField(null=True)
    isConverted = models.BooleanField(null=True)
    allocations = models.TextField(null=True)
    createdat_db = models.DateTimeField(null=True)
    def __str__(self):
        return f"{self.firstName} {self.lastName} - {self.leaveType}"
```

## Processed Table ER diagram

In the below processed ER diagram , the data from above raw table is normalized and merged inserted into the below tables

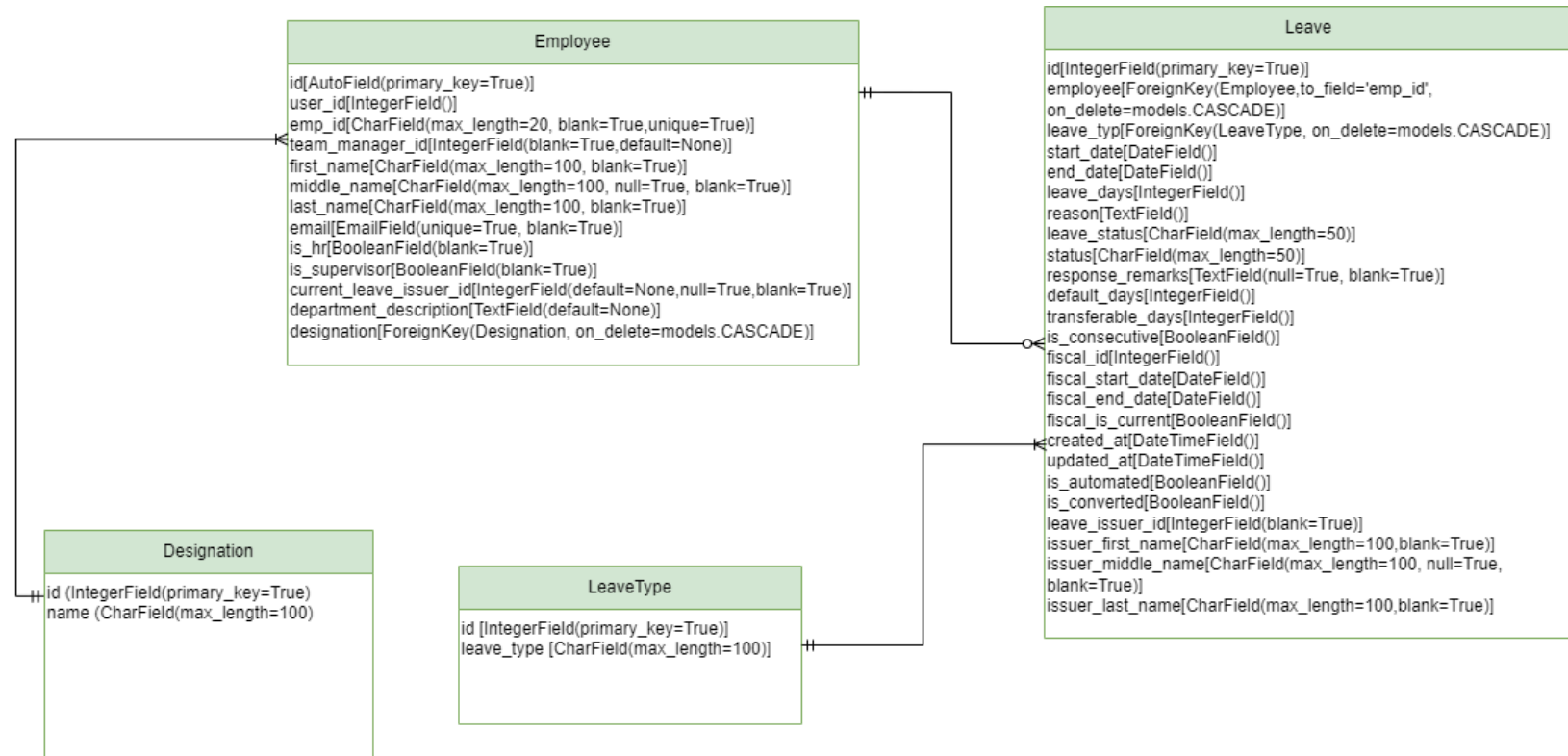


Fig: Entity Relationship Diagram

The above tables in the postgresQL is created from the Django model. The list of django model for the above tables are:

```
class Designation(models.Model):
    id = models.IntegerField(primary_key=True)
    name = models.CharField(max_length=100)

    def __str__(self):
        return self.name

class LeaveType(models.Model):
```

```

    id = models.IntegerField(primary_key=True)
    leave_type = models.CharField(max_length=100)

    def __str__(self):
        return self.name

class Employee(models.Model):
    id = models.AutoField(primary_key=True)
    user_id = models.IntegerField()
    emp_id = models.CharField(max_length=20, blank=True, unique=True)
    team_manager_id = models.IntegerField(blank=True, default=None)
    first_name = models.CharField(max_length=100, blank=True)
    middle_name = models.CharField(max_length=100, null=True, blank=True)
    last_name = models.CharField(max_length=100, blank=True)
    email = models.EmailField(unique=True, blank=True)
    is_hr = models.BooleanField(blank=True)
    is_supervisor = models.BooleanField(blank=True)
    #leave_issuer_id = models.IntegerField(blank=True)
    current_leave_issuer_id = models.IntegerField(default=None, null=True, blank=True)
    #issuer_first_name = models.CharField(max_length=100, blank=True)
    #issuer_middle_name = models.CharField(max_length=100, null=True, blank=True)
    #issuer_last_name = models.CharField(max_length=100, blank=True)
    current_leave_issuer_email = models.EmailField(default='unknown@unknown.com') # Provide a
        default value here
    department_description = models.TextField(default=None)
    designation = models.ForeignKey(Designation, on_delete=models.CASCADE)

    def __str__(self):
        return f'{self.first_name} {self.last_name}'

class Leave(models.Model):
    id = models.IntegerField(primary_key=True)
    employee = models.ForeignKey(Employee, to_field='emp_id', on_delete=models.CASCADE)
    leave_type = models.ForeignKey(LeaveType, on_delete=models.CASCADE)
    start_date = models.DateField()
    end_date = models.DateField()
    leave_days = models.IntegerField()
    reason = models.TextField()
    leave_status = models.CharField(max_length=50)
    status = models.CharField(max_length=50)
    response_remarks = models.TextField(null=True, blank=True)
    default_days = models.IntegerField()
    transferable_days = models.IntegerField()
    is_consecutive = models.BooleanField()
    fiscal_id = models.IntegerField()
    fiscal_start_date = models.DateField()
    fiscal_end_date = models.DateField()
    fiscal_is_current = models.BooleanField()

```

```
created_at = models.DateTimeField()
updated_at = models.DateTimeField()
is_automated = models.BooleanField()
is_converted = models.BooleanField()
leave_issuer_id = models.IntegerField(blank=True)
issuer_first_name = models.CharField(max_length=100, blank=True)
issuer_middle_name = models.CharField(max_length=100, null=True, blank=True)
issuer_last_name = models.CharField(max_length=100, blank=True)

def __str__(self):
    return f'{self.leave_type} ({self.start_date} to {self.end_date})'
```



## API data upload

The data is fetched from the given API passed through the .env variable . The data is fetched asynchronously making use of the python library ``asyncio`` and defining the function as async and using await to wait for all the data from the API.

The given API

```
`API_URL=https://dev.vyaguta.lftechnology.com.np/api/leave/leaves?fetchType=all&startDate=2024-04-23&endDate=2024-07-30&size=10000000&roleType=issu  
er`  
,
```

is broken down in several API by dividing the startDate and endDate in a three month gap and making the API request asynchronously .

After the data is fetched from the API , it is bulk inserted into the ``raw_employeeleave`` table along with the date of insertion.

After that data is successfully inserted in the raw table , then it runs the script from the application to normalize the data. Basically , it first filters the data from the date of insertion (i.e don't make any operation to older data) and then merges insert into the other normalized table.

Scripts 1:

```
-- Insert filtered results from source_table to destination_table  
INSERT INTO apidataupload_designation(id, name)  
SELECT DISTINCT "designationId", "designationName"  
FROM apidataupload_employeeleave a  
WHERE a."designationId" NOT IN  
(  
    SELECT DISTINCT "designationId" FROM apidataupload_designation  
) ;
```

## Script 2:

```
-- Insert filtered results from source_table to destination_table
MERGE INTO apidataupload_employee AS employee
USING (SELECT DISTINCT "userid",
                        "empid",
                        "teammanagerid",
                        "designationid",
                        "firstname",
                        "middlename",
                        "lastname",
                        "email",
                        "ishr",
                        "issupervisor",
                        "currentleaveissuerid",
                        "currentleaveissueremail",
                        "departmentdescription"
FROM      apidataupload_employeeleave el
WHERE     el."createdat_db" >= %s) a
ON employee."user_id" = a."userId"
WHEN NOT matched THEN
    INSERT (user_id,
            emp_id,
            team_manager_id,
            designation_id,
            first_name,
            middle_name,
            last_name,
            email,
            is_hr,
            is_supervisor,
            current_leave_issuer_id,
            current_leave_issuer_email,
            department_description)
VALUES ("userid",
        "empid",
        "teammanagerid",
        "designationid",
        "firstname",
        "middlename",
```

```

        "lastname",
        "email",
        "ishr",
        "issupervisor",
        "currentleaveissuerid",
        "currentleaveissueremail",
        "departmentdescription")
WHEN matched THEN
    UPDATE SET team_manager_id = a."teamManagerId",
        designation_id = a."designationId",
        first_name = a."firstName",
        middle_name = a."middleName",
        last_name = a."lastName",
        email = a."email",
        is_hr = a."isHr",
        is_supervisor = a."isSupervisor",
        current_leave_issuer_id = a."currentLeaveIssuerId",
        current_leave_issuer_email = a."currentLeaveIssuerEmail",
        department_description = a."departmentDescription";

```

### Script 3:

```

-- Insert filtered results from source_table to destination_table
MERGE INTO apidataupload_leave AS l
USING (SELECT DISTINCT "id_s",
        "empid",
        "leaveissuerid",
        "issuerfirstname",
        "issuermiddlename",
        "issuerlastname",
        "startdate",
        "enddate",
        "leavedays",
        "reason",
        "leavestatus",
        "status",
        "responseremarks",
        "leavetypeid",

```

```

        "defaultdays",
        "transferabledays",
        "isconsecutive",
        "fiscalid",
        "fiscalstartdate",
        "fiscalenddate",
        "fiscaliscurrent",
        "createdat",
        "updatedat",
        "isautomated",
        "isconverted"

FROM    apidataupload_employeeleave el
WHERE   el."createdat_db" >= %s) a
ON l."id" = a."id_s"
WHEN NOT matched THEN
INSERT ( "id",
        "employee_id",
        "leave_issuer_id",
        "issuer_first_name",
        "issuer_middle_name",
        "issuer_last_name",
        "start_date",
        "end_date",
        "leave_days",
        "reason",
        "leave_status",
        "status",
        "response_remarks",
        "leave_type_id",
        "default_days",
        "transferable_days",
        "is_consecutive",
        "fiscal_id",
        "fiscal_start_date",
        "fiscal_end_date",
        "fiscal_is_current",
        "created_at",
        "updated_at",
        "is_automated",
        "is_converted" )

```

```

VALUES ( "id_s",
        "empid",
        "leaveissuerid",
        "issuerfirstname",
        "issuermiddlename",
        "issuerlastname",
        "startdate",
        "enddate",
        "leavedays",
        "reason",
        "leavestatus",
        "status",
        "responseremarks",
        "leavetypeid",
        "defaultdays",
        "transferabledays",
        "isconsecutive",
        "fiscalid",
        "fiscalstartdate",
        "fiscalenddate",
        "fiscaliscurrent",
        "createdat",
        "updatedat",
        "isautomated",
        "isconverted" )
WHEN matched THEN
    UPDATE SET "id" = a."id_s",
              "employee_id" = a."empId",
              "leave_issuer_id" = a."leaveIssuerId",
              "issuer_first_name" = a."issuerFirstName",
              "issuer_middle_name" = a."issuerMiddleName",
              "issuer_last_name" = a."issuerLastName",
              "start_date" = a."startDate",
              "end_date" = a."endDate",
              "leave_days" = a."leaveDays",
              "reason" = a."reason",
              "leave_status" = a."leaveStatus",
              "status" = a."status",
              "response_remarks" = a."responseRemarks",
              "leave_type_id" = a."leaveTypeId",

```

```
"default_days" = a."defaultDays",
"transferable_days" = a."transferableDays",
"is_consecutive" = a."isConsecutive",
"fiscal_id" = a."fiscalId",
"fiscal_start_date" = a."fiscalStartDate",
"fiscal_end_date" = a."fiscalEndDate",
"fiscal_is_current" = a."fiscalIsCurrent",
"created_at" = a."createdAt",
"updated_at" = a."updatedAt",
"is_automated" = a."isAutomated",
"is_converted" = a."isConverted";
```

#### Script 4:

```
-- Insert filtered results from source_table to destination_table
merge INTO apidataupload_leavetype AS lt
USING (SELECT DISTINCT "leavetypeid",
                      "leavetype"
        FROM   apidataupload_employeeleave el
        WHERE  el."createdat_db" >= %s) a
ON lt."id" = a."leaveTypeId"
WHEN NOT matched THEN
  INSERT ( "id",
          "leave_type" )
VALUES ( "leavetypeid",
        "leavetype" )
WHEN matched THEN
  UPDATE SET "id" = a."leaveTypeId",
            "leave_type" = a."leaveType";
```

## **Bulk Data Upload**

The user can make bulk data upload of a .csv or .xlsx file . It calls the API to upload the data and make the chunk of data and asynchronously insert into the database raw table. After it inserts the data into the raw table , it makes use of similar SQL scripts as above to normalize the data.

## **Visualization**

The javascript library chart.js is used for visualization. Since chart.js has limited charts , I also made use of the D3.js visualization library.

Since javascript is used , all the I/O operation is asynchronous. It renders all the visualization individually and does not wait for other visualization to complete. Each visualization makes use of different API calls , so every visualization is rendered from their own API call.

# Employee Count by Designation

Filter by Department:

All Departments

▼



Fig 3: Employee Count by Designation



# Leave Type Distribution

Legend: Bereavement, Leave Without Pay, Sick, Paternity, Compensatory, Annual, Discretionary, Menstruation

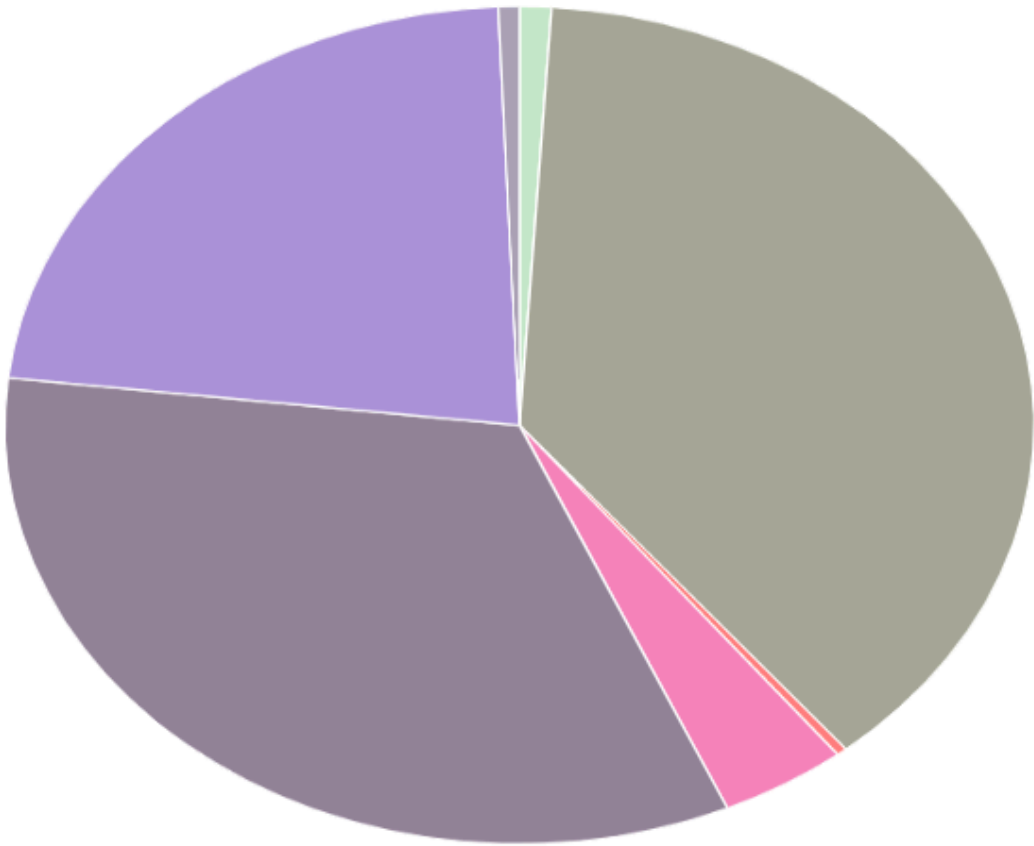


Fig 4: Leave Type Distribution

Start Date:  
07/09/2024

End Date:  
07/23/2024

## Daily Leave Count Of All Department

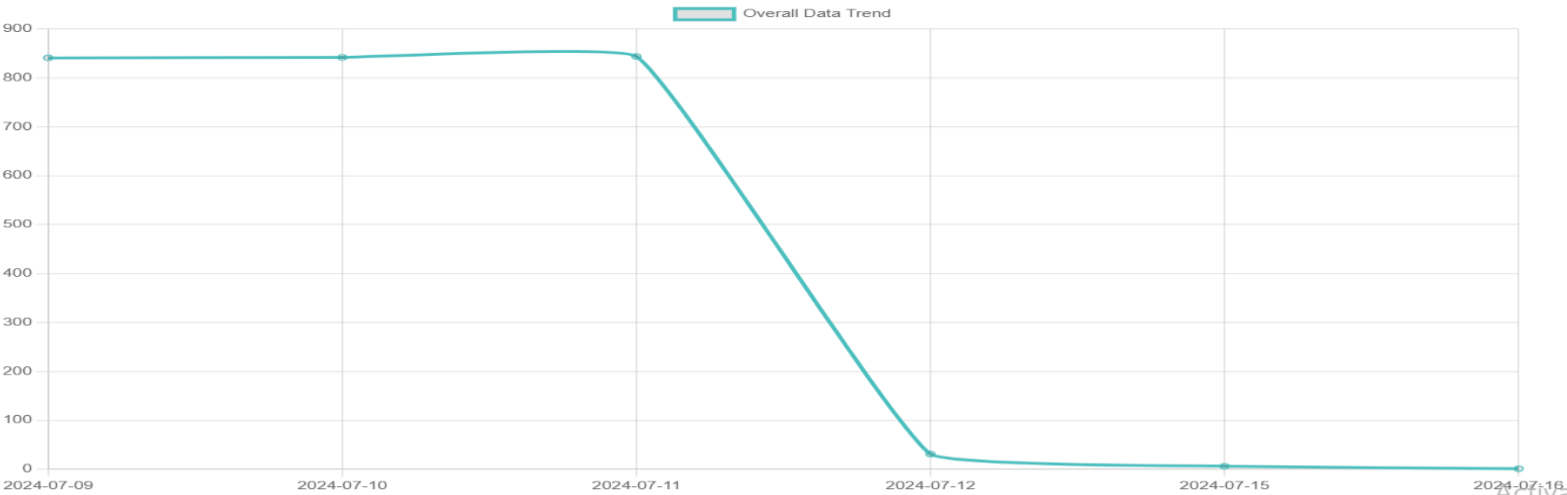


Fig 6: Daily Leave Data Trend

## Leave Calendar Chart

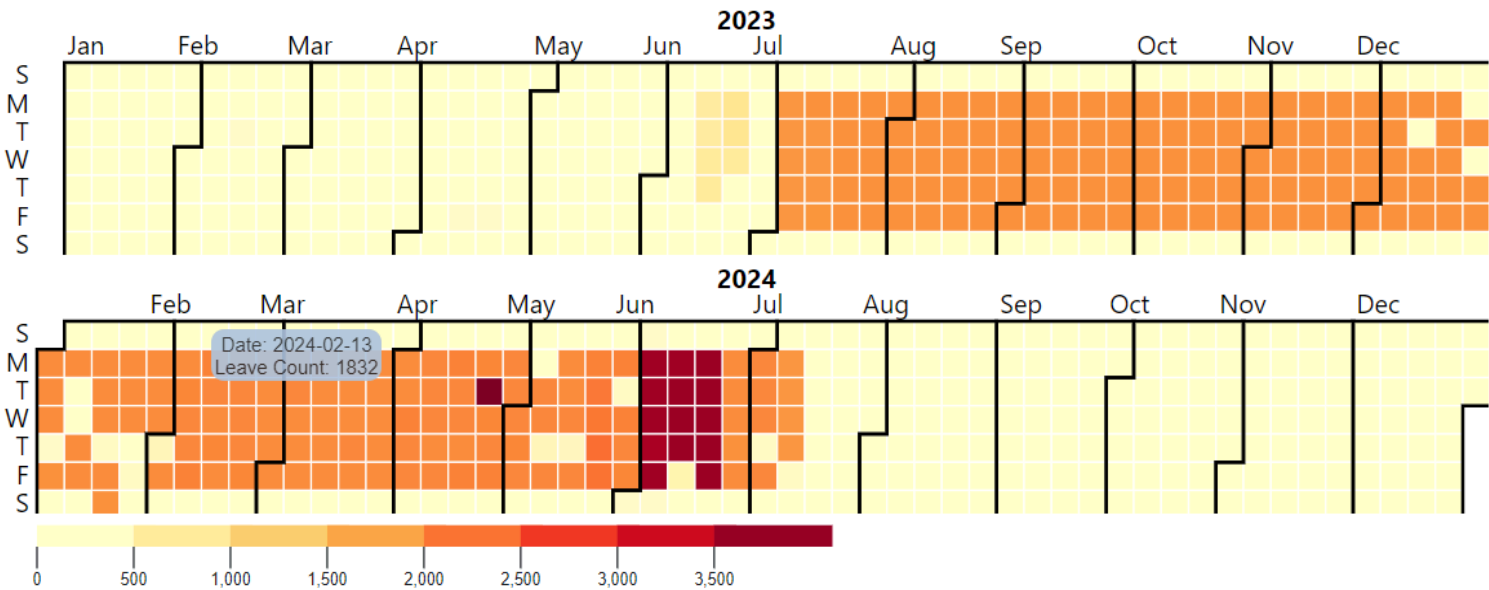
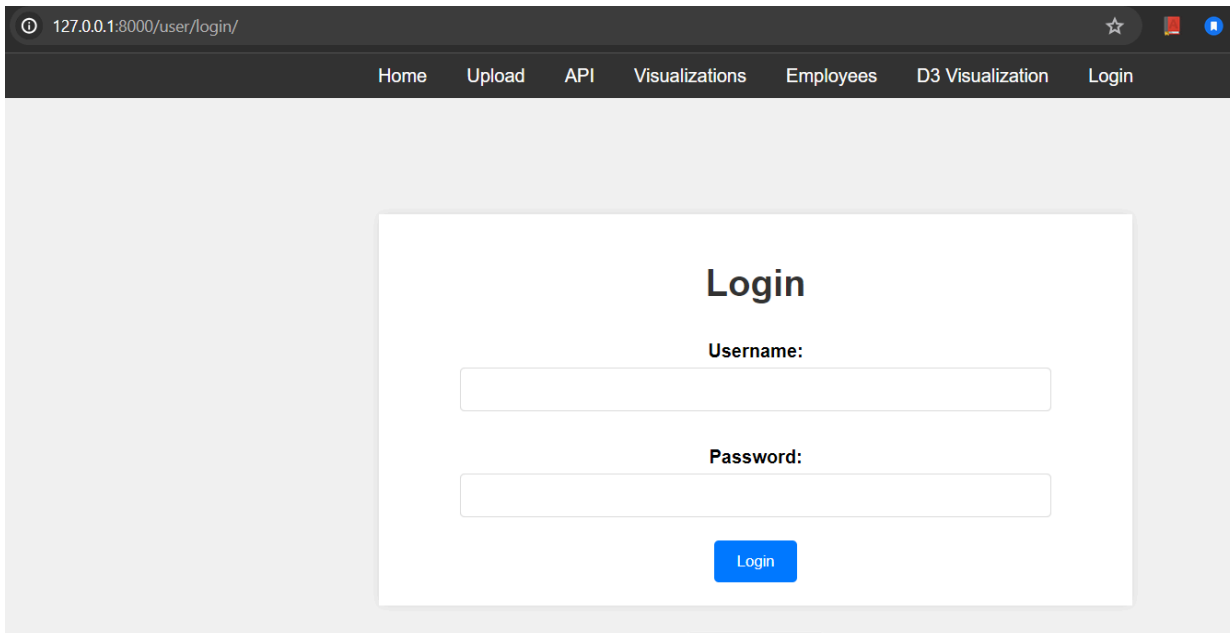


Fig: Calendar Chart

## Application Screenshot



A screenshot of a web browser displaying the login page. The browser's address bar shows the URL "127.0.0.1:8000/user/login/". The page has a dark navigation bar with links: Home, Upload, API, Visualizations, Employees, D3 Visualization, and Login. The main content area is light gray and contains a white login form. The form has the title "Login" in bold, followed by "Username:" and a text input field, then "Password:" and another text input field, and finally a blue "Login" button.

127.0.0.1:8000/user/login/

Home Upload API Visualizations Employees D3 Visualization Login

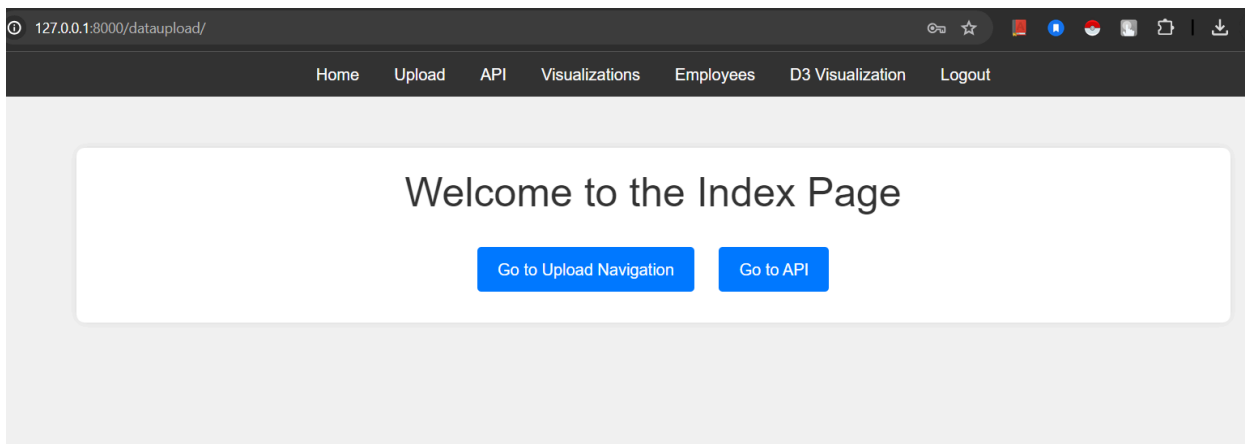
### Login

Username:

Password:

Login

Fig: Login page



A screenshot of a web browser displaying the home page. The browser's address bar shows the URL "127.0.0.1:8000/dataupload/". The page has a dark navigation bar with links: Home, Upload, API, Visualizations, Employees, D3 Visualization, and Logout. The main content area is light gray and contains a white box with the text "Welcome to the Index Page" and two blue buttons: "Go to Upload Navigation" and "Go to API".

127.0.0.1:8000/dataupload/

Home Upload API Visualizations Employees D3 Visualization Logout

## Welcome to the Index Page

Go to Upload Navigation Go to API

Fig: Home page

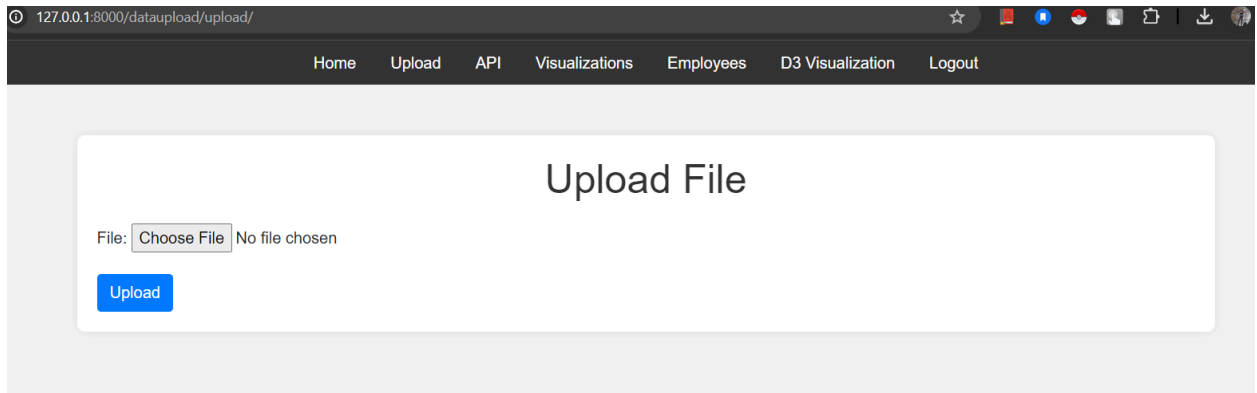


Fig: CSV upload page to DB

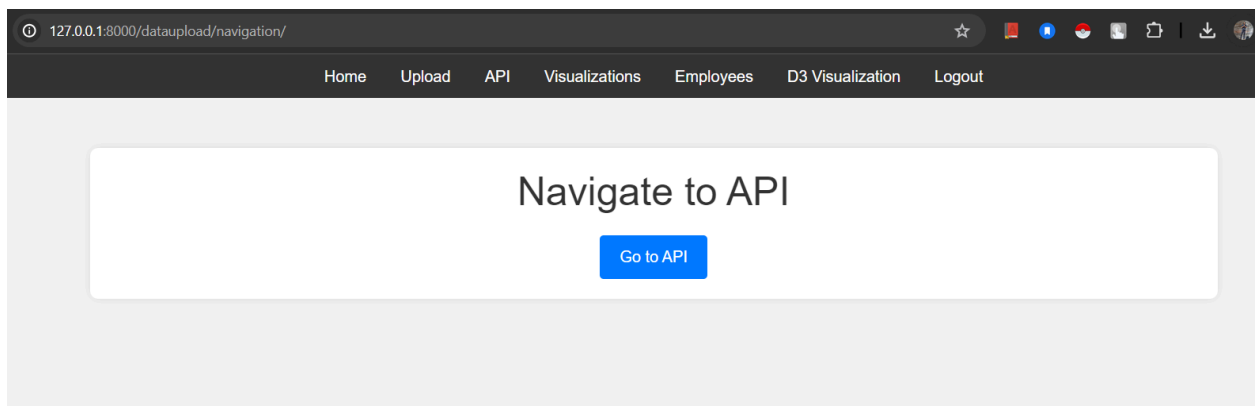


Fig: API data fetch and upload to DB page

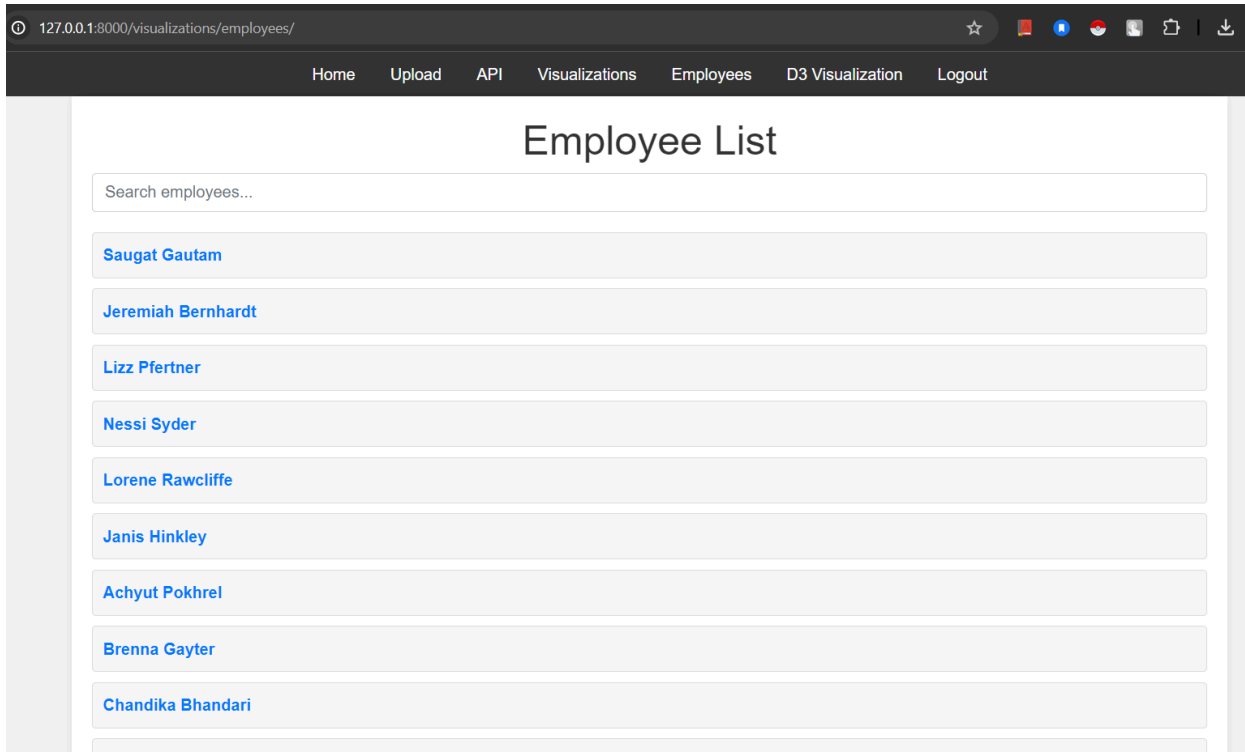


Fig: Employee List Page

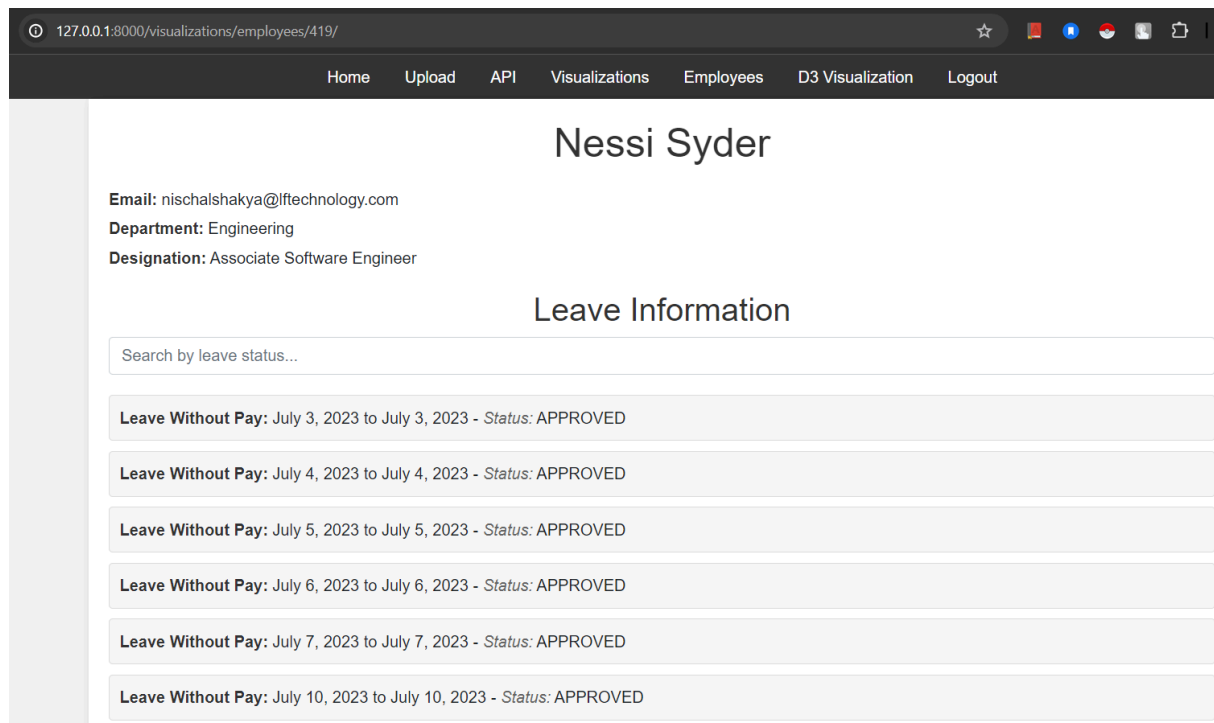


Fig: Employee Details Page

## Gantt Chart

The list of the tasks:

Task 1. Set up django project and database

Task 2. Check out the API data ( data exploration) and create models in django as required.

Task 3. Fetch data from the API and post it to the database.

Task 4. Make a proper template and routing path and views for the process of fetching data and posting it to the database and optimize the Task process.

Task 5. Set up the process for the bulk upload of the leave data through csv or excel file.

Task 6. Set up the login for the user(authentication)

Task 7. Make template for the login page

Task 8. Testing the authentication.

Task 9. Prepare schemas and views for the data visualization

Task 10. Make a separate app (data visualization) in the django for visualization of the data and connect it with the database

Task 11. Make use of the visualization tool to visualize the leave data.

Task 12. Optimize the process wherever necessary

Task 13. Documentation

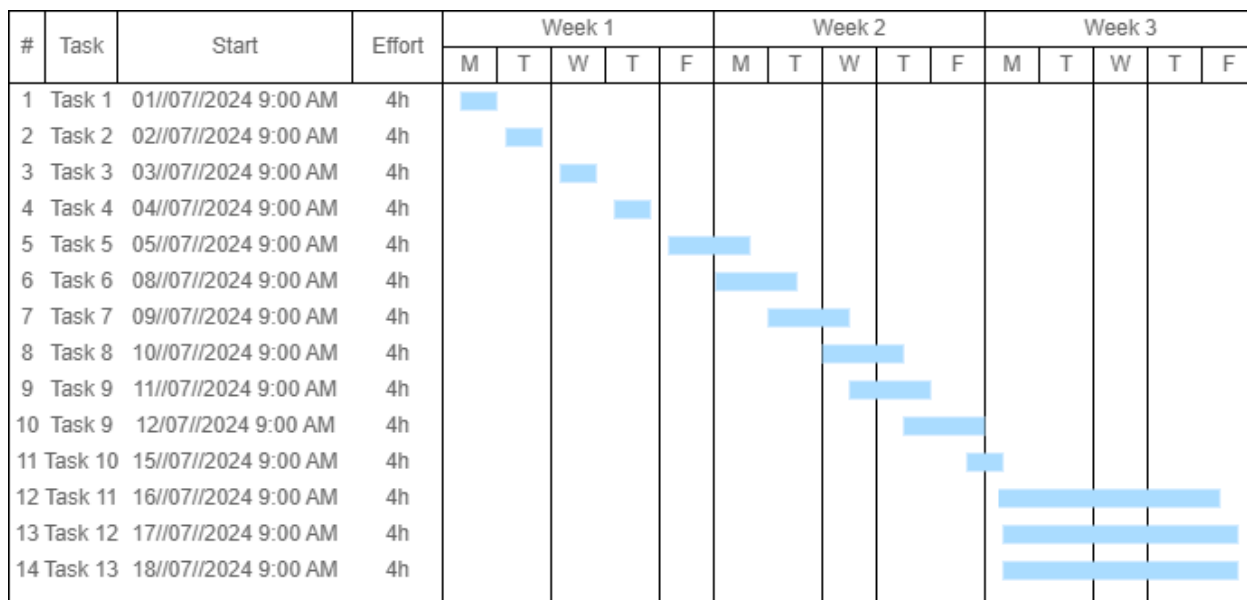


Fig: Gantt Chart