DEVELOPER GUIDE

This section instructs how to start the recommendation system from project source code from Github on 2 environments: Development and Production (Ubuntu OS). Furthermore, this section provides in detail the project structure (backend and frontend), describing how to update and adjust more modules and functionalities for the system.

1. Package Installation

This part lists out all software packages (programming language, database management system, project source code manager, etc.), which serves for downloading the system source code, running the project and maintaing the system's data. It is not required to install all software packages (some packages serving for Production environment).

Software	Description	Environment	Reference link
Python	The recommendation system (Backend) is developed on Django, a Python web framework. The recommended Python version is 3.X	Development, Production	https://www.digitalocean.com/community/tutorials/how-to-install-python-3-and-set-up-a-programming-environment-on-an-ubuntu-20-04-server
pip	A Python package manager, which allows to install and manage packages that aren't part of the Python standard library	Development, Production	https://linuxize.com/post/how-to-install-pip-on-ubuntu-20.04/
virtualenv	A tool to create isolated Python environments, which allows to creates a folder which contains all the necessary executables to use the	Development, Production	https://www.liquidweb.com/kb/how-to-setup-a-python-virtual-environment-on-ubuntu-18-04/

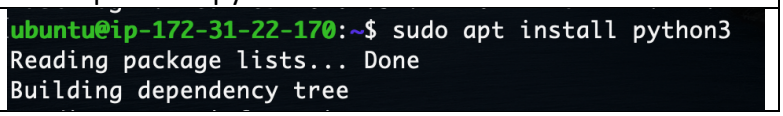
	packages that a Python project would need.		
NodeJS	The graphical user inerface (where system users can access on the browser to perform system functions (data importing, machine learning training, recommendation retrieving, etc.) is built on ReactJS, a Javascript front-end frameword. NodeJS would executes the JavaScript code, which build the interface.	Development, Production	https://www.geeksforgeeks.org/installation-of-node-js-on-linux/
npm	A JavaScript package manager, which allows to install and manage packages that aren't part of the Javascript standard library	Development, Production	https://www.geeksforgeeks.org/installation-of-node-js-on-linux/
MySQL	The recommendation system requires a database management system to keep system data (user, product information,	Development, Production	https://www.digitalocean.com/community/tutorials/how-to-install-mysql-on-ubuntu-20-04

	etc.). Currently, the system supports the connection to MySQL database.		
Git	A free and open source distributed version control system designed to handle everything from small to very large projects (storing source code in repositories, tracking source code changes, etc)	Development, Production	https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
pm2	A daemon process manager that keeps applications running. In production environment, pm2 is used to maintain running the front-end (the graphical user interface) application.	Production	https://faun.pub/deploy-react-application-to-aws-ec2-using-pm2-and-nginx-7a051fd6ea06. Search the download pm2 part in this link.
NGINX	An open-source web server, that can also be used as a reverse proxy which redirect all user requests from	Production	https://lowendbox.com/blog/how-to-replace-apache-with-nginx-on-ubuntu-18-04/

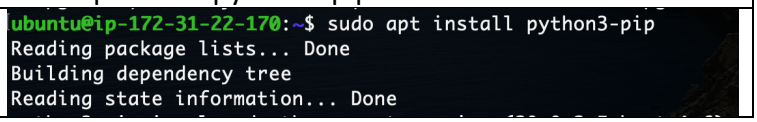
	outside a host server (where hosts the recommendation system and the graphical user interface applications) to running applications.		

Example: (The following part lists out step by step examples about the package installation on Ubuntu LST20.4 operating system, for both development and production)

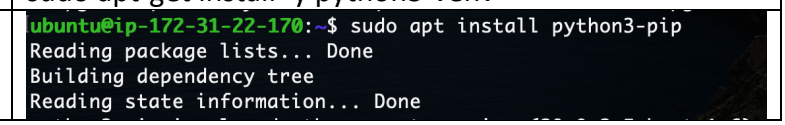
a. Python3:

Command	sudo apt install python3
Example	

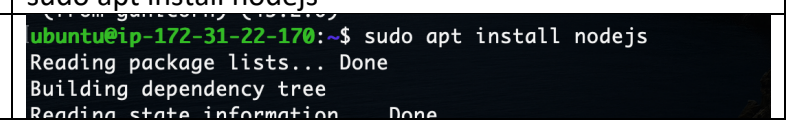
b. Pip:

Command	sudo apt install python3-pip
Example	

c. Virtualenv

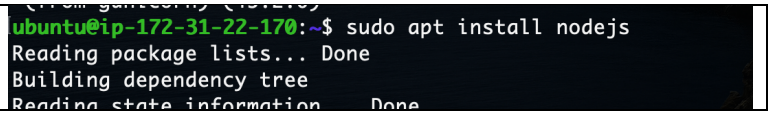
Command	sudo apt-get install -y python3-venv
Example	

d. NodeJS

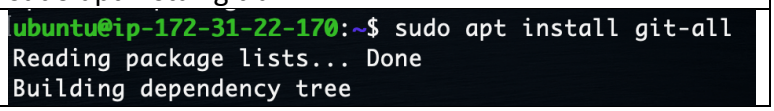
Command	sudo apt install nodejs
Example	

e. npm

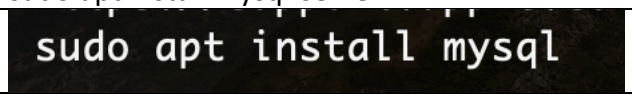
Command	sudo apt install npm

Example	

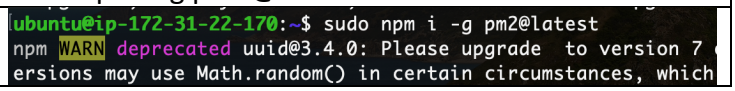
f. git

Command	sudo apt install git-all
Example	

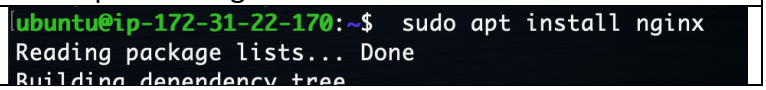
g. MySQL

Command	sudo apt install mysql-server
Example	

h. pm2 (Production only)

Command	sudo npm i -g pm2@latest
Example	

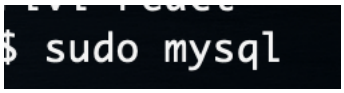
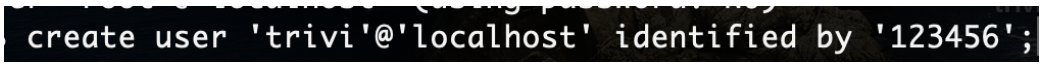
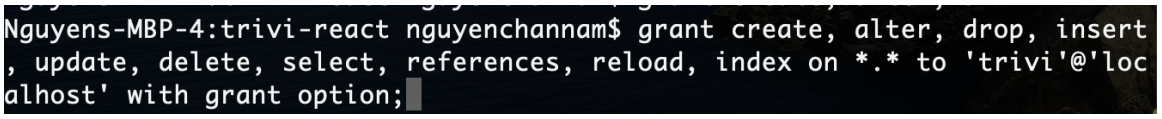
i. NGINX (Production only)

Command	Following the list of commands: sudo systemctl stop apache2 sudo systemctl disable apache2 sudo apt remove apache2 sudo apt autoremove sudo apt clean all && sudo apt update && sudo apt dist-upgrade sudo apt install nginx
Example	

2. Setup and Configuration

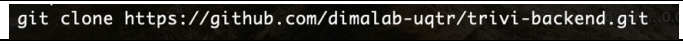
After the packages installation, this part illustrates how clone the project source code, install additional packages for the system, set up database, connect the recommendation system – the graphical user interface – the database.

a. Create MySQL user and system database: This part instruct how to create a MySQL user account, initialize a system database and grant user permissions to read, create, update, delete the system database. All database information (user account, database name) generated need to be keep carefully, since we reuse these information for the recommendation-database connection in the following steps.

Step	Description	Command/Example
1	Connect to the MySQL command line interface	Command: sudo mysql Example: 
2	Create user account	Command: create user '<db_user>'@'localhost' identified by '<db_password'; Example: 
3	Grant permission for user account	Command: grant create, alter, drop, insert, update, delete, select, references, reload, index on *.* to '<db_user>'@'localhost' with grant option; Example: 
4	Update all database configuration	Command: flush privileges; Example: 
5	Create system database	Command: create database <db_name> Example: 
6	Exit	Command: exit;

b. Cloning project source code: The source code for the recommendation system and the graphical user interface are stored in 2 Github repositories, which we need to clone on your machine (development environment) or on the host machine (production environment).

   i. The recommendation system:

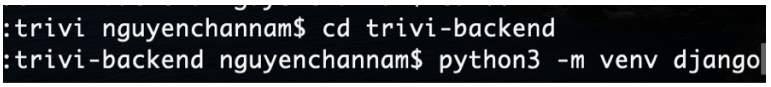
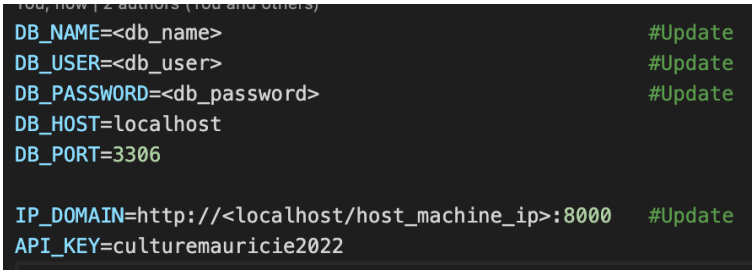
Command	git clone https://github.com/dimalab-uqtr/trivi-backend.git
Example	

ii. The graphical user interface:

Command	git clone https://github.com/dimalab-uqtr/trivi-react.git
Example	`git clone https://github.com/dimalab-uqtr/trivi-react.git`

After cloning 2 repositories, there will be 2 folders, trivi-backend (the recommendation system) and trivi-react (the graphical user interface).

c. Configure the recommendation system: The following sub-part instructs how to configure some information for the recommendation system before running.

Step	Description	Command/Example
1	Create a Python virtual enviroment for the recommendation system at the root folder (trivi-backend/)	Command: python3 -m venv <env_name> Example : 
2	Install additional Python packages	Command: pip install -r requirements.txt Example: 
3	Go to the folder /trivi-backend/recommender	Command: cd recommender Example: 
4	Open the file /trivi-backend/recommender/.env In this file, some information about database are requires to change based on your database information (database name, user, password). If the recommendation system is deployed on the development environment, the IP_DOMAIN field would be set “localhost”. Otherwise, the deployed environment is production, this field would be set the IP4_ADDRESS of the host machine.	File:  Update example:

		DB_NAME=trivi DB_USER=root DB_PASSWORD=cnam0203 DB_HOST=localhost DB_PORT=3306 IP_DOMAIN=http://localhost:8000 API_KEY=culturemauricie2022
5	Next, all data models (tables) that are requires to store system data have been already defined their schemas (fields, data types) but not exist yet in the database, they need to be create (migrate) into the database. At the path /trivi-backend/recommender/, following commands	Command 1: python3 manage.py makemigrations dimadb Example 1: recommender nguyenchannam$ python3 manage.py makemigrations dimadb Command 2: python3 manage.py migrate Example 2: recommender nguyenchannam$ python3 manage.py migrate
6	Next, create a user account for the system (this user account would be used for login the graphical user interface to perform the system functions). At the path /trivi-backend/recommender/, following commands	Command: python3 manage.py createsuperuser Example: recommender nguyenchannam$ python3 manage.py createsuperuser

d. Configure the graphical user interface: The following sub-part instructs how to configure some information for the graphical user interface before running.

Step	Description	Command/Example
1	At the root folder (trivi-react/), Install additional Python packages	Command: npm install Example: trivi-react nguyenchannam$ npm install
2	Open the file /trivi-react/src-constants/utils.py and the	Example:

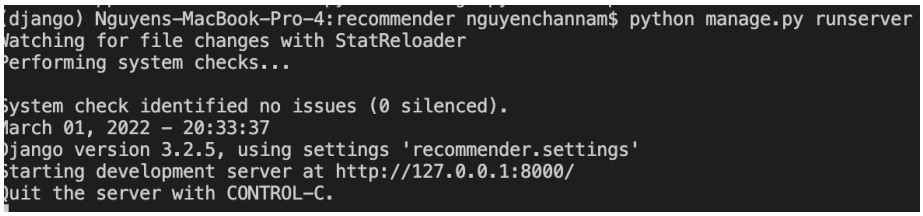
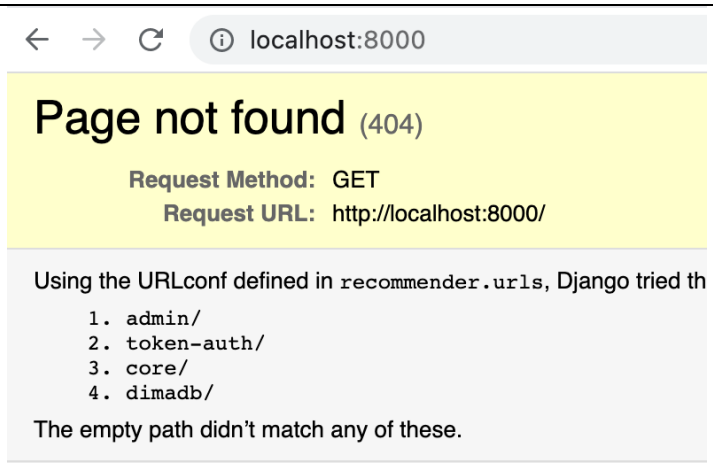
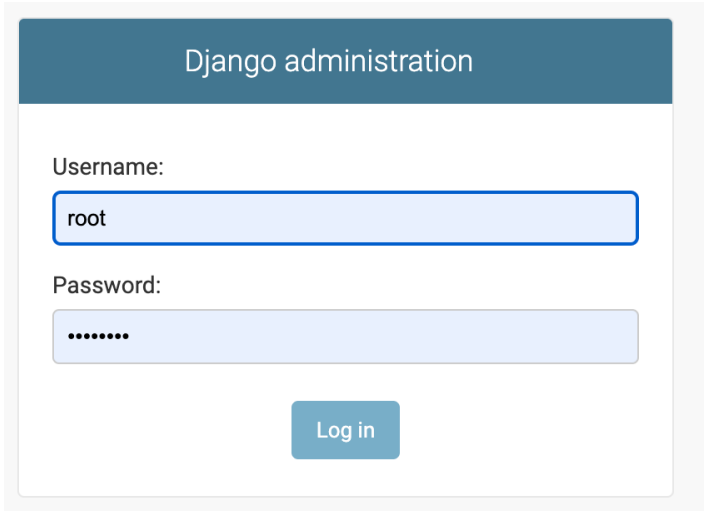
	update the domainPath (the address of the recommendation system that we just configure). If on development environement, set “localhost” and vice versa, set ip4_address of the host machine.	`domainPath = "http://<localhost/host_machine_ip>:8000/"`
		
		Update:
		
		`domainPath = "http://localhost:8000/"`

3. Deployment:

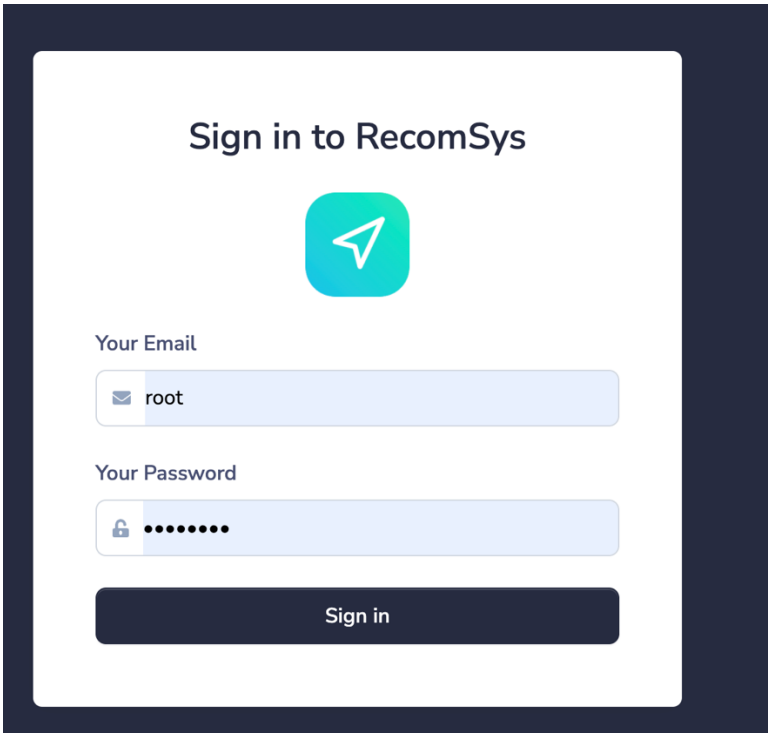
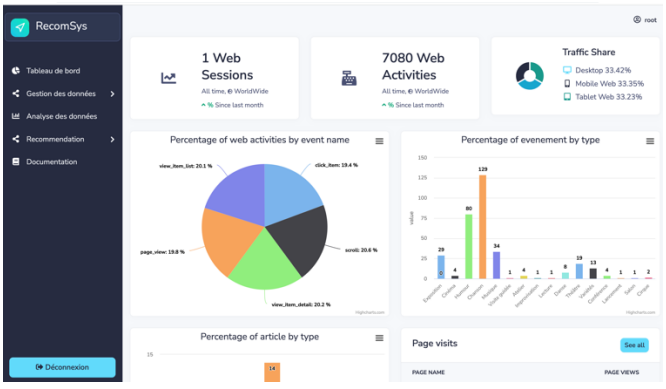
This part instructs how to deploy the recommendation system and the graphical user interface on 2 environments: development and production.

a. Development:

i. The recommendation system:

Step	Description	Command/Example
1	At the path '/trivi-backend/recommender/', starting the application by the following command	Command: python3 manage.py runserver Example : 
2	Open a browser, type the address of the recommendation system (localhost:8000) to check whether it runs or not. If the response page is same as the next image, the recommendation have run.	
3	Try to login the recommendation system by browsing the address: localhost:8000/admin. Then, input user account that was just created in previous steps.	

ii. The graphical user interface:

Step	Description	Command/Example
1	At the path '/trivi-react', starting the application by the following command	Command: npm start Example : 
2	Open a browser, type the address of the graphical user interface (localhost:3000) to check whether it runs or not. If the response page is same as the next image, the recommendation have run.	
3	Try to login the graphical user interface by browsing the address: localhost:8000/admin. Then, input user account that was just created in previous steps. If we can login the Homepage, it means that the graphical user interface communicates sucessfully with the recommendation system.	

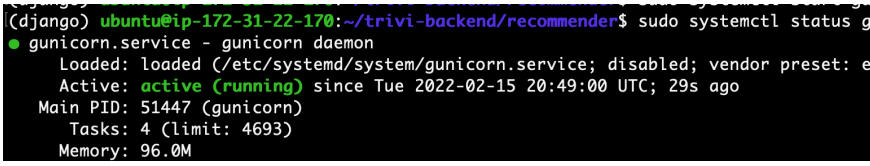
b. Production:

i. The recommendation system:

Step	Description	Command/Example
1	Setup Gunicorn, which would serve as an interface to the recommendation system, translating user requests in HTTP to Python calls that the recommendation system can process. Create a systemd unit file by following the command	Command: sudo nano /etc/systemd/system/gunicorn.service Example : `sudo nano /etc/systemd/system/gunicorn.service`

2	Next, insert the following source code into the systemd unit file. These commands would be automicatically executed when the host machine is turned on. We need to modify some variable (path to project, host machine user, etc). After inserting source code, save file and exit.	Source code:
		
		[Unit]
		Description=gunicorn daemon
		After=network.target
		
		[Service]
		User=<host_machine_user>
		Group=www-data
		WorkingDirectory=<path_to_project>/trivi-backend/recommender
		ExecStart=<path_to_project>//trivi-backend/django/bin/gunicorn
		workers 3 --timeout 3600 --bind unix: <path_to_project>//trivi-backend/recommender/recommender.sock recommender.wsgi:ap
		
		[Install]
		WantedBy=multi-user.target
		
		We need to insert host_machine_user (normally “root” on Ubuntu
		path_to_project, where contains the recommendation system proj
		
		Example:
		
		`[Unit]`
		`Description=gunicorn daemon`
		`After=network.target`
		
		`[Service]`
		`User=ubuntu`
		`Group=www-data`
		`WorkingDirectory=/home/ubuntu/trivi-backend/recommender`
		`ExecStart=/home/ubuntu/trivi-backend/django/bin/gunicorn --access-logfile - --workers 3 --bind unix:/home/ubuntu/trivi-backend/recommender/recommender.sock recommender.wsgi:application`
		
		`[Install]`
		`WantedBy=multi-user.target`

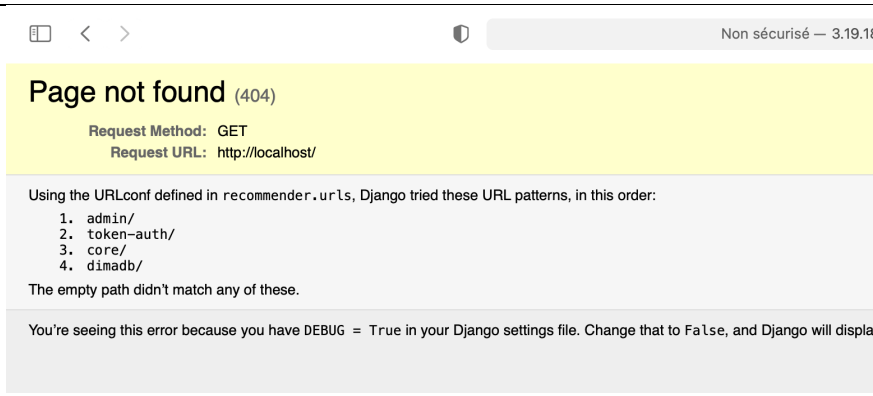
3	Next, start the Gunicorn service by the following command:	Command: sudo systemctl start gunicorn Example: `sudo systemctl start gunicorn`

4	Now, we can check whether the Gunicorn application is running or not by the following command (Active status means that the application is running):	Command: sudo systemctl status gunicorn Example: 
5	Next, we need to create a new config sites-available file by the following command:	Command: sudo nano /etc/nginx/sites-available/trivi-backend Example : 

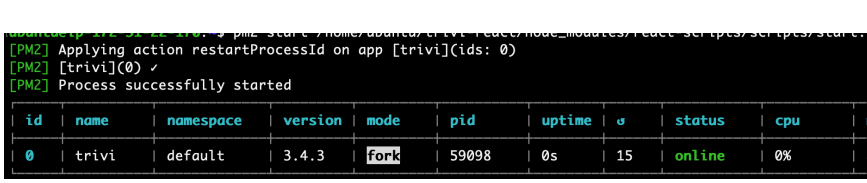
6	Next, insert the following source code. This source code allows NGINX reverse user requests (requests that destinates to a specified port on the host machine), to the Gunicorn application.	Source code:
		server {
		listen 8000;
		server_name <host_machine_ip4_address>;
		
		location = /favicon.ico { access_log off; log_not_found off; }
		location /static/ {
		root <path_to_project>/trivi-backend/recommender;
		}
		
		location / {
		proxy_pass http://unix:/<path _to_project>/trivi-backend/recommender/recommender.sock;
		}
		}
		
		We need to edit <host_machine_ip4_address> and <path_to_proje
		
		Example:
		 

7	Then, create a symbolic link by the following command	Command: sudo ln -s /etc/nginx/sites-available/trivi-backend /etc/ Example : `sudo ln -s /etc/nginx/sites-available/trivi-backend /etc/n`

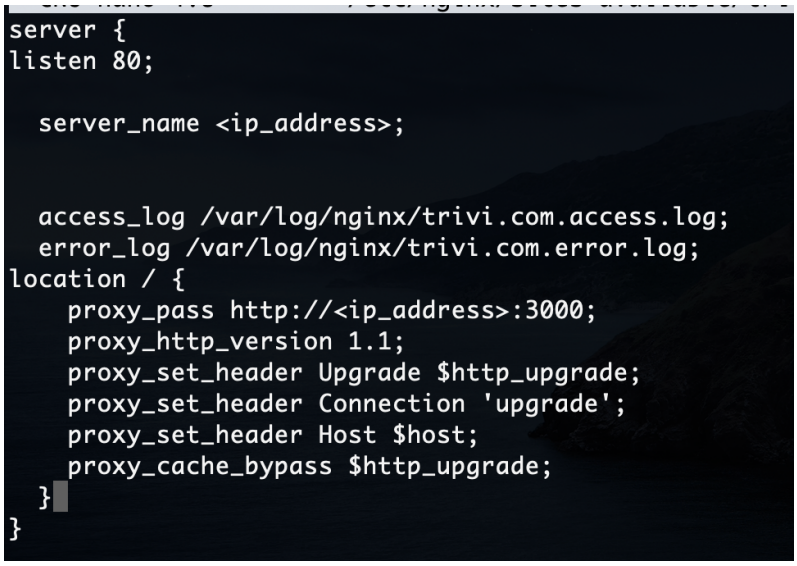
8	Next, run following commands for restating the NGINX. If the command “sudo nginx -t” results “OK”, it means that the above configure file has right syntax.	Command 1: sudo nginx -t
		Command 2: sudo systemctl restart nginx
		
		Example:
		
		`(django) ubuntu@ip-172-31-22-170:/etc/nginx/sites-enabled$ sudo nginx -t`
		`nginx: the configuration file /etc/nginx/nginx.conf syntax is ok`
		`nginx: configuration file /etc/nginx/nginx.conf test is successful`
		`(django) ubuntu@ip-172-31-22-170:/etc/nginx/sites-enabled$ sudo systemctl res`

9	Now, we can check whether the recommender system is running or not by opening a browser and then connect to the <ip4_address_host_machine>:8000, for example: http://3.19.188.61:8000/. If the browser displays like the image below, the recommender system is running.	

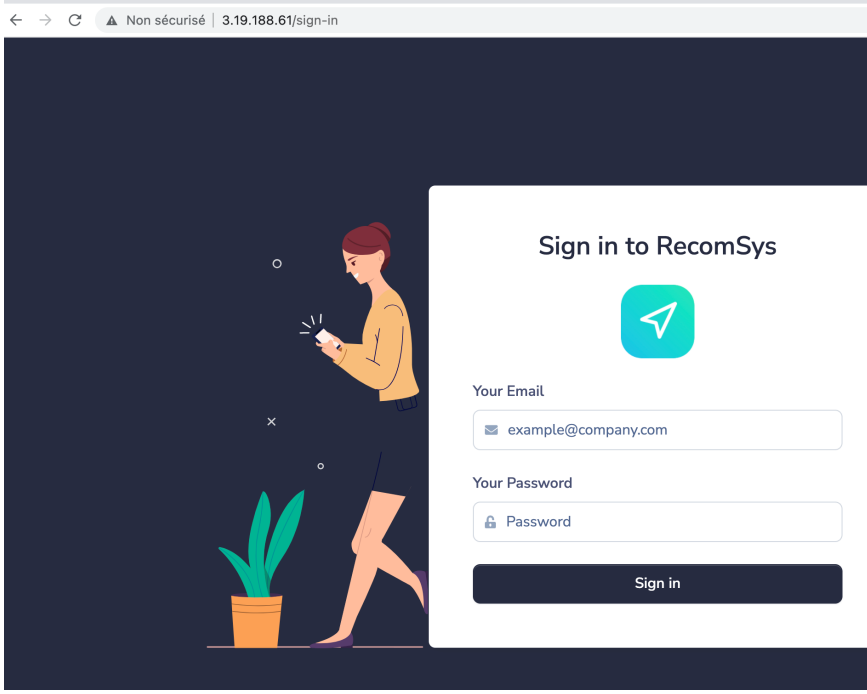
ii. The graphical user interface:

Step	Description	Command/Example
1	Start the graphical user interface by PM2 by the following command	Command: pm2 start <path_to_project>/trivi-react/node_module scripts/scripts/start.js --name trivi Example : `pm2 start /home/ubuntu/trivi-react/node_modules/react-scripts/script`
2	After executing the command, the command line interface will display the status of the graphical user interface. If the status is “online”, it means the graphical user interface is started successfully by PM2	
3	Next, we need to create a new config sites-available file by the following command:	Command: sudo nano /etc/nginx/sites-available/trivi-react Example :

		sudo nano /etc/nginx/sites-available/

6	Next, insert the following source code. This source code allows NGINX reverse user requests (requests that destinates to a specified port on the host machine), to the PM2 application.	Source code:
		server {
		listen 80;
		server_name <ip4_address_host_machine>;
		
		access_log /var/log/nginx/trivi.com.access.log;
		error_log /var/log/nginx/trivi.com.error.log;
		location / {
		proxy_pass http:// <ip4_address_host_machine>:3000;
		proxy_http_version 1.1;
		proxy_set_header Upgrade $http_upgrade;
		proxy_set_header Connection 'upgrade';
		proxy_set_header Host $host;
		proxy_cache_bypass $http_upgrade;
		}
		We need to edit <host_machine_ip4_address>
		
		Example:
		 

7	Then, create a symbolic link by the following command	Command: sudo ln -s /etc/nginx/sites-available/trivi-react /etc/ngi Example : 
8	Next, run following commands for restating the NGINX. If the	Command 1: sudo nginx -t Command 2: sudo systemctl restart nginx

	command “sudo nginx -t” results “OK”, it means that the above configure file has right syntax.	Example: 
9	Now, we can check whether the recommender system is running or not by opening a browser and then connect to the <ip4_address_host_machine>, for example: http://3.19.188.61/. If the browser displays like the image below, the recommender system is running.	

iii. The NGINX config file: NGINX serves as a reverse proxy, which has some rules about the maximum file size that a user request can send to NGINX or about the response timeout. Those parameters is defaultly set, and we can update them. For the recommendation system, user could import a massive data file, so it is recommended to increase values of those parameters for the higher performance.

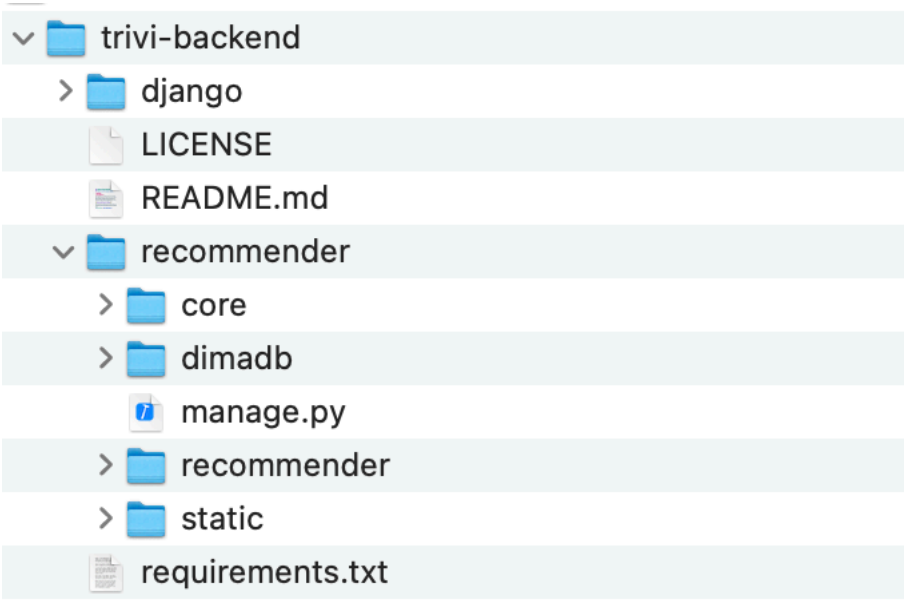
Step	Description	Command/Example
1	Open the NGINX conf file	Command: sudo nano /etc/nginx/nginx.conf Example : 
2	After executing the command, add the following lines inside the http part in the file.	Lines: keepalive_timeout 0;

		types_hash_max_size 2048;
		client_max_body_size 100M;
		proxy_read_timeout 3600;
		proxy_connect_timeout 3600;
		proxy_send_timeout 3600;
		
		Example:
		
		`http {`
		
		`##`
		`# Basic Settings`
		`##`
		
		`sendfile on;`
		`tcp_nopush on;`
		`tcp_nodelay on;`
		`keepalive_timeout 0;`
		`types_hash_max_size 2048;`
		`client_max_body_size 100M;`
		`proxy_read_timeout 3600;`
		`proxy_connect_timeout 3600;`
		`proxy_send_timeout 3600;`
		`# server_tokens off;`

3	Next, run following commands for restating the NGINX. If the command “sudo nginx -t” results “OK”, it means that the above configure file has right syntax.	Command 1: sudo nginx -t
		Command 2: sudo systemctl restart nginx
		
		Example:
		
		`(django) ubuntu@ip-172-31-22-170:/etc/nginx/sites-enabled$ sudo nginx -t`
		`nginx: the configuration file /etc/nginx/nginx.conf syntax is ok`
		`nginx: configuration file /etc/nginx/nginx.conf test is successful`
		`(django) ubuntu@ip-172-31-22-170:/etc/nginx/sites-enabled$ sudo systemctl re`

4. Project Structure:
   This part shows project structures for both the recommendation system and the graphical user interface, that indicates the function of each sub-folder/file. Besides, this part instructs how to add, update, edit system modules which are responsible for system functionalities (for example, modules for item recommdation/modules for data mapping, etc.)

   a. The recommendation system:
      i. Project structure overview:



Folder name / File name	Description
trivi-backend/	The root folder (project folder for the recommendation system)
Trivi-backend/django/	The Virtual environment folder (which contains all python packages)
Trivi-backend/requirements.txt	The file list all python library names, which are requires for the execution of the recommendation system. All python library names listed in this file would be installed and stored in the trivi-backend/django folder
Trivi-backend/recommender/	The main folder contains all source code for the recommendation system.

Trivi-backend/recommender/core/	The folder contains soure code for user authentication
Trivi-backend/recommender/recommender/	The folder contains all setting/configuration files (for example, file settings.py) for the system.
Trivi-backend/recommender/recommender/manage.py	The file contains all execution commands for the system (migrating data schema between the system and the database, starting the system, etc.)
Trivi-backend/recommender/recommender/.env	The file contains environment variables, such as database information, ip address of the host machine, api key used for request authentication
Trivi-backend/recommnder/dimadb	The folder contains all source code for data managamenent and recommendation functionalities. If developer would adjust/modify functionalities, they need to edit/add source code in this folder. The function of each file in this folder would be described in the next part.

ii. Main project folder: The folder /trivi-backend/recommender/dimadb

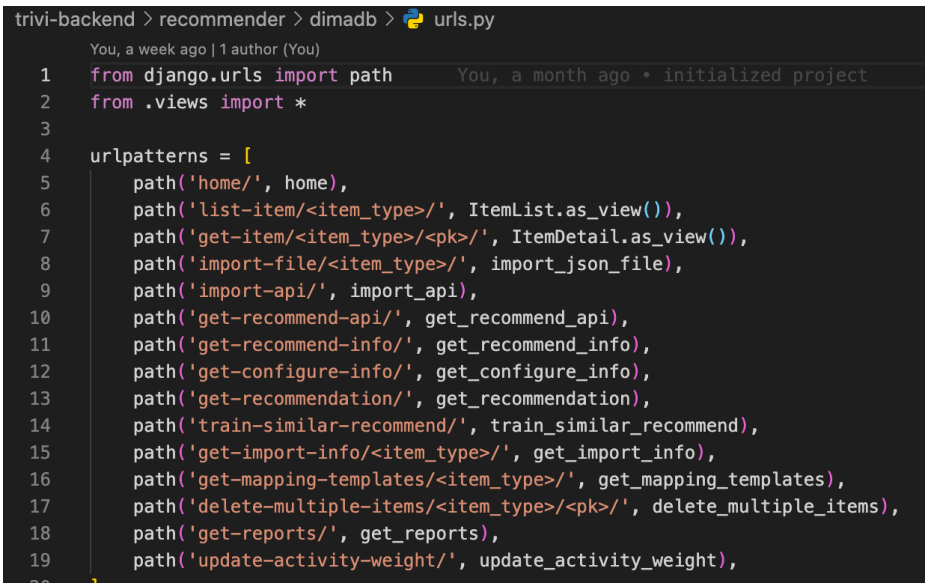
Folder name / File name	Description
urls.py	The file serves as a user request router for the system. Each user request (API) is specified by an URL, this URL is then assigned to map with a logic function (in views.py) to handle (retriving data and responding).
Views.py	The file contains all logic functions which are responsible for processing system functionalities (data mapping, recommendation), responding user requests.
Models.py	The file declares all tables and their of the recommendation system. For example, the system needs to be store data about Events, Products,

	etc., those tables need to be clearly and sufficiently defined about their fields, their meta data in this file.
Configuration/schema_detail.json	The file defines all fields of a database table that could be seen by system users when they need to view in detaill, create, edit, delete a row in that table. Beside fields belonging to that table, we could add more additional fields which are existed in other tables. For example, users would see information about a specified event (these information in the Event Table) and some information about the location of that event (which in the GeoLocation Table). This file has capablities for retrieving all necessary information in many tables.
Configuration/schema_table.json	The file defines all fields of a database table that could be seen by system users, same as the file schema_detail.json. However, the number of visible fields are fewer than those in the schema_detail.jsom file and we can not adjust fields existing in other tables (just fields in a specified table).
Configuration/mapping_template.json	The file defines data structure mapping templates between database tables and imported data sources. Each imported data source or each database table has a specified data structure, it requires that imported data need to be organized, restructured to resemble the data structure of database tables.
Model_recommend/	The folder contains source codes for machine learning modules (each file in this folder could contain several machine learning modules related to a specified functionality, such as finding item similarities, etc.). Alll trained models used for recommendation or

	other purposed are also kept in this folder.
Utils.py	The file contains all utility functions (which are frequently resusable, callable).

iii. Detail:

1. /trivi-backend/recommender/dimadb/urls.py:
   This module file is a pure Python source code, which is responsible for mapping between URL path expression (API) to logic functions in the file views.py



   For example, when a user request with url 'localhost:8000/dimadb/home' is sent to the system, this file would assign function 'home' in the file views.py to handle this request.

   From now, when developers want to create a new functionality, they could define a url expression for that functionality in this file, a define a logic function to handle this functionality in the file views.py. This url when it is called would be in format 'http://<ip_address:8000>/dimadb/<url_expression>'.

2. Views.py:

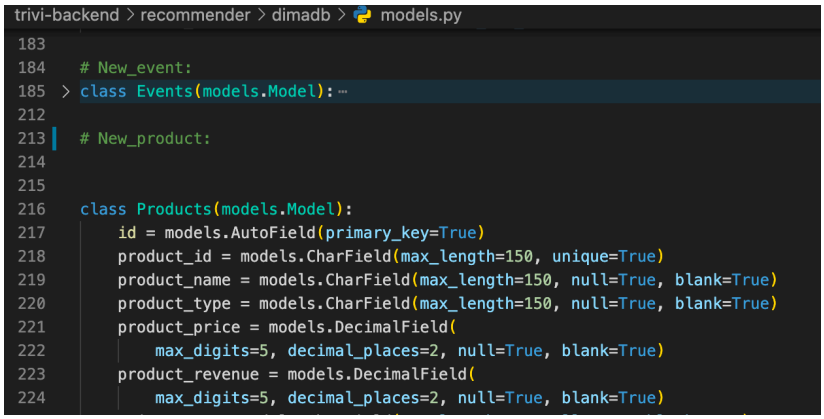
   A view function, or view for short, is a Python logic function that takes a user request and returns a user response. This response can be the HTML contents of a web page, or a redirect, or a 404 error, or an XML document, or an image . . . or anything, really. The view itself contains whatever arbitrary logic necessary to return that response.

```python
@api_view(['GET'])
def get_configure_info(request):
    try:
        similar_train_info = get_similar_train_info()
        web_activity_types = Interaction.objects.values('event_name').
        web_activity_types = [item['event_name'] for item in list(web_a
        existed_web_activity_types = WebActivityType.objects.values('na
        existed_web_activity_types = [item['name'] for item in list(exi
        web_activity_types = web_activity_types + existed_web_activity_
        web_activity_types = list(dict.fromkeys(web_activity_types))

        web_activities_info = {}
        for activity_type in web_activity_types:
            try:
                activity_type_obj = WebActivityType.objects.get(name=ac
                activity_type_obj = model_to_dict(activity_type_obj)
                web_activities_info[activity_type] = activity_type_obj[
            except:
                web_activities_info[activity_type] = 0

        return Response({'similarTrainInfo': similar_train_info, 'webAc
    except Exception as error:
        return Response({'message': error})
```

The image above illlustrate a logic function example in the file views.py. Each called logic function needs to retrieve data and then return a response for each user request.

3. Models.py:

A model is the single, definitive source of information about your data. It contains the essential fields and behaviors of the data you're storing. Generally, each model maps to a single database table.



After developer adjusts/modifies a table, they need to run the 2 commands "python3 manage.py makemigrations dimadb" and then "python3 manage.py migrate", to update schemas in the database.

4. Configuration/schema_detail.json

This file contains fields of each database table (main table) that could be seen by users when users want to view in detail. Besides, there are some additional fields existing in other tables that we could display for users. There are 2 kinds of other tables: many_to_many_tables (m2m_tables) and one_to_many_tables (o2m_tables). If each record in the main table is connected to multiple records in another table by a connected_table, this another table is the m2m_table. By contrast, if each record in the main table is connected to multiple records in another table by without a connected_table, this another table is the o2m_table.

There are some important key properties in the shema_detail.json

Key properties	Description	Data type	Example
Least nested key properties	The arbitrary name of the main table	Object	Event, product
model_name	The name of the main able in the database	String	Events, products
display_name	The alternative name of the main table in the database, which is displayed for users	String	Events, products
fields	The list of fields in the main table, which are displayed for users	List of strings	Id, event_name, etc.
M2m_tables	The list of m2m_tables	List of objects	
connected_table	The connected_table between the main table and the m2m_table, which holds 2 primary keys of each record in the main table and the m2m_table as foreign keys. This key property exists in the m2m_table object	Object	
connected_field1	The name of the field in the connected table that holds the primary key of a record in the main table. This key property exists in the m2m_table object	String	Event_id
connected_field2	The name of the field in the connected table that holds the primary key of a record in the m2m table. This key property exists in the m2m_table object	String	Location_id
o2m_tables	The list of o2m_tables	List of objects	
connected_field	The name of the field in the o2m_table that holds the primary key of a record in the main table. This key	String	Event_id

	property exists in the o2m_table object		

5. Configuration/schema_table.json:

This file contains fields of each database table (main table) that could be seen by users when users want to view the list of multiple records in the table. Differently from the schema_detail.json, we not display additional

fields existing in other for users. There are some important key properties in the shema_detail.json

Key properties	Description	Example
Least nested key properties	The arbitrary name of the main table	Event, product
model_name	The name of the main table in the database	Events, products
fields	The list of fields in the main table, which are displayed for users	Id, event_name, etc.
view_detail	The boolean field that allows users to view a record in detail.	True, false

6. Configuration/mapping_template.json

The file defines mapping structure templates between database tables and imported data. Each imported data sources has a different data structure, it requires an automatic mechanism having capability to map with database table. The a general format for this file is similar to the schema_detail.json, however, there are few of changes in the “fields” key property.

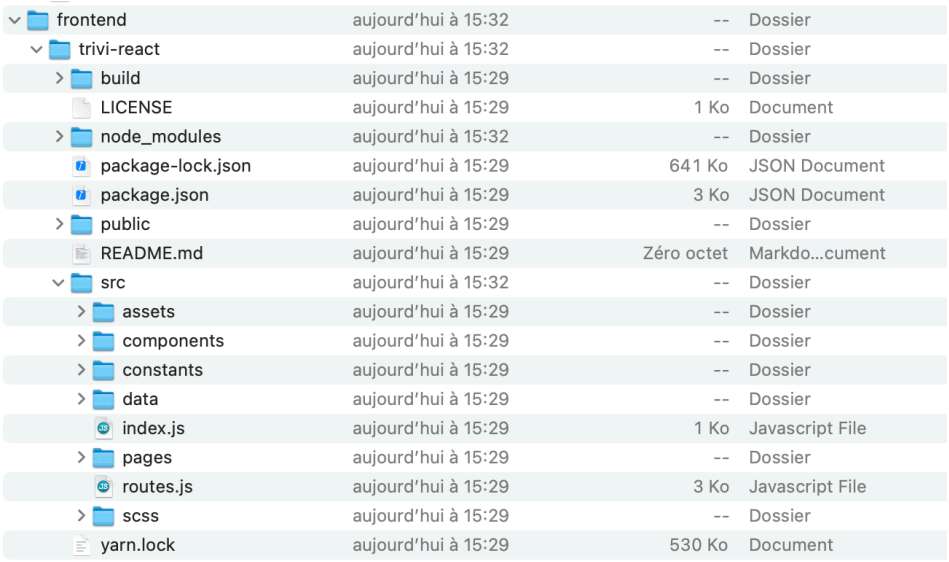
Key properties	Description	Data type	Example
Least nested key properties	The name of the main table	Object. Each key properties in this object is a data source name	Event, product

2nd Leasted nested key properties	The source name of imported data	Object	Default, Google
model_name	The name of a table in the database	String	Events, products
is_reformat	The Boolean field that check whether imported data in this data source need to be reformatted or not	Boolean	Events, products
fields	The list of fields in a database table, which are displayed for users	Object. Each key properties in this object is a field name in the table	Id, event_name, etc.
type	The data type of the field	String	String, number, datetime, etc.
source_name	The key property name in the imported json file that is mapped with the field in the database table	String	Id, title, etc.
M2m_tables	The list of m2m_tables	List of objects	
connected_table	The connected_table between the main table and the m2m_table, which holds 2 primary keys of each record in the main table and the m2m_table as foreign keys. This key property exists in the m2m_table object	Object	
connected_field1	The name of the field in the connected table that holds the primary key of a record in the main table. This key property exists in the m2m_table object	String	Event_id
connected_field2	The name of the field in the connected table that holds the primary key of a record in the m2m table. This key property exists in the m2m_table object	String	Location_id
o2m_tables	The list of o2m_tables	List of objects	
connected_field	The name of the field in the o2m_table that holds the primary key of a record in the main table. This key	String	Event_id

	property exists in the o2m_table object		

b. The graphical user interface:

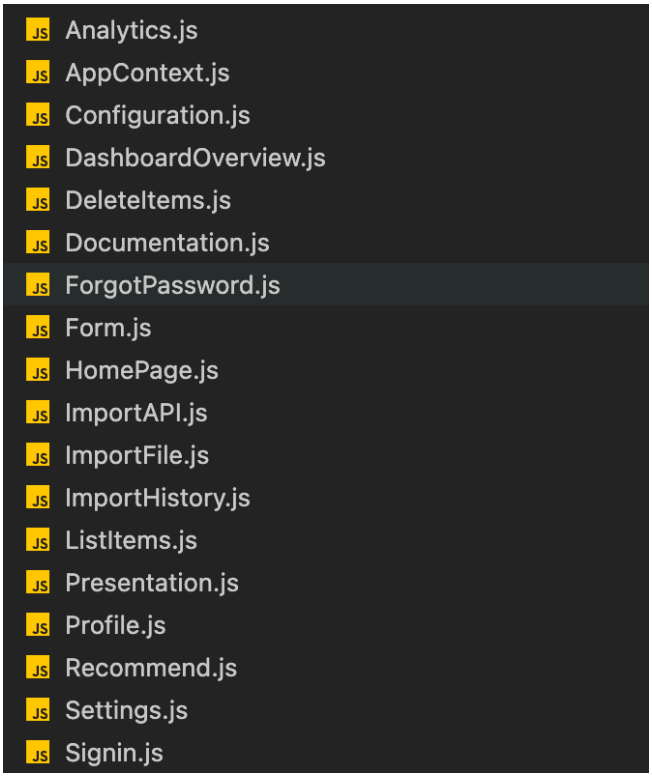
i. Project Structure Overview:



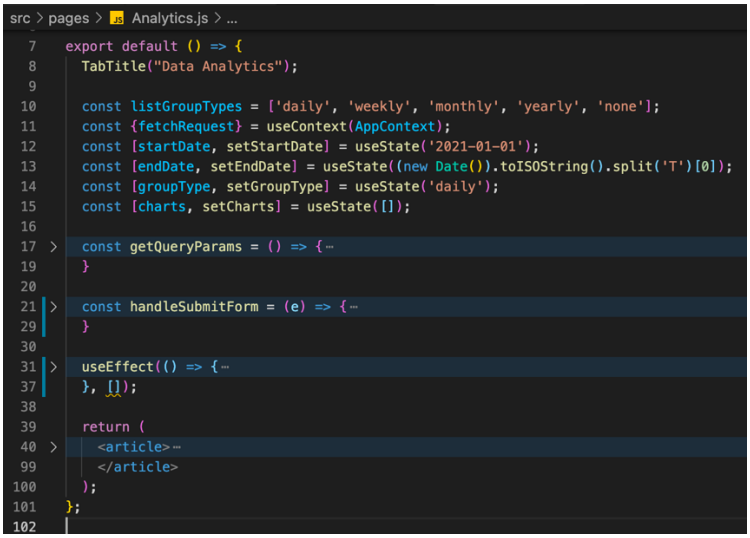
Folder Name/ Filename	Description
trivi-react	The root folder (project folder)
trivi-react/package.json	The file contains all JavaScript library packages names required for the project.
trivi-react/src	The folder contains all source code used for graphical user interfaces
trivi-react/node_modules	The folder contains all installed JavaScript library packages.
trivi-react/src/scss	The folder contains "css/scss" files used for styling graphical user interface
trivi-react/src/assets	The folder contains images, fonts used for graphical user interfaces
trivi-react/src/components	The folder contains all source codes generating components (charts, navbar, sidebar, footer, etc.), which are used to constructed pages.
trivi-react/src/constants	The folder contains util functions which are frequently used when coding, as well as, configuration information (recommender system IP, etc)
trivi-react/src/pages	The folder contains source codes for pages/screens which are displayed for users (HomePage, ListItems, Recommendation, etc)
trivi-react/src/routes.js	The file defines all routers, which reverse each url on the browse bar to a correspondent page.

ii. Main project folder: trivi-react-/src/pages.

This folder contains all source codes for all pages (Screens) for the graphical user interface. Each page is corresponds a file in this folder, however, a file could be resusable for multiple pages.

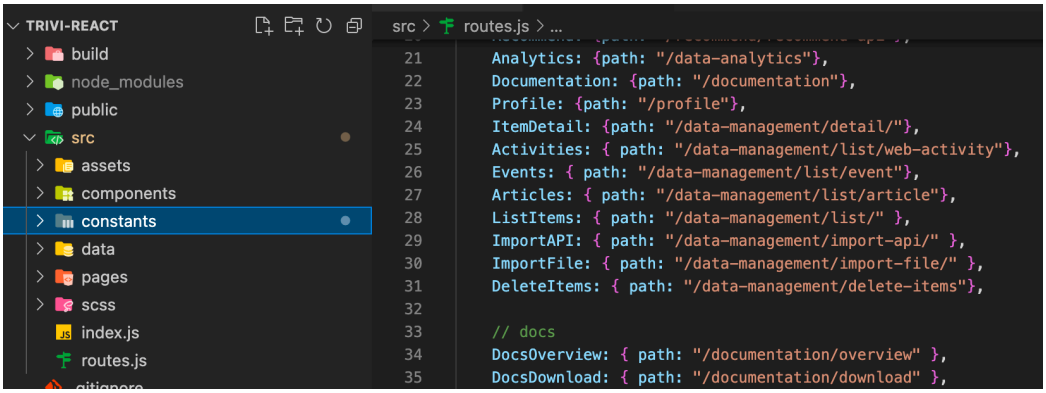


In order to create a new page on the graphical user interface, add a new file for that page in the folder trivi-react/src/page.

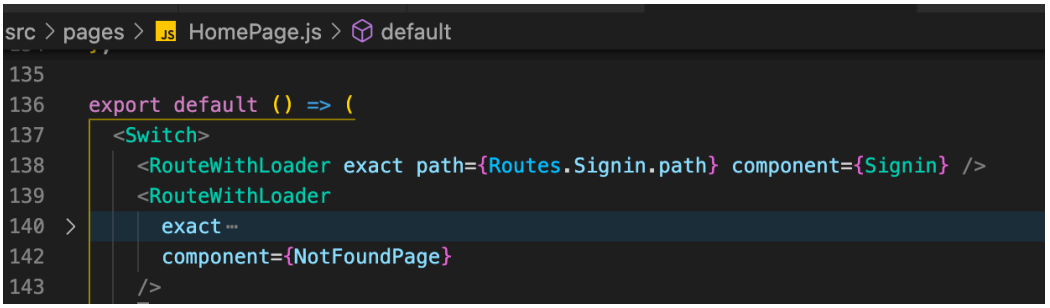


Each file would be exported into a module (export default()) as a component. To show data on that page, we need to retrive data from the recommendation system by calling a HTTP request. This request must be called inside the function useEffect(). To render components + data for that page, we should write code inside the function return().

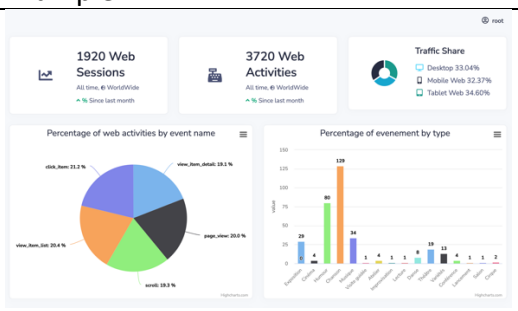
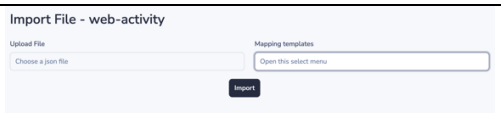
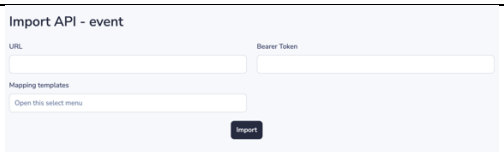
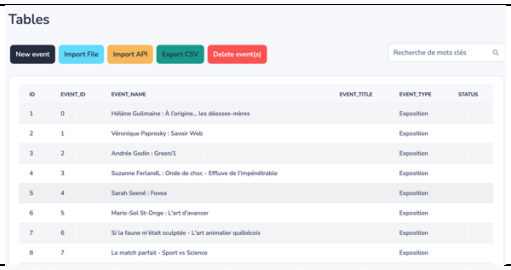
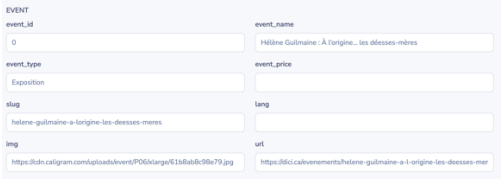
After creating new file for that page, we need to generate an url path for that page, so that system users can access on the browser. To do that, access to the file trivi-react/routes.js and define a new url path:

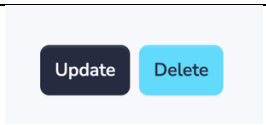
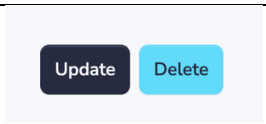
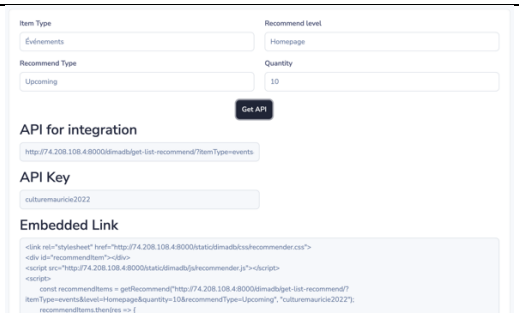
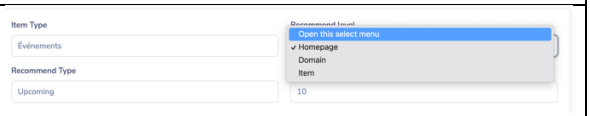


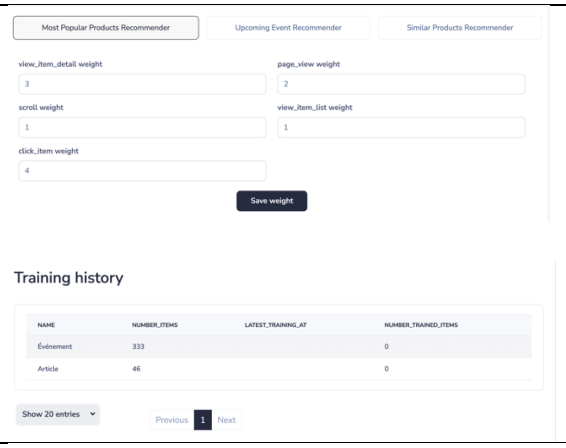
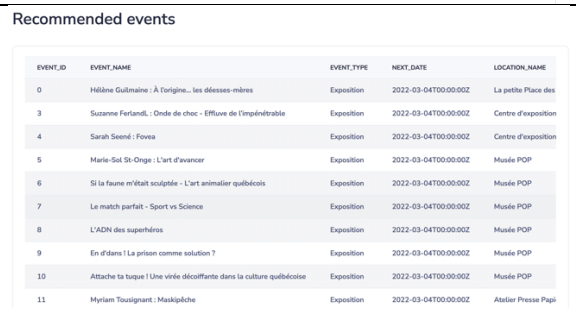
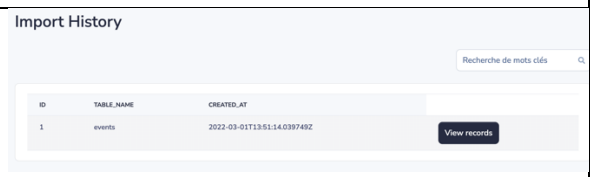
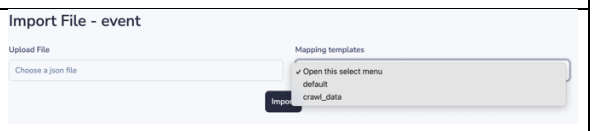
After creating a new url path for that new page, access to the file trivi-react-src-HomePage.js in order to map new url path with new component just created in the folder trivi-react/src/pages.

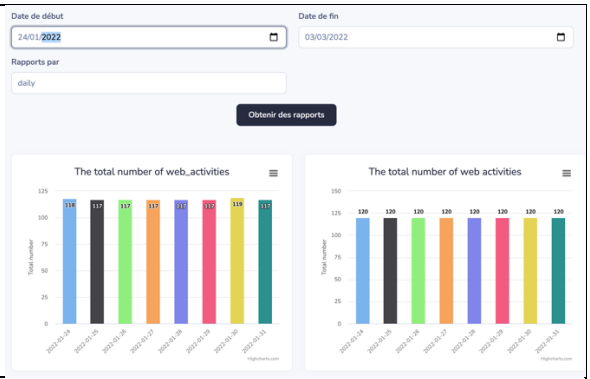
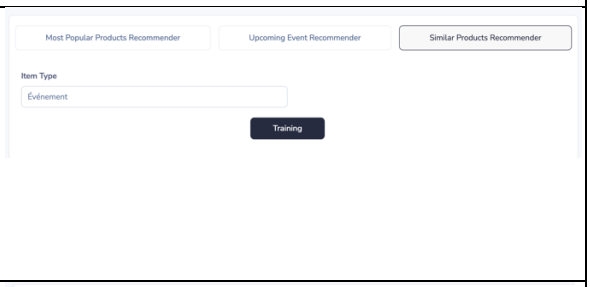


5. Main modules in views.py:
   This part shows important modules in the file views.py, which handle user requests in data management (import data, update data, mapping data, import json file, etc.) and recommendation (train items, retrieve list of recommended items, etc.)

Modules	Description	Example
Home	This module retrieves data about web activities, events and products for making some general reports in form of charts	
Import_json_file	This module handles user request for importing data (event, product, web activity) in a json file, then mapping data structure	
Import_api	This module handles user request for import data (event, product, web activity) which could be collected by an available API provided by data providers (Caligram)	
ItemList (get)	This module retrieves list of rows in a specified table (event, product, web activity)	
ItemDetail (get)	This module retrieves information of a specified row in a specified table	

	(event, product, web activity)	
ItemDetail (put)	This module handles user requests for updating information of a specified row in a specified table (event, product, web activity)	
ItemDetail (delete)	This module handles user requests for deleting information of a specified row in a specified table (event, product, web activity)	
ItemDetail (post)	This module handles user requests for creating a new row in a specified table (event, product, web activity)	
Get_recommend_api	This module handles for creating a recommend api and embedded link (which is then inserted in a website for retrieving recommended items)	
Get_recommend_info	This module handles for retrieving list of recommend types, list of recommend levels, etc. so that users could configure to	

	retrieve suitable recommendations.	
Get_configure_info	This module responses all configure information related to recommendation (web activity weight, training info, etc.)	
Get_list_recommend	This module handles for retrieving list of recommended items based on config parameters (recommend level, type, etc.) in user requests. In this module, there are many sub-modules, each sub-modules handle for a recommendation type/algorithms.	
Get_import_info	This module handles for retrieving import data history records .	
Get_mapping_templates	This module handles for retrieving list of mapping templates for a specified table.	

Get_reports	This module handles for making reports, charts for analysis.	
Delete_imported_items	This module handles for deleting all imported items that existed in imported json file.	
Train_similar_recommend	This module handles for running machine learning model for finding similarity values between 2 events/products.	
Update_activity_weight	This module handles for updating web_activity_weight (which is used for calculating popular value of an event/product)