**IBM Full Stack Developer Final Project Notes:-**

**Car Dealership Project**

**Project breakdown**

**Fork the GitHub repo containing the project template. The main web application is a predefined Django application. You will need to add some new features, and then build and run your project implementation.**

1. Fork the repository in your account.
2. Clone the repository in the Cloud IDE environment.
3. Create static pages to finish the user stories.
4. Run the application locally.

**Add user management to the Django application.**

1. Implement user management using the Django user authentication system and create a REACT frontend.

**Implement backend services.**

1. Create Node.js server to manage dealers and reviews using MongoDB and dockerize it.
2. Deploy sentiment analyzer on Code Engine.
3. Create Django models and views to manage car model and car make.
4. Create Django proxy services and views to integrate dealers and reviews together.

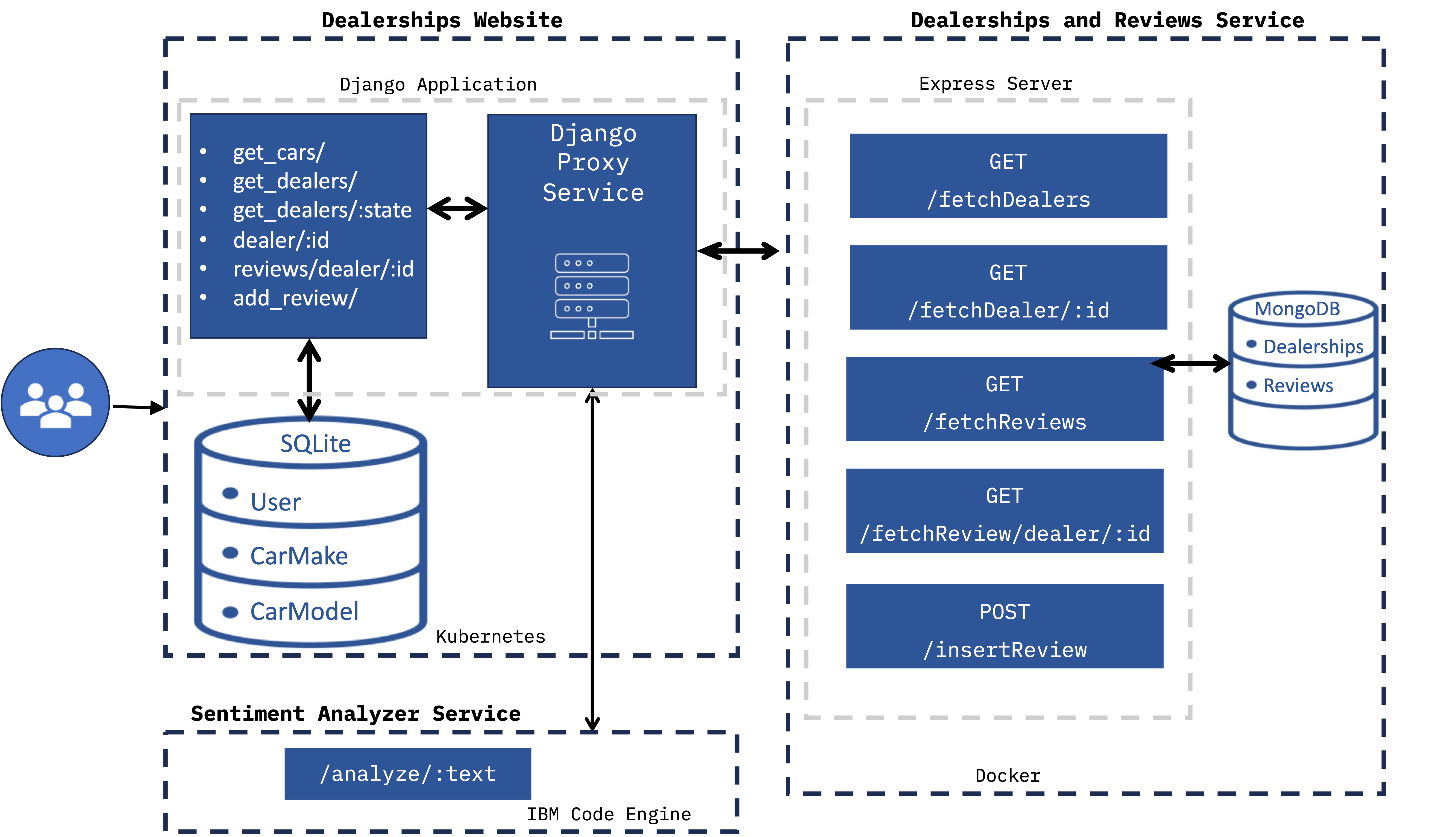
**Add dynamic pages with Django templates.**

1. Create a page that displays all the dealers.
2. Create a page that displays reviews for a selected dealer.
3. Create a page that lets the end user add a review for a selected dealer.

**Implement CI/CD, and then run and test your application**

1. Set up continuous integration and delivery for code linting.
2. Run your application on Cloud IDE.
3. Test the updated application locally.
4. Deploy the application on Kubernetes.

**Solution architecture**



The solution will consist of multiple technologies

1. The user interacts with the "Dealerships Website", a Django website, through a web browser.
2. The Django application provides the following microservices for the end user:

* **get\_cars/** - To get the list of cars from
* **get\_dealers/** - To get the list of dealers
* **get\_dealers/:state** - To get dealers by state
* **dealer/:id** - To get dealer by id
* **review/dealer/:id** - To get reviews specific to a dealer
* **add\_review/** - To post review about a dealer

1. The Django application uses SQLite database to store the Car Make and the Car Model data.
2. The "Dealerships and Reviews Service" is an Express Mongo service running in a Docker container. It provides the following services:

* **/fetchDealers** - To fetch the dealers
* **/fetchDealer/:id** - To fetch the dealer by id
* **fetchReviews** - To fetch all the reviews
* **fetchReview/dealer/:id** - To fetch reviews for a dealer by id
* **/insertReview** - To insert a review

1. "Dealerships Website" interacts with the "Dealership and Reviews Service" through the "Django Proxy Service" contained within the Django Application.
2. The "Sentiment Analyzer Service" is deployed on IBM Cloud Code Engine, it provides the following service:

* **/analyze/:text** - To analyze the sentiment of the text passed. It returns *positive, negative or neutral*.

1. The "Dealerships Website" consumes the "Sentiment Analyzer Service" to analyze the sentiments of the reviews through the Django Proxy contained within the Django application.

### Use cases for anonymous users:

1. View the **Contact Us** page.
2. View the **About Us** page.
3. View the list of dealerships.
4. Filter the list of dealerships by state:
   1. Select **Show all** or a specific state from the State dropdown on the dealership page.
   2. View all states if nothing is selected in the dropdown.
   3. View a table of dealerships for the selected state when the form is submitted.
5. Click on a dealership to view the reviews for that dealership on the details page with each review displayed on a bootstrap card.
6. Log in using their credentials.

### Use cases for authorized users:

In addition to the above, authorized users should be able to write a review for any dealership on the dealership's page. In order to enable authorized users to write their reviews:

1. A Review button should be provided against each dealer listed in the dealership table.
2. Clicking on the Review button should take the user to the review page.
3. Filling the form on the review page and submitting it should add the review.

{

"user\_id": 1, => from Django

"name": "Berkly Shepley", => from Django

"dealership": 15, => from the form

"review": "Total grid-enabled service-desk", => form textbox

"time": "", => current time

"purchase": true, => form checkbox

"purchase\_date": "07/11/2020", => form calendar (bootstrap)

"car\_make": "Audi", => from django dropdown

"car\_model": "A6", => from django dropdown

"car\_year": 2010 => form django dropdown

}

1. On submission, the user should be taken back to the dealership detail page with the submitted review featured at the top of the reviews list, sorted on time.

### Use cases for admin users:

1. Log in to the admin site with a predefined username and password.
2. Add new make, model, and other attributes.

Your organization has assigned you as the Lead Full-Stack Software Developer on this project. Your job is to develop this portal as part of your Capstone project by following best practices for Full-Stack software development.

**Node Js Inserts**

const express = require("express");

const mongoose = require("mongoose");

const fs = require("fs");

const cors = require("cors");

const app = express();

const port = 3030;

app.use(cors());

app.use(require("body-parser").urlencoded({ extended: false }));

const reviews\_data = JSON.parse(fs.readFileSync("reviews.json", "utf8"));

const dealerships\_data = JSON.parse(

  fs.readFileSync("dealerships.json", "utf8")

);

mongoose.connect("mongodb://mongo\_db:27017/", { dbName: "dealershipsDB" });

const Reviews = require("./review"); // Json saved along with their schemas

const Dealerships = require("./dealership"); // Json saved along with their schemas

try {

  Reviews.deleteMany({}).then(() => {

    Reviews.insertMany(reviews\_data.reviews);

  });

  Dealerships.deleteMany({}).then(() => {

    Dealerships.insertMany(dealerships\_data.dealerships);

  });

} catch (error) {

  res.status(500).json({ error: "Error fetching documents" });

}

// Express route to home

app.get("/", async (req, res) => {

  res.send("Welcome to the Mongoose API");

});

// Express route to fetch all reviews

app.get("/fetchReviews", async (req, res) => {

  try {

    const documents = await Reviews.find();

    res.json(documents);

  } catch (error) {

    res.status(500).json({ error: "Error fetching documents" });

  }

});

// Express route to fetch reviews by a particular dealer

app.get("/fetchReviews/dealer/:id", async (req, res) => {

  try {

    const documents = await Reviews.find({ dealership: req.params.id });

    res.json(documents);

  } catch (error) {

    res.status(500).json({ error: "Error fetching documents" });

  }

});

// Express route to fetch all dealerships

app.get("/fetchDealers", async (req, res) => {

  try {

    const documents = await Dealerships.find();

    res.json(documents);

  } catch (error) {

    res.status(500).json({ error: "Error fetching dealerships" });

  }

});

app.get(“/fetchDealers”, async(req, res) => {

try {

const documents = await Dealership.find();

res.json(documents);

} catch {

res.status(500).json({error: “Error fetching dealership”})

}

})

// Express route to fetch Dealers by a particular state

app.get("/fetchDealers/:state", async (req, res) => {

  try {

    const documents = await Dealerships.find({ state: req.params.state });

    res.json(documents);

  } catch (error) {

    res.status(500).json({ error: "Error fetching dealerships" });

  }

});

// Express route to fetch dealer by a particular id

app.get("/fetchDealer/:id", async (req, res) => {

  try {

    const documents = await Dealerships.find({ id: req.params.id });

    res.json(documents);

  } catch (error) {

    res.status(500).json({ error: "Error fetching dealerships" });

  }

});

//Express route to insert review

app.post("/insert\_review", express.raw({ type: "\*/\*" }), async (req, res) => {

  data = JSON.parse(req.body);

  const documents = await Reviews.find().sort({ id: -1 });

  let new\_id = documents.id + 1;

  const review = new Reviews({

    id: new\_id,

    name: data.name,

    dealership: data.dealership,

    review: data.review,

    purchase: data.purchase,

    purchase\_date: data.purchase\_date,

    car\_make: data.car\_make,

    car\_model: data.car\_model,

    car\_year: data.car\_year,

  });

  try {

    const savedReview = await review.save();

    res.json(savedReview);

  } catch (error) {

    console.log(error);

    res.status(500).json({ error: "Error inserting review" });

  }

});

// Start the Express server

app.listen(port, () => {

  console.log(`Server is running on http://localhost:${port}`);

});

**Docker/ Executing different part of package:-**

Under the Database start Mongo server, docket build and get the url. Under the DjangoApp folder change the .env file make it point to the link you get for backend mongo. Go back to server and start/restart Django server.

A starter code of the client has been provided for you.

1. Open a New Terminal and switch to the client directory.
2. cd /home/project/xrwvm-fullstack\_developer\_capstone/server/frontend
3. Install all required packages.
4. npm install
5. Run the following command to build the client.
6. npm run build

Django and Sentiment Analysis Lab:- [Django\_Sentiment\_Analysis](https://labs.cognitiveclass.ai/v2/tools/cloud-ide-kubernetes?ulid=ulid-c44313a1d84bdaabef9d19bed137ee7d3a1e3b5f)

To start microservices:-

CD:- cd xrwvm-fullstack\_developer\_capstone/server/djangoapp/microservices

* docker build . -t us.icr.io/${SN\_ICR\_NAMESPACE}/senti\_analyzer
* docker push us.icr.io/${SN\_ICR\_NAMESPACE}/senti\_analyzer
* IBM Cloud Link:- ibmcloud ce application create --name sentianalyzer --image us.icr.io/${SN\_ICR\_NAMESPACE}/senti\_analyzer --registry-secret icr-secret --port 5000

The backend Mongo Express server needs to be up and running in one of the terminals in the lab environment. At this stage, the server code will have all the end points implemented already.

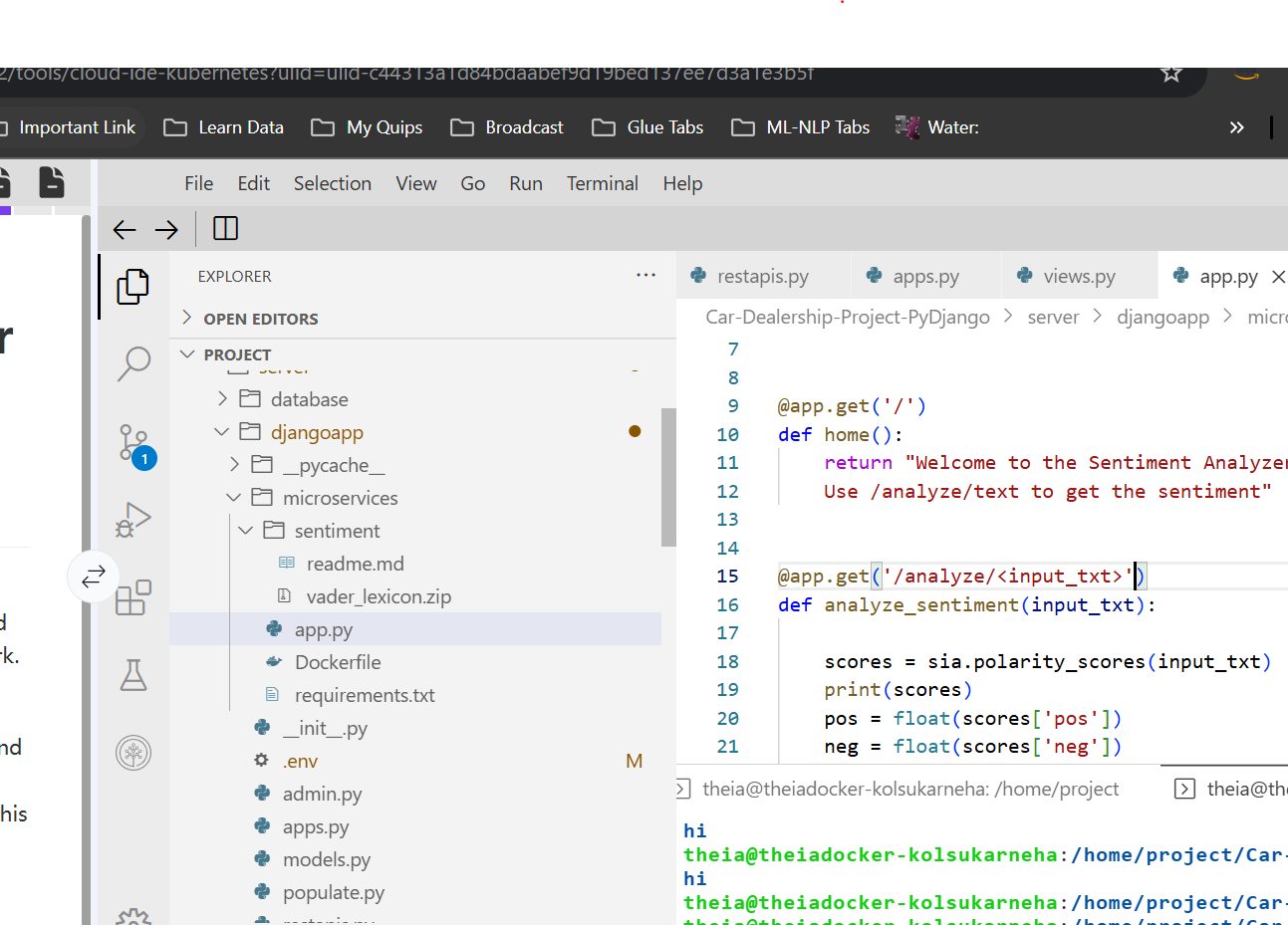
Link :- [Coursera\_Tutorial\_Link](https://author-ide.skills.network/render?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJtZF9pbnN0cnVjdGlvbnNfdXJsIjoiaHR0cHM6Ly9jZi1jb3Vyc2VzLWRhdGEuczMudXMuY2xvdWQtb2JqZWN0LXN0b3JhZ2UuYXBwZG9tYWluLmNsb3VkL0lCTVNraWxsc05ldHdvcmstQ0QwMzIxRU4tQ291cnNlcmEvbGFicy92Mi9tMy9CYWNrZW5kU2VydmljZXNfTW9uZ28ubWQiLCJ0b29sX3R5cGUiOiJ0aGVpYWRvY2tlciIsImF0bGFzX2ZpbGVfaWQiOjEyNzY4LCJhZG1pbiI6ZmFsc2UsImlhdCI6MTcyOTg5ODg4OH0.P9RKQddgyfBVepbL0FiHUYCyBOe22ejJIfpr3odolXk)

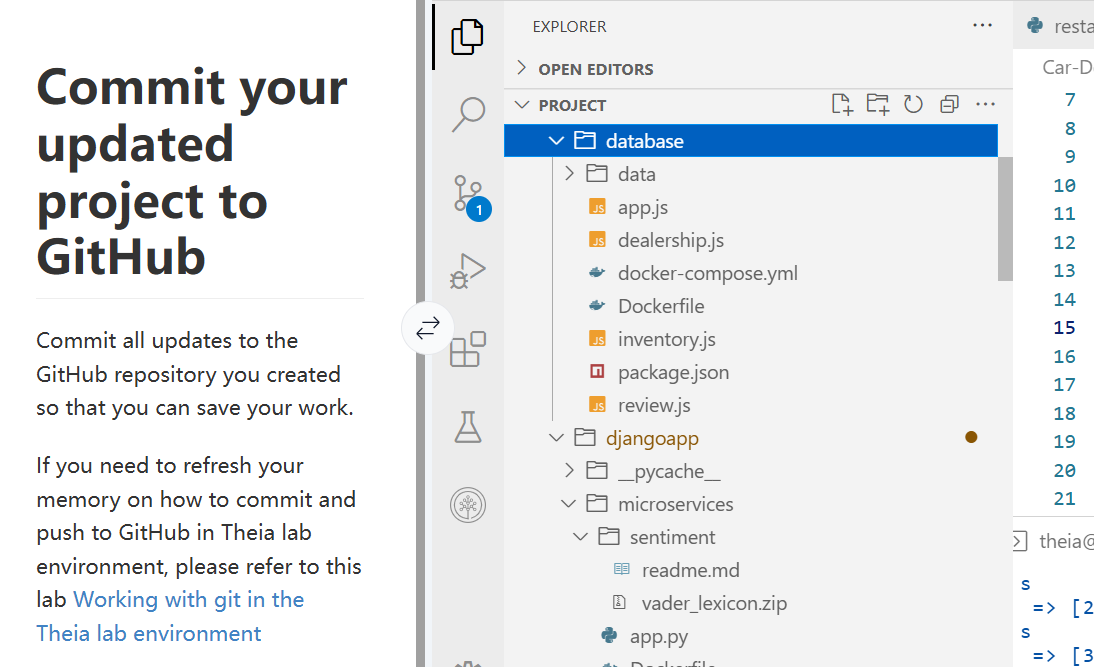
Docker Build Command -

* 1. **docker build . -t nodeapp**
  2. **docker-compose up**

**Django integration with MongoDb/Nodejs**

To integrate external dealers and review data(API’s created in node app to fetch the data from MongoDB), you will need to call the APIs from the Django app and process the API results in Django views(Views attached to the API’s/Django app url’s, Views using RestApi request to call the Node API) to be later rendered through REACT pages(Used in REACT components/get/post requests in HTML/React pages). Such Django views use proxy services(RestApi request to call the Node API) to fetch data from external resources as per users' requests and renders it using REACT components.





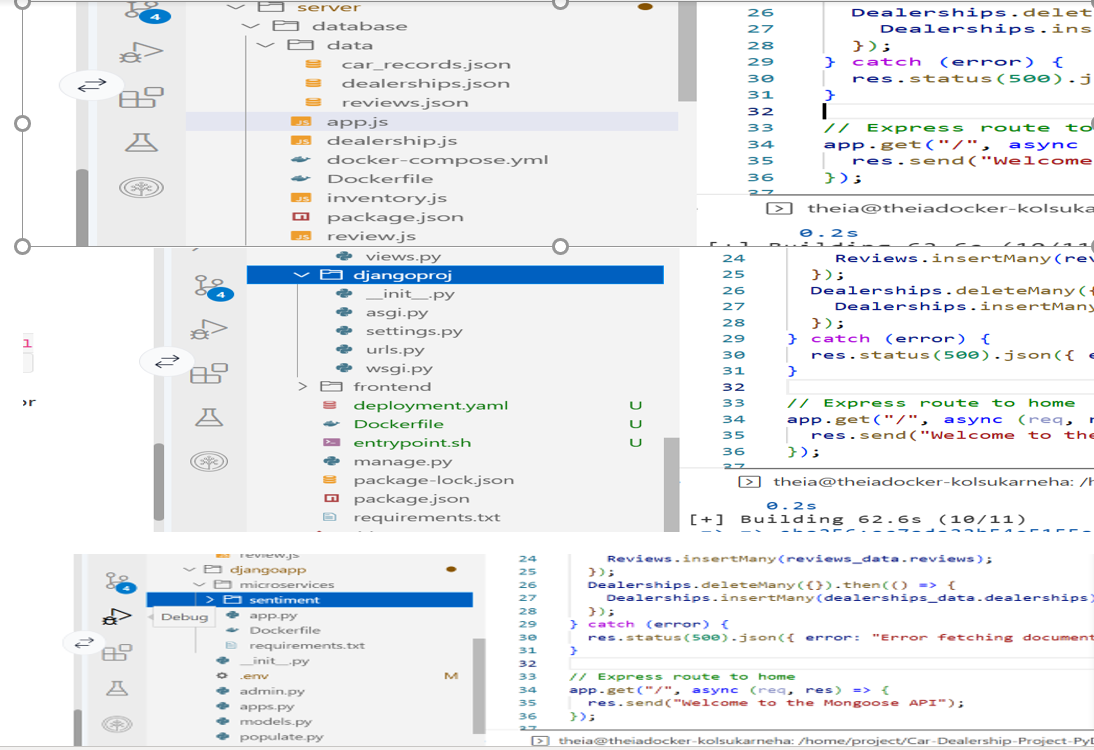
Database folder has MongoDB App. It’s being wrapped in docker once the app is build, you will get the link created for mongoDB integration as backend-url which can be used in .env folder under djangoapp.

Djangoapp has folder microservices which has sentiment analysis app written using Flask wrapped in a docker. Once build using docker, link is being used for integration in .env folder under djangoapp.

Url folders has views. Views are making calls to MongoDB and Flask(Sentiment app) API’s through request library.

There is another folder djangoProj

**Understanding on Deployment files and flow:-**



As per above folder structure we have **frontend, database, DjangoApp/microservices, DjnagoProject** under the **server** folder.

**Server**

* **Database**/MongoDB (Docker)
* **DjangoApp**/Microservices(Flask) (Docker)
  + .env
  + Views
  + url
  + models
  + Other required scripts
* **DjangoProj**
  + Settings – Directory for frontend/ static folder/Allowed\_Hosts/CSRF\_TRUSTED\_ORIGIN/Additional\_Installed\_Application/Middleware/Database(Django Database)
  + urls
* **Frontend** – run

(npm install

npm run build)

* Created **Docker file** under the main folder server.
  + Port expose
  + App
  + Location of entrypoint script for Django app
  + Default run command
  + Copy requirement file
* **entryPoint.SH** – It has make Migration commands related to Django Application
* **Requirement File**- Usually software to run applications
* **Deployment.yml** – Kub deployment reference  (image: us.icr.io/sn-labs-kolsukarneha/dealership:latest)

Sequence of execution - Docker Database/Docker FLASK and get the URL – Add the URL in .env under Django APP – Build frontend – Docker filer for server folder which require entrypoint.sh – Push it to the IBM Cloud repo/registry - last remaining thing is deployment file for .yaml (For kub deployments) which will pull image from IBM registry and setup a pod.

Useful commands:-

* **Build the nodeapp.**
  + docker build . -t nodeapp
* **Run the following command to start the server.**
  + docker-compose up
* **When not running docker - local DjangoSetup** 
  + cd /home/project/xrwvm-fullstack\_developer\_capstone/server
  + pip install virtualenv
  + virtualenv djangoenv
  + source djangoenv/bin/activate
  + python3 -m pip install -U -r requirements.txt

- **Run the following command to perform models migration.**

* + python3 manage.py makemigrations
  + python3 manage.py migrate
  + python3 manage.py runserver
* **Sentiment Analyzer app deploy to IBM Code engine:-** 
  + docker build . -t us.icr.io/${SN\_ICR\_NAMESPACE}/senti\_analyzer
  + docker push us.icr.io/${SN\_ICR\_NAMESPACE}/senti\_analyzer
  + ibmcloud ce application create --name sentianalyzer --image us.icr.io/${SN\_ICR\_NAMESPACE}/senti\_analyzer --registry-secret icr-secret --port 5000
* Create Docket file
* Create EntryPoint Sh
* **Kub deployment Command** - kubectl apply -f deployment.yaml
* **Port Forward Command:-** kubectl port-forward deployment.apps/dealership 8000:8000

**Understanding on entire project:-**

**Backend Component:-**

Views

* Login - By Django login Module -- API connected to Frontend URL mentioned in URL folder
* Logout - By Django logout Module -- API connected to Frontend URL mentioned in URL folder
* Registration - By Django login Module -- API connected to Frontend URL mentioned in URL folder
* Get dealer details – Saved in Mongo DB - API created using express – then called using RestAPI using request library used within getdealship views – views attached to url - Frontend URL mentioned in URL folder
* Get individual dealer - RestAPI folder using request library
  + Get dealer reviews (Sentiment Analysers) - RestAPI folder using request library
* Add reviews - RestAPI folder using request library -- API connected to Frontend URL mentioned in URL folder
* Get cars – From Django directory

**FrontEnd Component:-**

* React Components :-
  + Dealers
    - Dealers
    - Dealer on click dealer with id.
  + Header
  + Login
  + Logout - redirect to login with remove login
  + Register - (csrf\_exempt)
* Static Pages:-
  + About
  + Contact
  + Home