Evaluation Metrics

Backend

Basic

1. **MVC** pattern followed 10%
2. Frequent use of **ES6** syntax (arrow fn, spread op, promises, async-await) 5%
3. **Code readability** - commenting on codes, appropriate variable names, etc.
   On how many occasions DRY, SOLID concepts were used? 5%
4. **Error handling** being incorporated? - server down, duplicate data entry, wrong inputs from the user, null values, unicode (prevent hackers), etc. 5%
5. Minimum 5000 **mock data** dumping. students may use the option of seeder.js for the creation and deletion of data. Use mockaroo for random data generation. 5%
6. How many **aggregations** used on the main endpoint. Minimum CRUD endpoints should be there. More the number of aggregations more extra marks will be given. (aggregation here means many things including search, sort, limit, skip, etc) 20%
7. **DB schema** - choice between mongoose or sequelize should be justified. (schema here means columns their names and their types) 10%
8. **Deployment** on mongo-cloud/heroku mandatory
9. **Project track** must be updated on Trello board and github. mandatory
10. **Deadlines** to be followed - 2.5 weeks for preparation and 3 days for demonstration and feedback. mandatory

Moderate

1. Email sending for different applications, for example ‘forgot password’, ‘email verification’ etc, image uploading. 10%
2. Implement request rate-limiting for the server. 10%
3. Authentication headers through all endpoints. 10%

Advanced

1. Testing of every endpoint for edge case handling. 10%

Frontend

Structure - HTML

1. Usage of semantic tags to build the html structure 5%

Styling - CSS

1. UI/UX Design 5%
2. Page responsiveness 5%
3. CSS preprocessors are used 5%

React-Redux

1. Small, reusable components have been built 10%
2. Data flow is easy to understand, and debug, i.e it follows a top-down approach if redux is not used to pass data 5%
3. If redux is used, application level data is stored in the redux store, there are multiple reducers handling different parts of the application logic 10%
4. Component Logic is extracted using design patterns like HOC, Render Props 7.5%
5. API calls are made asynchronously using REDUX-THUNK as a middleware, not stopping the flow of the program, and data updates happen in the components asynchronously 10%
6. Proper separation of actions has been done 5%
7. Only a component which needs data is connected to the store, and gets only that data which is needed by it 5%
8. All data is not stored in the redux store, and data which is only relevant to a component, is locally stored 5%
9. Routing system is in place, and works well 7.5%
10. If you refresh on a route, the page stays and not routes back to home page 5%
11. Components that don't require state are made into functional components 5%
12. Testing on logics, components has been added 5%