Week 1

Created a web server using node js. Then I was taught about how to use express to built a default web server. Learnt how to build a router using express router to connect to various endpoints in a URI assignment of week 1 Enable all those API to handle methods like get put post delete.

Week 2

This week was all about learning how to store data in mongo db, Its quite a hassle when we have to write a code for building a server . there is a library called express generator that creates a default Intro to mongo db How to connect to a mongo db. How to use mongoose odm and mongoose schema Finally building a full fledged rest API server that can take request from user and extract data from db then return the relevant data back to user In the form of json. the assignment for week 2

Task 1

- The Promotions schema and model correctly supports all the fields as per the example document given above

- The label field is set to an empty string by default

- The price schema is be supported with a new SchemaType called Currency.

- The REST API endpoints /promotions and /promotions/:promoId are implemented to interact with the MongoDB database

Task 2

- The Leaders schema and model correctly supports all the fields as per the example document given above.

- The REST API endpoints /leaders and /leaders/:leaderId are implemented to interact with the MongoDB database Week 3 How to use cookies and sessions . difference between them . how to use passport and how to authenticate a user using facebook , google etc. learnt about token based authentication

Week 3

How to use cookies and sessions . difference between them . how to use passport and how to authenticate a user using facebook , google etc. learnt about token based authentication Used to validate credentials, Then learnt how to use mongoose population. main assignment

Task 1

- You have implemented the verifyAdmin() function in authenticate.js.

- The verifyAdmin() function will allow you to proceed forward along the normal path of middleware execution if you are an Admin

- The verifyAdmin() function will prevent you from proceeding further if you do not have Admin privileges, and will send an error message to you in the reply.

Task 2

• Any one is restricted to perform only the GET operation on the resources/REST API end points.

• An Admin (who must be first checked to make sure is an ordinary user), can perform the GET, PUT, POST and DELETE operations on any of the resources/ REST API end points.

Task 3

• A GET operation on http://localhost:3000/users by an Admin will return the details of the registered users

• An ordinary user (without Admin privileges) cannot perform the GET operation on http://localhost:3000/users.

Task 4

• A registered user is allowed to update and delete his/her own comments.

• Any user or an Admin cannot update or delete the comment posted by other users. Week 4 Learnt about various key cryptography (public and symmetric mainly) Learnt about cross origin resource sharing and how to implement it. the main assign.

Task 1

In this task you will be implementing a new Mongoose schema named favoriteSchema, and a model named Favorites in the file named favorite.js in the models folder. This schema should take advantage of the mongoose population support to populate the information about the user and the list of dishes when the user does a GET operation.

Task 2

In this task, you will implement the Express router() for the '/favorites' URI such that you support GET, POST and DELETE operations

• When the user does a GET operation on '/favorites', you will populate the user information and the dishes information before returning the favorites to the user.

• When the user does a POST operation on '/favorites' by including [{"_id":"dish ObjectId"}, . . ., {"_id":"dish ObjectId"}] in the body of the message, you will (a) create a favorite document if such a document corresponding to this user does not already exist in the system, (b) add the dishes specified in the body of the message to the list of favorite dishes for the user, if the dishes do not already exists in the list of favorites

. • When the user performs a DELETE operation on '/favorites', you will delete the list of favorites corresponding to the user, by deleting the favorite document corresponding to this user from the collection.

• When the user performs a POST operation on '/favorites/:dishId', then you will add the specified dish to the list of the user's list of favorite dishes, if the dish is not already in the list of favorite dishes.

• When the user performs a DELETE operation on '/favorites/:dishId', then you will remove the specified dish from the list of the user's list of favorite dishes. Task 3 You will update app.js to support the new '/favorites' route.