
Implementation Document

for

surakshIIT

Group Name: GITHUB SPAMMERS

Course: CS253

Mentor TA: Mr. Aman Aryan

Date: 20-03-2022

Name	Roll No.	Email ID
Aarchie	200004	aarchi20@iitk.ac.in
Arpit Kumar	200189	arpitk20@iitk.ac.in
Harsh Jain	200412	harshj20@iitk.ac.in
Kajal Deep	200483	kajald20@iitk.ac.in
Kuldeep Singh Chouhan	200530	kuldeepsc20@iitk.ac.in
Mannem Rishwan Reddy	200562	mannemr20@iitk.ac.in
Rohit	200813	rohitg20@iitk.ac.in
Sandeep Kumar Bijarnia	200856	sandeepb20@iitk.ac.in
Udit Prasad	201055	uditp20@iitk.ac.in
Yash Raj Mittal	201148	yashrajm20@iitk.ac.in

Contents

CONTENTS	I
REVISIONS	II
1 IMPLEMENTATION DETAILS	1
2 CODEBASE	20
3 COMPLETENESS	25
APPENDIX A - GROUP LOG	27

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
---------	-------------------	------------------------	----------------

0.1	Mannem Rishwan Reddy, Kuldeep Singh Chouhan, Udit Prasad, Sandeep Kumar Bijarnia, Yash Raj Mittal, Aarchie, Kajal Deep, Arpit Kumar, Rohit, Harsh Jain	First Draft Completed	20/03/22
-----	--	-----------------------	----------

1 Implementation Details

USER SIDE:

We have used different libraries and applications to implement the Frontend.

- **Language** - We have chosen **JS** to develop our project as it is most popular and also it provides a variety of Libraries which makes our life easy.
- **Library**- We have made use of **React** JS. There are many reasons for going with React over other JS frameworks like :
 - i) It is very easy to learn and get used to.
 - ii) It is highly flexible compared to other JS frameworks
 - iii) It provides various features like reusable components and virtual DOM which makes both the code well structured and highly effective.
 - iv) It provided us with a great routing tool **react-router** which helps in managing multi-pages website like this one.
- Utilities -
 - i) **Axios** -
 - It is the client which we used for connecting our Backend with the Frontend.
 - It is very easy to use and supports all the types of HTTP methods like Get, Post, Put, etc
 - It is better than other similar services like Fetch because it allows cancelling of requests and also setting timeout for the requests.
 - ii) **Redux** -
 - We are using Redux mainly for session creation as it is one of the best state containers on Frontend side.
 - We used this state container mainly due to its simple and effective usage.
 - It also connects all the components and we can use state data in any of the components.
 - Also it creates session easily and we can manage it as per our convenience.
- Template-

For reducing our workload of CSS and focusing on implementing functional software, we used a template namely Argon Dashboard which provided us with a great CSS and a well structured template to get started with.

We decided to go with this because

- It is implemented in React JS

- It has rich bootstrap libraries to work with.

SERVER SIDE:

Language used : We decided to go with Python because :

- It has a smooth learning curve and all our project team members found it best to get started with.
- It is easily readable.
- It is object oriented programming language which makes it easy to manage the code
- It has many libraries and frameworks which makes our life easy.

Framework: Backend uses **Django**, a **Python** based web framework.

We chose Django over other similar frameworks because of:

- Reliability
- supporting faster development needs
- supporting large networking
- ease of creating APIs which can be transferred as Json data using serializers
- It has ready-made components to make clean and systematic architecture.
- It is easy to learn and implementation requires less time.
- It supports various databases
- It is easy to manage the whole project as it is very clearly divided into subparts like Views, Urls, Models and utils.
- It also provides us a temporary interface to keep check on our backend working.

Database: We have used **SQLite** as our database because of following reasons:

- We were using Django and it is the default database with Django

- Also, it is very easy to manage and can be used easily online other databases.
- It is easy to make queries and debug if any error happens

API Endpoints

Register

URL: /register

Method : POST

Data : {

```
    name ,  
    uid,  
    email ,  
    phone,  
    room_no ,  
    address,  
    gender,  
    password,  
    re_password,  
    dp
```

}

Status : {

If successfull : 200_OK

Else : 400_Bad Request(Already Registered / some required data is missing)

}

Login

```
URL: /login

Method : POST

Data : {
    uid,
    password
}

Status : {
    If successfull : 200 OK
    Else : {
        400_Bad Request(Already Loggedin)
        401_Unauthorized(Wrong Password/ Not registered)
    }
}
```

Logout

```
URL: /logout

Method : POST

Data : {
    uid
}

Status : {
    If successfull : 200_OK
    Else :
        {
            401_Unauthorized(Not logged in)
        }
}
```

Profile

URL: /profile

Method : GET

Response : USER DATA

```
Status : {  
    If successfull : 200_OK  
    Else :  
        {  
            404_NOT_Found  
            400_Bad_Request  
        }  
}
```


Report Lost Query

```
URL: /lost_found/add_lost  
      /security/add_lost
```

```
Method : POST
```

```
Data : {  
    name ,  
    details,  
    last_seen ,  
    image,  
    lost_time,  
    item_color ,  
    owner  
}
```

```
Status : {  
    If successfull : 201_Created  
    Else :  
        {  
            401_Unauthorized(Not logged in)  
            400_BAd Request(Missing required data or other error)  
        }  
}
```

View Lost Items

URL: /security/lost_item

Method : GET

Response : Lost items Data

```
Status : {  
    If successfull : 200_OK  
    Else :  
        {  
            401_Unauthorized(If it is not security)  
        }  
}
```

View Found Items

URL: /security/found_item

Method : GET

Response : Found items Data

```
Status : {  
    If successfull : 200_OK  
    Else :  
        {  
            401_Unauthorized(If it is not security)  
        }  
}
```

Report Found query

URL: /security/add_found

Method : POST

Data : {

```
    name ,  
    details,  
    item_color,  
    where_found,  
    if_returned,  
    who_found,  
    found_time,  
    image
```

}

Status : {

```
    If successfull : 201_Created
```

```
    Else :
```

```
        {
```

```
            401_Unauthorized(Not logged in)
```

```
            400_BAd Request(Missing required data or other error)
```

```
        }
```

```
    }
```

Delete Lost Query

URL: /lost_found/delete_lost
/security/delete_lost

Method : DELETE

Response : Lost item Deleted

```
Status : {  
    If successfull : 200_OK  
    Else :  
        {  
            401_Unauthorized(Not logged in)  
            400_BAd Request(Lost Item not present in Database)  
        }  
}
```

Delete Found Query

URL: /security/delete_found

Method : DELETE

Reponse : Found item deleted

```
Status : {  
    If successfull : 200_OK  
    Else :  
        {  
            401_Unauthorized(Not logged in)  
            400_BAd Request(Found Item not present in Database)  
        }  
}
```

Add Hall entry

URL: /security/add_hallentry

Method : POST

```
Data : {  
    person1,  
    person2,  
    destination,  
    entry_time,  
    hall,  
    if_exited = False  
}
```

```
Status : {  
    If successfull : 201_Created  
    Else :  
        {  
            401_Unauthorized(Not logged in)  
            400_BAd Request(Missing required data or other error)  
        }  
}
```

View Hall entries

URL: /security/hall_entries

Method : GET

Response : Hall Entries Data (For which if_exited is false)

```
Status : {  
    If successfull : 200_ok  
    Else :  
        {  
            401_Unauthorized(Not logged in)  
        }  
}
```

View all Hall entries

URL: /security/all_hall_entries

Method : GET

Response : All Hall Entries Data

```
Status : {  
    If successfull : 200_ok  
    Else :  
        {  
            401_Unauthorized(Not logged in)  
        }  
}
```

Add Hall exit

URL: /security/add_hallexit

Method : PUT

Data : {
 exit_time,
 if_exited = True
}

Status : {
 If successfull : 200_OK
 Else :
 {
 401_Unauthorized(Not logged in)
 400_BAd Request(Missing required data or other error)
 }
}

Add Campus entry

URL: /security/add_campusentry

Method : PUT

Data : {
 entry_time,
 if_entered = True
}

Status : {
 If successfull : 200_OK
 Else :
 {
 401_Unauthorized(Not logged in)
 400_BAd Request(Missing required data or other error)
 }
}

View Campus exit

URL: /security/campus_exits

Method : GET

Response: Campus Exits (for which if_entered = false)

```
Status : {  
    If successfull : 200_OK  
    Else :  
        {  
            404_NOT_FOUND  
        }  
}
```

View all Campus exits

URL: /security/all_campus_exits

Method : GET

Response : All Campus Exits Data

```
Status : {  
    If successfull : 200_ok  
    Else :  
        {  
            404_NOT_FOUND  
        }  
}
```

Add Campus exit

URL: /security/add_campusexit

Method : POST

Data : {

```
    person,  
    if_entered = False,  
    destination,  
    exit_time
```

}

Status : {

```
    If successfull : 201_CREATED
```

```
    Else :
```

```
        {
```

```
            401_Unauthorized(Not logged in)
```

```
            400_BAd Request(Missing required data or other error)
```

```
        }
```

}

Add Non resident Campus entry

URL: /security/add_nonresident_campusentry

Method : POST

Data : {
 name,
 vehicle,
 vehicle_number,
 concerned_person,
 reason,
 entry_time,
 if_exited = false,
 id_document,
 id_number
}

Status : {
 If successfull : 201_CREATED
 Else :
 {
 401_Unauthorized(Not logged in)
 400_BAd Request(Missing required data or other error)
 }
}

Add Non resident Campus exit

URL: /security/add_nonresident_campusexit

Method : PUT

```
Data : {  
    exit_time,  
    if_exited = true  
}
```

```
Status : {  
    If successfull : 200_OK  
    Else :  
        {  
            401_Unauthorized(Not logged in)  
            400_BAd Request(Missing entry)  
        }  
}
```

See Non resident Campus entries

```
URL: /security/nonresident_campuseries

Method : GET

Response : Non Resident Campus Entries ( For which if_exited = false)

Status : {
    If successfull : 200_OK
    Else :
        {
            404_NOT_FOUND
        }
}
```

See All Non-resident Campus entries

```
URL: /security/all_nonresident_campuseries

Method : GET

Response : Non Resident Campus Entries

Status : {
    If successfull : 200_OK
    Else :
        {
            404_NOT_FOUND
        }
}
```

Setting up the Backend : Django is installed and then five new apps are started by running the command [python3 manage.py {appname}]. These apps include campus_movement,

hall_movement, lost_found, security and user. Then these apps are registered in the [settings.py] file. Models are created in each app according to requirement which defines how the application's items are stored in the database. Makemigrations and migrate commands are then run to make the migration file and apply the changes.

The database used is the inbuilt SQLite which is the default database and the easiest one to use as it is included in Python and does not need installation of any kind.

Setting up the APIs : APIs are created using the Django REST framework. To create APIs serializers are created [serializers.py] which convert model instances to JSON so that the frontend can work with the received data. Then the views are created in the [views.py] file and URL path for the APIs are specified in the [urls.py] file. This almost completes the backend application.

2 Codebase

Link to GitHub repository – <https://github.com/aarchie-r/surakshIIT>













Steps to run the code-

- Clone the repository

```
$ git clone https://github.com/aarchie-r/surakshIIT.git
```

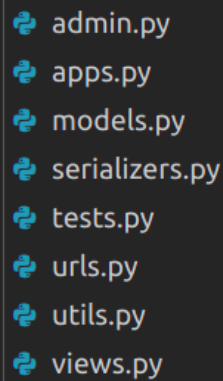
1. Backend -

It contains these folders and files

	campus_movement
	hall_movement
	lost_found
	media
	security
	surakshIIT_backend
	templates
	user
	.gitignore
	db.sqlite3
	ez_setup.py
	manage.py

- **Campus movement, hall_movement, lost_found, security and user** are 5 apps
- **surakshIIT_backend** is the main app which contains settings and main urls.

Each of the above mentioned apps mainly contains these files :



```
admin.py
apps.py
models.py
serializers.py
tests.py
urls.py
utils.py
views.py
```

Here :

- urls.py contains all the url endpoints of that app
- models.py contains the model of that app which acts as the data point.
- serializers.py contains all the serializers
- views.py contains the views.
- utils.py contains the functions required for that app

How to run Backend

- Make sure you have django installed.
- Move to the Backend directory in the terminal.
- Give the following commands to start the backend server.

```
$ python3 manage.py makemigrations
```

makemigrations is responsible for creating new migrations based on the changes we have made in our models. We need to run this command in the terminal after making any change in any of the models file.

```
$ python3 manage.py migrate
```

migrate is responsible for applying changes made by makemigrations to our database schema.

```
$ python3 manage.py runserver
```

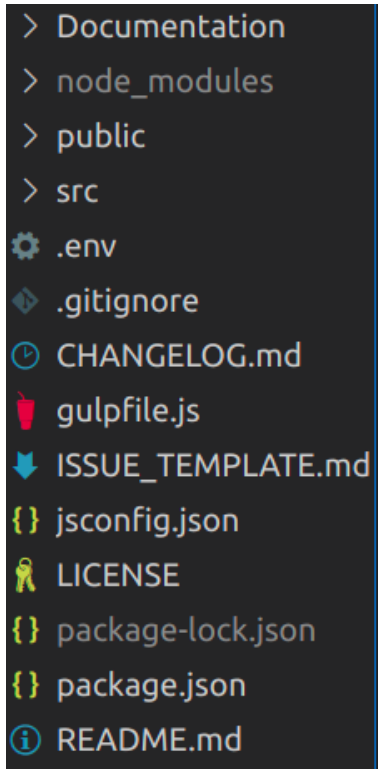
createsuperuser - Django provides us Admin Panel for its users. So we need not worry about creating a separate Admin page or providing authentication features as Django provides us with that feature. Before using this feature, we need to migrate our project, otherwise the superuser database will not be created.

This can be created using following command -

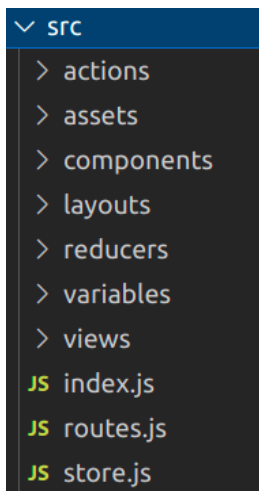
```
$ python3 manage.py createsuperuser
```


2) Frontend

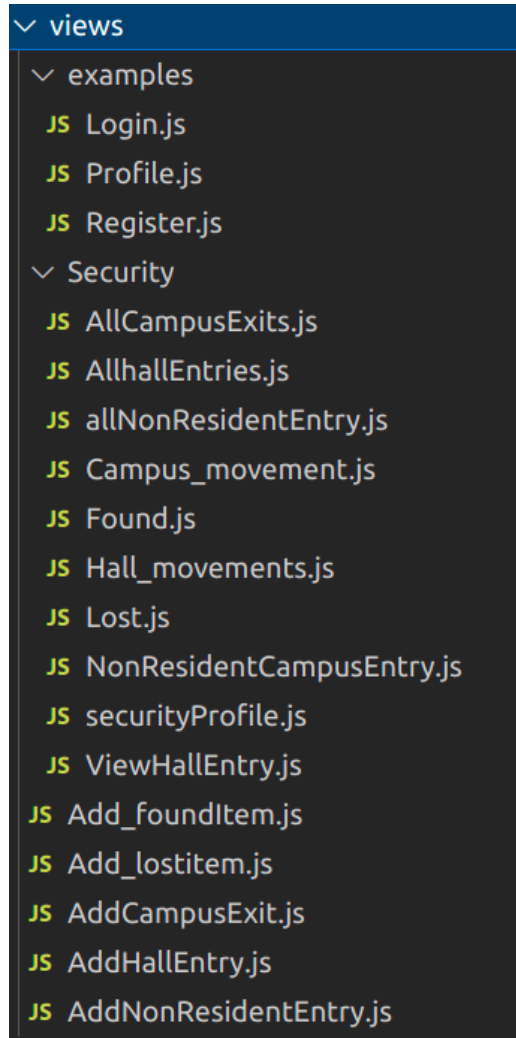
Frontend contains these folders and files



- All the packages which we have used are stored in **package.json**
- All the components are in **src** folder
- **public** folder contains the **index.html** which is landing page which will make use of src folder
- These are the folders and files of src folder:



- **index.js** is landing page which will be used by React.
- **routes.js** contain the urls and which component will be used by which url.
- **store.js** is the file by which Redux will work.
- **assets** folder contains the images,css and other assets
- **components** folder contains the header,footer,sidebar and navbar.
- **actions** contain the userAction file which is used for session creation.
- **layouts** contains files which will make use of **routes.js** to manage the routing and role based access i.e. **security and user** can see only their concerned user side.
- **reducers** contains the redux-reducers file which will also be used for session making and login and logout functions
- **views** contains all the ui files



How to run Frontend

- Make sure you have node and npm installed.
- Move to frontend directory and give following commands -

```
$ npm install
```

- The **npm install** installs all modules that are listed on **package.json** file and their dependencies. **npm update** updates all packages in the node_modules directory and their dependencies.

```
$ npm start
```

- **npm start** script is used to execute the defined file in it without typing its execution command.

3 Completeness

Provide the details of the part of the SRS that have been completed in the implementation.

Provide the future development plan by listing down the features that will be added in the (may be hypothetical) future versions.

All the features mentioned in the first version of SRS Documentation have been Developed and fully Functional.

The Software requirement specification document specifies three main type of functions and two type of users:

Users : Residents and Security

Residents Accessibility : According to the SRS document, any registered Resident can login and will have access to **add lost items** from their profile which will be updated in the database maintained at Backend.

Security Accessibility: According to the SRS document, any registered Security official can access to the listed functions-

- **Add/Delete Lost Item**: updating the database by adding lost items if reported to them or deleting lost items when the item has been found & returned. Adding the item will require details like found location, time and item specifications.
- **Add/Delete Found item**: updating the database by adding found items if reported to them or they themselves find them or deleting lost items when the item has been returned. Adding the item will require details like found location, time and item specifications.
- **Keeping record of campus entry/exits**: updating the database by adding the entries of the person checking in/out by putting their uid if person is resident or just filling their details.
- **Keeping record of Hall entry/exits**: updating the database by adding the entries of the person checking in/out by putting their uid, destination, uid of other person if came to meet.

The features that can be added in **future versions** of our surakshIIT software are -

- OTP generation at the time of registering new user for checking authenticity
- Ability of recreating password after verification in case user forgets password
- Notification generation whenever any non-resident comes to meet any registered resident to strengthen security.
- A category-wise filter option for Lost and Found portal (like color, date wise, location wise etc) and entry/exits portals' (like hall wise, gender wise, date or time wise etc)
- A user-wise filter option which shows the logs of a particular user which have been made across all parts of the application
- Option for updating Profile and all the related info of the user from the portal itself.
- Enhancing the UI/UX for the Web application.
- A campus-wide Notice Board for delivering important information.

Appendix A - Group Log

Time period	Work done
18th Feb to 20th Feb	Learning the required prerequisites of using Django framework, React, npm, various related libraries and languages like python, Javascript etc Also, planning the implementation with the desired final functional software
21st Feb to 26th	-----Midsem (No work)-----
27th Feb to 1st March	Initial commits for backend as well as frontend
2nd March to 4th March	Created initial models for Lost & Found app, Hall movements app, Campus Entry/Exit app and User (security + non security) app Cleaned Frontend initial layout as per requirements
5th March to 8th March	Created models for Registering new user in Backend Updated views for Lost & Found app and Hall movements app in Backend allowing it to have HTTP methods like Get, Post, Put, Delete etc as per requirement
9th March to 13th March	Updated Models for the different apps Updated views for Campus Entry/Exit app and User app in Backend allowing it to have HTTP methods like Get, Post, Put, Delete etc along with taking care of Login/Logout of the user
14th March to 19th March	Prefinalized Backend and updated the required changes keeping check on bugs and issues Created User Interface components like Header, Footer, Sidebar etc along with the initial layout for different apps in Frontend along with integration to Backend
20th March to 21st March	Finalized overall software backend as well as frontend also modified to achieve better results Checked on software functionality, performance, reliability, accessibility, efficiency, correctness, usability, integrity Updated Implementation Document