<epam>

Prerequisite

Node.js and MongoDB installed on your work station
I expect, you have spent time on this book reading

Description

Implement UBER like service for freight trucks, in REST style, using MongoDB as database. This service should help regular people to deliver their stuff and help drivers to find loads and earn some money. Application contains 2 roles, driver and shipper.

IMPORTANT: *all requirements and acceptance criteria items are strict and mandatory, since automated tool will check quality of your work.*

Acceptance criteria:

- Driver is able to register in the system;
- Driver is able to login into the system;
- Driver is able to view his profile info;
- Driver is able to change his account password;
- Driver is able to add trucks;
- Driver is able to view created trucks;
- Driver is able to assign truck to himself;
- Driver is able to update not assigned to him trucks info;
- Driver is able to delete not assigned to him trucks;
- Driver is able to view assigned to him load;
- Driver is able to interact with assigned to him load;
- Shipper is able to register in the system;
- Shipper is able to login into the system;
- Shipper is able to view his profile info;
- Shipper is able to change his account password;
- Shipper is able to delete his account;
- Shipper is able to create loads in the system;
- Shipper is able to view created loads;
- Shipper is able to update loads with status ‘NEW';
- Shipper is able to delete loads with status 'NEW';
- Shipper is able to post a load;
- Shipper is able to view shipping info;
- API documentation uploaded to repository;
- README with basic information on how to set up and launch your project should be included (also what external software should be installed to run your app);
- Project logic distributed across different directories and files in simple and easy-to-understand structure, MVC pattern will be a plus;

- Source code uploaded to repository and link sent to google sheets document with all homeworks.
- Gitlab repo link and project id are saved in Google spreadsheets by link: https://docs.google.com/spreadsheets/d/1_PhsGuS_ucQO1WWaarGxdb6aYvAs6remqFLFHHVKCkY/edit?usp=sharing
- Simple UI for your application(would be a big plus).

Optional criteria

- Any system user can easily reset his password using 'forgot password' option;
- User is able to attach photo to his profile;
- Any system user can see weather information which should be stored on server side;
- User can generate reports about shipped loads, in excel or pdf formats and download them;
- Ability to filter loads by status;
- Pagination for loads;
- [UI] User can interact with application through simple UI application(choose comfortable for you framework or use native js);
- [UI] Shipper is able to see his load info(pick-up address, delivery address), and [UI] driver assigned to his load coordinates on the map on UI;
- [UI] Driver is able to see info about assigned to him load(pick-up address, delivery address) on the map on UI;
- [UI] Any system user is able to interact with the system UI using a mobile phone without any issues;

🤘🎸Rockstar criteria

- Driver and Shipper can contact each other through simple chat related to load;
- Driver and Shipper can receive real time shipments updates through WS;
- The most important functionality covered with unit and acceptance tests;
- [UI] Ability for any system user to choose language on UI(localization);
- Application can handle time zones difference and notify driver if needed;
- Any system user can get notifications through the email about shipment updates;

Requirements:

- Mandatory npm start script.
- Please check that your app can be launched with 'npm install' and 'npm start' commands just after git pull;
- Ability to run server on port which is defined as PORT environment variable, default to 8080;
- Application code should follow eslint rules described in .eslintrc.json file(requires eslint and eslint-config-google packages install);

- Use express to implement web-server;
- Use express Router for scaling your app and MVC pattern to organise project structure;
- Use JWT as strategy for authentication;
- Use jsonwebtoken package for jwt authentication;
- Use mongoose as ODM for MongoDB;
- Use Mongo Atlas for MongoDB connection;
- Application should validate all endpoints parameters with Joi npm package;
- Follow REST API rules described in Swagger file;
- All incoming requests should be logged;
- All errors should be handled by server and appropriate status and error message should be sent in response to client;
- All operations such as posting a load, assigning load to driver etc. should be logged;
- Git repo should not contain secrets or other sensitive info, use predefined defaults instead.
- Send response to client only when server finished to process client request and all needed operations finished;
- Use config or dotenv npm modules for configuration, anyone should be able to run your project with config defined in config file or environment variable;
- All users passwords should be stored as encrypted in database;
- Encode all users password with bcrypt npm package.

Notes and hints:

- It's better to use additional layer, DAO, for heavy and tricky requests to database;
- MongoDB aggregations might be useful for some database requests;
- OOP might help to organize your code and different operations better;
- Moment library can be useful if you want to handle time or timezones;
- Use socket.io in case you want to implement WS in your app;
- Request-promise package can be useful in case you want to send http request to 3rd party service;
- Please check ability to run your project after pulling and installing modules;
- In case you want to deploy your application somewhere, you can contact Kyrylo_Yezhov@epam.com.
- In case of any issues or questions, please contact your mentors or Kyrylo_Yezhov@epam.com.

Evaluation Criteria:

- 60 point - some of the acceptance criteria items included to implementation(at least 10);
- 70 points - some of the acceptance criteria items included to implementation(at least 20);
- 80 points - all acceptance criteria items implemented, code looks good, automated testing tool returns success at least for 60% of specs;
- 90 points - all acceptance criteria items implemented, code looks good, automated testing tool returns success at least for 90% of specs and at least 3 optional criteria items implemented;
- 100 points - all acceptance criteria items implemented, code looks good, automated testing tool returns success at least for 100% of specs and at least 5 optional or rockstar criteria items implemented;