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

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

There is no doubt about it's much quicker and might even easier to setup. It gives a more simple solution.

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

Debug statements are only in our developer environment. And can be turned off by one variable. ApplicationLogging is only when middleware is being used, and lets us keep information about errors in a log file.

Console.log shows potential flaws in a browser, which is a risk.

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

Application logging: startcode/src/app.ts

Environment controlled debug statement: We can change the NODE_env variable in .env to production.

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

We have begun testing using Mocca, Chai and Nock packages to also make tests easier for non-coding people.

startcode/test/friendEndpointsTest.ts

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.

The startcode itself portraits a great image of that.

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

Values that are secret will be stored in the .gitignore'd file .env, where we will find things such as Mongo-db connection string, which we would NOT want public. Our dbConnector class has two

classes. The InMemoryConnector which creates a mock db which is a safe test environment and is deleted after use. And then there is the dbConnector which is NOT for testing but for production use, which means it connects to our "real" database, and luckily doesn't wipe that.

The dbConnector is made in the www.ts file in /bin. Here we set the db type to REAL(line 12). We set the db and db-type on app so it can be used globally.

The db-type is set to test-db in the test files, so it shows in the log files that it was used for testing.

Authentication is handled using the middleware called Basic-auth which is implemented in the friendRoutes. The friendRoute checks the .env file an env variable called SKIP_AUTHENTICATION, if the variable is set to ANY, the friendRoute will skip using authentication. Otherwise if not set, we use Basic-auth middleware on the route.

startcode/src/routes/friendRoutesAuth.ts (line 37)

Authorization is determined by the .env variable and the credentials given when a user logs in.

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)

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

Middleware: functions that handles request and response objects. The middleware will block the application, which is why you should invoke the next() method in your middleware, if you are no returning the response back to the client.

startcode/src/middleware

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

Middleware example: startcode/src/middleware

And the usage of middleware: startcode/test/friendEndpointsTest.ts

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

The friendRoutes file handles the routing between the endpoint and related facadefunction.

startcode/src/routes/friendRoutesAuth,ts

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

Mocha is a test framework
Chai is an assertframework

startcode/src/facades/friendFacade.ts

NoSQL and MongoDB

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

NoSQL databases are non-relational, so, no relations in-between tables. There is no specific pattern.

***Explain* Pros & Cons in using a NoSQL database like MongoDB as your data store, compared to a traditional Relational SQL Database like MySQL.**

NoSQL databases can contain all types of data in the same scheme. MongoDB has no relation-mapping between documents.

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

_id is a default unique index which contains a date.

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.

Non existent as of now.

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

All examples for the CRUD operations can be found in

startcode/src/facades/friendFacade.ts

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

Before and beforeEach functions

startcode/test/friendEndpointsTest.ts

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

startcode/test/friendEndpointsTest.ts

Mocha is a test-framework for async testing.
Chai is an assertion framework.
Supertest sets up test endpoints for us to call when running tests.
Nock used for testing external endpoints.

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

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.

Any test with .auth in:

startcode/test/friendEndpointsTest.ts

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

The whole project works as a full javascript backend.

Geo-location and Geojson (Period-4)

Explain and demonstrate basic Geo-JSON, involving as a minimum, Points and Polygons

Explain and demonstrate ways to create Geo-JSON test data

Explain the typical order of longitude and latitude used by Server-Side APIs and Client-Side APIs

Explain and demonstrate a REST API that implements geo-features, using a relevant geo-library and plain JavaScript

Explain and demonstrate a REST API that implements geo-features, using Mongodb's geospatial queries and indexes.

Explain and demonstrate how you have tested the gameFacade and gameAPI for the game-related parts of the period exercises

This will come in period-5

Explain and demonstrate a React Native Client that uses geo-components (Location, MapView, etc.)

Explain and demonstrate both server and client-side, of the geo-related parts of your implementation of the ongoing semester case.