**Shubhangi Agarwal**

**Self-Work Report**

**Week 1**

**Status**

Implemented work

* Basic database model for taking user input (Through Django, sqlite3)
* Handling HTTP requests and responses (Through Django)
* Addition of user input into training data (csv file) (Through Django)
* A REST API for connecting with frontend (Through Django)
* Creation of serialiser to serialise and de-serialise the incoming and outgoing data of any format (Through Django)
* API mapping to create a request from React to Django (Through React.js)

In Progress

* Connection of frontend with backend for getting an end-to-end workflow

**Backend Details**

Final Database Schema (As discussed with all)

* Partner

id : AutoField, primary key

firstName : CharField

lastName : CharField

email : EmailField, unique

contact : CharField

address : CharField

city : CharField

state : CharField

typeStore : CharField

size : CharField

workingEmployees : PositiveIntegerField

customers : PositiveIntegerField

service : CharField

password : CharField

Initial Database Schema (self-thought)

* Partner

id

firstName

lastName

email

contact

age

password

location of the store (city,state,pincode)

number of employees working in the store

type of store

size of store

google maps rating

annual sales of the store

additional info.

website link of the store (if any)

storage size willing to give to the locker

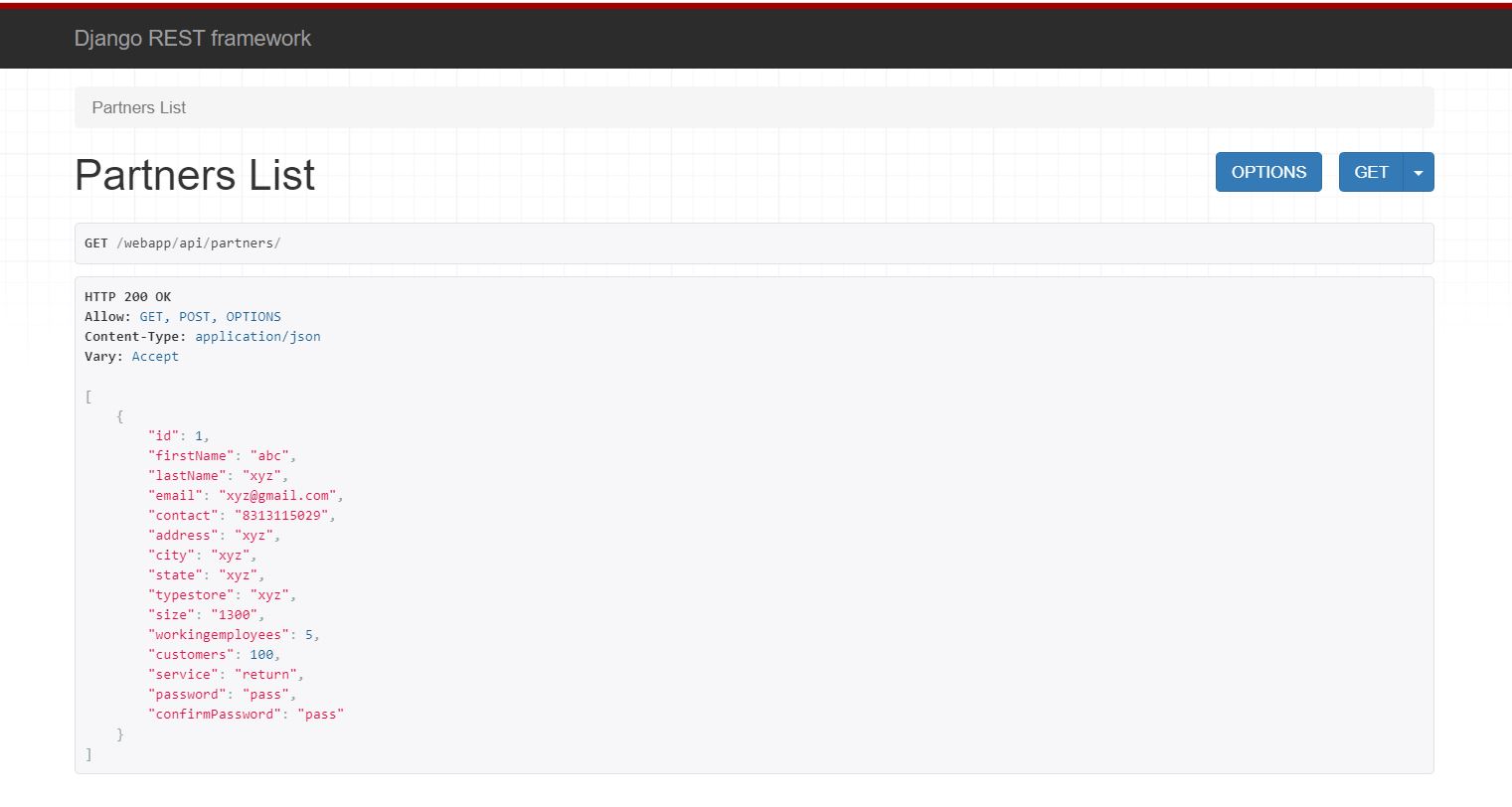
Routes

* webapp/api/partners/list/ : { method : ‘post’ , ‘get’ }
* webapp/api/partners/list/details/ : { method : ‘put’ , ‘delete’ }

Requests and responses

**{ ‘request\_method’ ? response\_if\_true : response\_if\_false }**

* ‘get’ ? displays data stored in ‘Partner’ database : HTTP\_400\_BAD\_REQUEST
* ‘post’ ? HTTP\_201\_CREATED : HTTP\_400\_BAD\_REQUEST
* ‘put’ ? HTTP\_204\_NO\_CONTENT : HTTP\_400\_BAD\_REQUEST
* ‘delete’ ? HTTP\_204\_NO\_CONTENT : HTTP\_400\_BAD\_REQUEST



**Fig. 1:** Response for ‘get’ request

**Technical Challenges Faced**

* Understanding MVC and MVT architecture : Solved
* Making migrations to the model through Django : Solved
* Sending requests from React app to Django : Solved

**Week 2**

**Status**

Implemented work

* Connection of Frontend with Backend (Client-Server Model)
* Made required changes in the attributes of the database model
* Read about various classification algorithms
* Read about google maps API
* Discussed upon and finalized some key parameters to be used for prediction
* Established and suggested a relation between the parameters for prediction of correct program so as to increase the accuracy and discussed about it with the group
* Read about deploying pre-trained ML models using Django

In Progress

* Identification of more key parameters and algorithms to be used for prediction of program

**References**

* <https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052>
* <https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/>
* <https://towardsdatascience.com/productionize-a-machine-learning-model-with-a-django-api-c774cb47698c>
* <https://developers.google.com/maps/documentation/api-picker>
* <https://www.geeksforgeeks.org/understanding-logistic-regression/>

**Technical Challenges Faced**

None

**Week 3**

**Status**

Implemented work

* Integration of login, sign-up flow for the user and created different REST APIs for each respectively at backend
* Refactoring of the database model so that a user can onboard multiple stores, to decrease data concurrency and for making the project more scalable and made the required alterations
* Established a many-to-one relationship amongst the database model

In Progress

* Authentication of the user from the backend

**Refactored Database Schema**

Store

Customer id : AutoField

firstName : CharField n owner : ForeignKey

lastName : CharField storeName : CharField

email : EmailField, Primary Key 1 address : CharField

contact : CharField city : CharField

password : CharField state : CharField

pincode : CharField

typeStore : CharField

size : CharField

employees : IntegerField

customers : IntegerField

service : CharField

**Updated Routes**

* webapp/api/login/ : { method : ‘post’ }
* webapp/api/signup/ : { method : ‘post’ }
* webapp/api/dashboard/add-store/ : { method : ‘post’ }
* webapp/api/dashboard/delete-store/ : { method : ‘put’ , ‘delete’ }

**Technical Challenges Faced**

None

**Week 4**

**Status**

Implemented work

* Read about JWT Authentication
* Read about QuerySet and making queries through Django
* Validation and authentication of the Customer using token authentication for login page at the server side with the required requests handling and sending responses
* Testing of the Customer auth implemented at the server side

In Progress

* Sync-up and testing of all the components for the end-to-end flow

**References**

* <https://medium.com/python-pandemonium/json-web-token-based-authentication-in-django-b6dcfa42a332>
* <https://docs.djangoproject.com/en/3.0/topics/db/queries/>
* <https://docs.djangoproject.com/en/3.0/ref/models/querysets/>

**Technical Challenges Faced**

* Small errors faced while making query to validate the customer’s email and password : Solved
* Object of Byte type is not JSON serializable : Solved

**Week 5**

**Status**

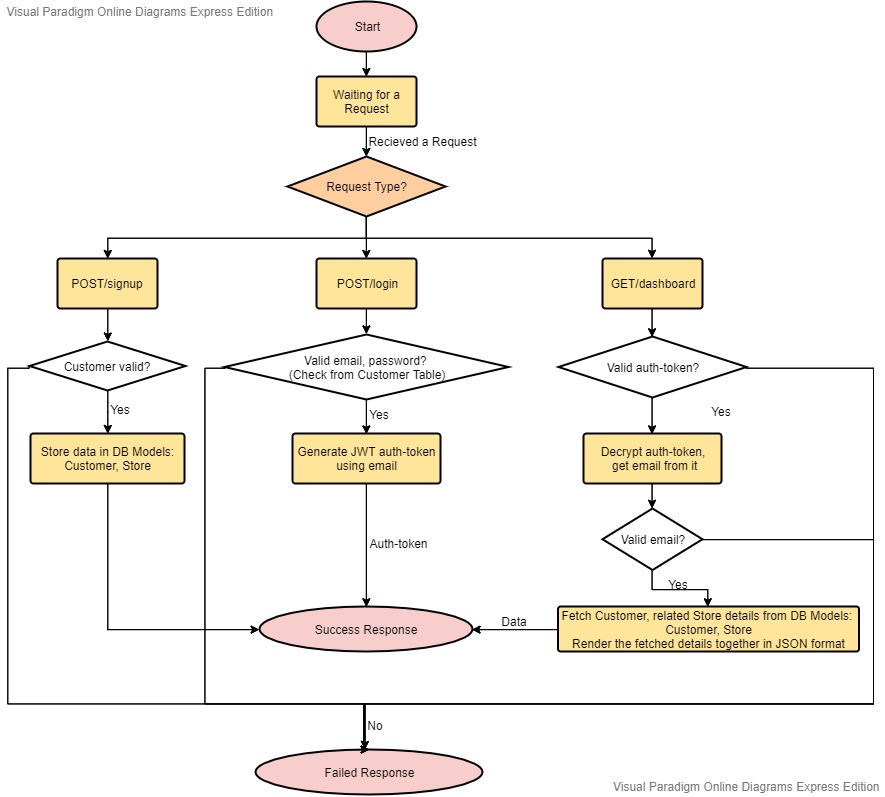
Implemented work

* Fixed a bug of taking store details from signup request and storing it into the database
* Created new routes at the server side for fetching details of the customer and his/her store after getting a request from the dashboard
* Implemented validation of the customer at the server side for dashboard page through the encrypted token received with the request and sent the response to the client side.
* Updated the above created routes at the client side
* Synced the variable names at the server side with the client side
* Designed a present, remaining server flow diagram along with Anisha Jain
* Testing of end-to-end flow of the application built till now
* Tried deploying ML model at the server side

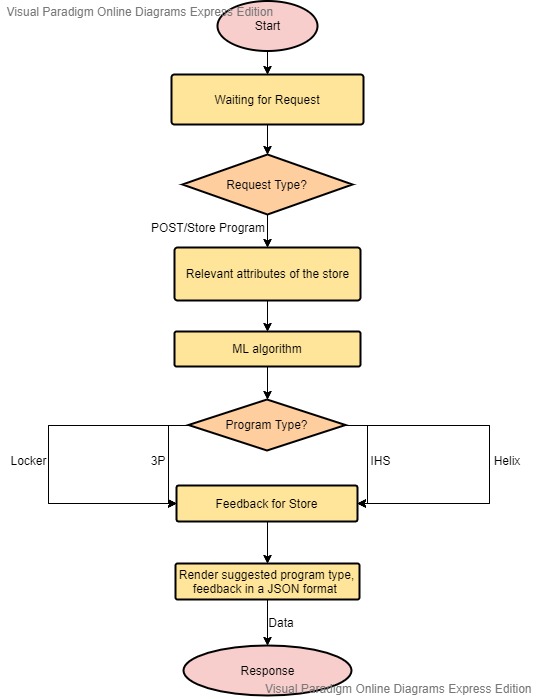
Remaining Work

* Integration of ML model at the server side
* Prediction of program for the store, generating a feedback for the store and sending these along with the response from server side to client side

**Present flow diagram**

****

**Remaining flow diagram**



**Routes Added**

* webapp/api/dashboard/customer-details/ : { method : ‘get’ }
* webapp/api/dashboard/store-details/ : { method : ‘get’ }

**Technical Challenges Faced**

* Resolving the bug of taking store details from signup request and storing it into the database : Solved
* ML model using Random Forest algorithm trained in 64-bit Python version and Server side is working on 32-bit version : Solved  
  First solution: Training the model again on 32-bit version of Python so that the training and testing system remains the same because Random Forest algorithm uses different types of indices on 32-bit and 64-bit machines.

Another solution: Shifting of the server to a 64-bit version of Python. (Used this)

**Week 6**

**Status**

Implemented work

* Fixed minor bugs for security issues at the server side
* Connection of ML model with server side for prediction of program
* Stored the predicted program in the database along with other store details
* Testing of final flow of the application
* Minor changes in client side at *Project/frontend/src/component/Dashboard/Store.js* to display the predicted program (earlier it was hardcoded and displayed locker for every store)
* Suggested changes according to the implemented flow in the application flow diagram designed by Ayushi Gupta

In Progress

* Taking suggestions by the Mentor [Full flow completed]

**Technical Challenges Faced**

* None