interview project

Express & MongoDB

Overview

As the next step in your interview process for a software development position with Joot, we are asking you to complete a short project. This will help us assess your ability to quickly learn the technologies you would use in this role.

This project is intended to take no more than 2-3 hours to complete if you already have some experience with Node.js, Express, and MongoDB, and no more than 4-5 hours to complete if this is your first time ever using these technologies.

Project instructions

In this section, we will provide some instructions to point you in the right direction as you complete this project. We have intentionally provided limited instructions as an important part of this exercise is your ability to find and learn from online resources.

- Set up your development environment and bootstrap a Node.js project.
    - You must use Express API framework, MongoDB, and Mongoose ODM. You are otherwise free to implement any libraries of your choosing.
- Provide simple API documentation in whatever format you feel is best to aid with our testing.
- Create a GitHub repository and push your project source code.
- Send a link for your GitHub code repository to Kyle Gundrum at kyle@refractlabs.com and joe.stazak@joot.io.
    - Make your code repository public or add my GitHub username (KyleGundrum and joestazak-joot) to your private repository.

Application Requirements

In this section, we will review the application requirements. These requirements constitute only high-level, general capabilities the application should support. Aside from implementing Express, MongoDB, and Mongoose, you are free to use any other libraries and technologies in your app.

You are building a backend for a fictional bicycle shop website that the store's employees use to organize their customers, inventory, and scheduled repairs. The backend will handle requests from the frontend to create, read, update, and delete data related to customers, inventory, and repairs.

Create all the following API endpoints and related database models for the application modules listed below. Each section begins with a brief description of the solution requirements from which you can design data models.

Customers

We want to keep track of customer contact information, how long each customer has shopped with us, and any details that may help with quickly scheduling repairs for each customer.

- POST /customers – create a new customer

    - Post JSON object with customer details
    - Return object with new created Mongo document
- GET /customers – returns all customers in database
    - Return array of all customer objects
- GET /customers/{id} – return customer {id}, e.g., /customers/7 returns customer ID 7
    - Returns customer object for given ID in URL route
- PUT /customers/{id} – updates customer {id}
    - Returns updated customer object for given ID in URL route
- DELETE /customers/{id} – deletes customer {id}
    - Returns { deleted: 1 }

Inventory

We need to organize our store item catalog and current stock. Items are categorized as bicycles, parts, or accessories.

- POST /inventory – add item to inventory
    - Post JSON object with item details
    - Return object with new created Mongo document
- GET /inventory – returns all items in database
    - Return array of all item objects
- GET /inventory/{id} – return item {id}
    - Returns item object for given ID in URL route
- PUT /inventory/{id} – updates item {id}
    - Returns updated item object for given ID in URL route
- DELETE /inventory/{id} – deletes customer {id}
    - Returns { deleted: 1 }

Repairs

We need to know which established customer has scheduled the repair, any information we received from the customer about what needs to be repaired, and details about the bicycle that will help our technicians effectively prepare for their repair schedule.

- POST /repairs – add scheduled repair to inventory
    - Post JSON object with repair details
    - Return object with new created Mongo document
- GET /repairs – returns all scheduled repairs in database
    - Return array of all scheduled repair objects

- GET /repairs/{id} – return scheduled repair {id}
    - Returns scheduled repair object for given ID in URL route
- PUT /repairs/{id} – updates scheduled repair {id}
    - Returns updated scheduled repair object for given ID in URL route
- DELETE /repairs/{id} – deletes scheduled repair {id}
    - Returns { deleted: 1 }
- POST/repairs/{id}/schedule – updates scheduledDate for repair {id}
    - Post JSON object { scheduledDate: *date* }
    - Returns updated schedule repair object

Project assessment

We will consider several criteria as we review your project. These are simply guidelines, not a fixed rubric.

- **Project Completion** – did you complete the entire project in the expected amount of time?
- **Code Quality** – is your code generally clean, or is it sloppy and using bad conventions? Is your codebase structured according to reasonable Express best practices?
- **API Functionality** – do all the endpoints work as documented?

We should be able to test your submission by cloning your repository, installing dependencies, updating MongoDB connection details to a blank local database of our own, and running “npm start”. We will then test each of the endpoints listed above.