Period-2 Node, Express with TypeScript, JavaScript Backend Testing, MongoDB (and Geo-location)

LAVET AF: Christian Madsen, Frederik Hurup & Jacob Simonsen

Note: This description is too big for a single exam-question. It will be divided up into separate questions for the exam

Explain Pros & Cons in using Node.js + Express to implement your Backend compared to a strategy using, for example, Java/JAX-RS/Tomcat

Det hele er bygget sammen til et stort fullstack modul i samme mappe, hvilket gør nemmere for udvikleren til at holde styr på sin kode, i stedet for at skulle have flere forskellige moduler rundt omkring med database, front samt backend. Ydermere er det lettere for udvikleren at sætte projektet op (package.json) i forhold til java.

Explain the difference between *Debug outputs* and *ApplicationLogging*. What's wrong with console.log(..) statements in our backend code?

Forskellen er at debug outputs bliver printet i terminalen når der bliver kørt debug:test mens Application Logging logger alle endpoints requests til en logfil.

Console.log er et problem da det logger til browserens console, der gør at end-brugeren kan se statementet/printet.

Demonstrate a system using application logging and environment controlled debug statements.

Se app.ts hvor logger.ts er importeret og brugt som middleware så hver gang der bliver lavet et request til et endpoint under/api/friends bliver det logget til logfilen (CombinedLog eller ErrorLog).

Explain, using relevant examples, concepts related to testing a REST-API using Node/JavaScript/Typescript + relevant packages

se /src/test/friendEndpointsTest.ts hvor der bliver brugt Mocha som test-environment, sammen med supertest som bruges til at starte en test-server. Der bliver brugt Chai der gør det muligt at asserte de testede værdier.

En test er bygget op via et describe shell med et 'before' (til at starte en InMemory DB), 'beforeEach' (til at reset tabellerne med data) efterfulgt af 'it' (som er selve testene)

Explain a setup for Express/Node/Test/Mongo-DB/GraphQL development with Typescript, and how it handles "secret values", debug, debug-outputs, application logging and testing.

Secret values bliver håndteret med en .env fil, hvor f.eks. connectionstrengen til db ligger. Værdierne bliver hentet ind i koden ved at gøre brug af 'dotenv' biblioteket. Programmet kan bruge .env filen til at diktere hvilket stadie koden er i (development/production), og derved om der skal logges til konsollen eller logfiler.

Explain a setup for Express/Node/Test/Mongo-DB/GraphQL development with Typescript. Focus on how it uses Mongo-DB (how secret values are handled, how connections (production or test) are passed on to relevant places in code, and if use, how authentication and authorization is handled

Programmet connecter til databasen via connectionstrengen som bliver importeret fra .env filen for at skjule database credentials.

Se /src/config/dbConnector.ts hvor connectionen bliver lavet, enten til mongodb eller inMemoryDB. Selve MongoDB connection bliver oprettet i /bin/www.ts, hvor databasen bliver sat på app objektet så den er tilgængelig i de relevante filer. InMemoryDB connection bliver oprettet i testfilerne.

Explain, preferably using an example, how you have deployed your node/Express applications, and which of the Express Production best practices you have followed.

Explain possible steps to deploy many node/Express servers on the same droplet, how to deploy the code and how to ensure servers will continue to operate, even after a droplet restart.

Explain, your chosen strategy to deploy a Node/Express application including how to solve the following deployment problems:

- **Ensure that you Node-process restarts after a (potential) exception that closed the application**
- **Ensure that you Node-process restarts after a server (Ubuntu) restart**
- **Ensure that you can run "many" node-applications on a single droplet on the same port (80)**

KAN IKKE SVARE PÅ EFTERSOM VI IKKE HAR HAFT OM DEM

Explain, using relevant examples, the Express concept; middleware.

Middleware-funktioner er en række af funktioner som har adgang til req, res objekterne og next funktionen. De kan ændre på req og res objekterne når deres kode bliver kørt, samt stoppe req/res cyklussen, hvis next() ikke bliver kaldt, og dermed blokere programmet.
Eksempel kan ses i src/routes/friendRoutesAuth.ts, hvor der bliver brugt authentication middleware til at give korrekt adgang eller ikke give adgang til de forskellige endpoints.

Explain, conceptually and preferably also with some code, how middleware can be used to handle problems like logging, authentication, cors and more.

I forhold til logging, som man kan se i src/app.ts, bliver middleware brugt som opsætning inden man når selve endpointsne. Det er vigtigt at loggeren ligger først, ellers vil den ikke logge noget, eftersom rækkefølgen er afgørende. Authentication og cors bliver brugt som et check, hvor rækkefølgen igen er vigtig.

Explain, using relevant examples, your strategy for implementing a REST-API with Node/Express + TypeScript and demonstrate how you have tested the API.

Vores REST-API ligger i src/routes/friendRoutesAuth.ts, hvor vi ved brug af express' routes objekt har lavet de routes, som er relevante i forhold til metoderne lavet i friendFacaden. Yderligere er der brugt authentication på de routes, som ikke skal kunne tilgås af alle. Her er der brugt et rollesystem, som gør det muligt for en bruger, kun at kunne se og ændre sig selv. Hvorimod en admin bruger har rettigheder til at tilgå alle endpoints.

Endpoints er testet via Mocha, med brug af supertest & Chai. En test er bygget op med en beskrivelse, et request, en eventuel authorization, og assertion af værdier.

Explain, using relevant examples, how to test JavaScript/Typescript Backend Code, relevant packages (Mocha, Chai etc.) and how to test asynchronous code.

Backend test er lavet via at oprettet forbindelse til inMemoryDB via 'Before' hvorefter dataet bliver resettet i 'BeforeEach'.

Hver test element har et describe efterfulgt af tests, for at teste asynchronous funktioner er der brugt await.

NoSQL and MongoDB

***Explain*, generally, what is meant by a NoSQL database.**

NoSQL er non-relational database som ikke kræver en prædefineret struktur, ydermere har den collections og documents i stedet for tabeller og rækker. Det gør den specielt effektiv til meget store datasæt, eftersom den er horisontalt skalérbar.

***Explain* Pros & Cons in using a NoSQL database like MongoDB as your data store, compared to a**

traditional Relational SQL Database like MySQL.

Fordelene er at den er væsentligt hurtigere, og at man ikke behøver definere alle collections på forhold. Det kan den selv finde ud. En anden fordel er at man ret nemt kan gøre databasen meget større (horisontalt skalérbar).

Ulemperne er at man ikke kan lave relationelle collections, og derved kan det være svært/umuligt at samle data fra flere collections, og hvis man har al data i den samme tabel kan man hurtigt miste overblikket.

***Explain* about indexes in MongoDB, how to create them, and *demonstrate* how you have used them.**

Indexes i MongoDB opfører sig på samme måde som i andre datamodeller udover at man kan lave subindexes på hvilket som helst field eller subfield.

I MongoDB bliver der automatisk genereret et id, når man indsætter et dokument, og det er det eneste vi har brugt.

Explain*, *using your own code* examples, how you have used some of MongoDB's "special" indexes like *TTL* and *2dsphere and perhaps also the Unique Index.

Ikke brugt endnu.

***Demonstrate*, using your own code samples, how to perform all CRUD operations on a MongoDB**

Insert, Find, Update og Delete er alle sammen brugt i src/facades/friendFacade.ts

Demonstrate how you have set up sample data for your application testing

Se i test/* hvor der er i "beforeEach" er blevet brugt insertMany() til at indsætte sample data.

Explain the purpose of mocha, chai, supertest and nock, which you should have used for your testing

Mocha er brugt som test miljø, chai er et assertion bibliotek, supertest er brugt til at generere en test server(endpoint-testing) og nock er brugt til mock HTTP-requests.

Explain, using a sufficient example, how to mock and test endpoints that relies on data fetched from external endpoints

Se i test/whattodoTest.ts hvor nock er brugt til at mocke svarene fra externe API'er, så det er muligt at teste pålidelig data.

Nock opsnapper requested til API'et og returnere et selvgenereret response, som følger det samme format/opbygning.

Explain, using a sufficient example a strategy for how to test a REST API. If your system includes authentication and roles explain how you test this part.

Se test/friendEndpointTest.ts

Endpoints er testet via Mocha, med brug af supertest & Chai. En test er bygget op med en beskrivelse, et request, en eventuel authorization, og assertion af værdier.

Eksempel: /api/friends/editme

Request bliver sent med en body med information der skal redigeres. Sammen med requestet bliver der sendt et .auth request indeholdende korrekte login credentials, hvorefter der bliver testet på responset. (Statuskode, modifiedCount).

***Explain*, using a *relevant example*, a full JavaScript backend including relevant test cases to test the REST-API (not on the production database)**

Se startkoden :)