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

- Simplicity and quicker to set up.

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

- Debug statements replace console.log() statements, and can be turned off by an environment variable. This means that we can stop unnecessary output to console, and log the important things using applicationLogging. Application logging is a middleware, used to log events in a log file.

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

Example:

- *Logging:* fullstack-startcode/src/app.ts (L. 13 – 17)
- *env controlled debug statements:* fullstack-startcode/src/routes/friendRoutesAuth.ts

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

Chai, mocca and Nock packages.

Example:

- fullstack-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.

- This question is answered by the answers given in the first four questions

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
- Secret values and connection string to Mongo-DB are stored in a .env file. The dbConnector has two different classes, (DbConnector and InMemoryConnector). The InMemoryConnector creates a mock db, which is used in the test environment, and it wiped after usage (ideal for testing). The DbConnector is used for production, and connects to the actual Mongo-DB server.

Example:
- fullstack-startcode/src/config/dbConnector.ts
- fullstack-startcode/src/bin/www.ts

The db connection is established in the src/bin/www.ts file, where the db-type is set to "REAL". In the test files, the db-type is set to TEST-DB on the app instantiated by the supertest package.

Example:
- fullstack-startcode/test/friendEndpointsTest.ts

The authorization is handled by using the Basic-Authentication middleware. An environment variable called SKIP_AUTHENTICATION is used, to enable/disable authentication. In the example, use authentication middleware is implemented, by checking if the environment variable is set to anything.

Example:
- fullstack-startcode/src/routes/friendRoutesAuth.ts

Authorization of routes are determined by the use of authentication .env variable, and the credentials in the request object. These credentials are set in this file /src/middelware/basic-auth.ts. Furthermore, using the Joi package, requirements for adding a user is determined.

Example:
- Joi: fullstack-startcode/src/facades/friendFacade.ts

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.
- Express is a node-js framework, which writes handlers for different routes. It can also set different web application settings. The Request and Response object can be imported from the express framework, which handles the HTTP protocol. fullstack-startcode/Src/app.ts
- Middleware is *"the glue that connects different software platforms and devices together"*. Middleware will block other request from the express framework, which is why the *next()* function is used, when implementing middleware, if the response object is not returned to the client.
fullstack-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: fullstack-startcode/Src/middleware
Use of middleware: fullstack-startcode/src/app.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 strategy for implementing REST-API with Node/Express + TypeScript is handling routes with the express framework.
Example:
- fullstack-startcode/src/routes/friendRoutesAuth.ts
- fullstack-startcode/test/friendEndpointsTest.ts

Testing these endpoints, is done by using the Nock package. Nock is a server mocking and expectations library for Node.js. It can be used to test modules that perform HTTP requests in isolation.
Example: fullstack-startcode/playground/usingnock/whattodoTest.ts

Explain, using relevant examples, how to test JavaScript/Typescript Backend Code, relevant packages (Mocha, Chai etc.) and how to test asynchronous code.
Example:
- Façade test: fullstack-startcode/test/friendFacadeTest.xxx
- Endpoint test: fullstack-startcode/test/friendEndpointsTest.ts

NoSQL and MongoDB

***Explain*, generally, what is meant by a NoSQL database.**
noSQL databases are non-relational. This means your data does not need to follow a specific pattern, in order to store data.

***Explain* Pros & Cons in using a NoSQL database like MongoDB as your data store, compared to a traditional Relational SQL Database like MySQL.**
NO-SQL databases can contain all types of data in the same schema. These types of databases are simper to scale horizontally (adding more machines).

***Explain* about indexes in MongoDB, how to create them, and *demonstrate* how you have used them.**
"MongoDB creates a unique index on the _id field during the creation of a collection. The _id index prevents clients from inserting two documents with the same value for the _id field. You cannot drop this index on the _id field.".
The _id object contains a lot of data, including the timestamp of data insertion.

The example uses the default _id, created by mongoDB
Example:
https://account.mongodb.com/account/login

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.
???

***Demonstrate*, using your own code samples, how to perform all CRUD operations on a MongoDB**
Examples for all CRUD operations, can be found here:
Example: fullstack-startcode/src/facades/friendFacade.ts

Demonstrate how you have set up sample data for your application testing
- In a beforeEach.
Example: fullstack-startcode/test/friendEndpointsTest.ts

Explain the purpose of mocha, chai, supertest and nock, which you should have used for your testing
- mocha is a test framework for asynchronous testing.
Example: fullstack-startcode/test/friendFacadeTest.xxx

- chai is used for **expect** API, for Behavior-driven development.
Example: fullstack-startcode/test/friendFacadeTest.xxx

- supertest is a node package, for testing HTTP. It sets-up a server (request object) automatically for testing.
Example: fullstack-startcode/test/friendEndpointsTest.ts

- Nock is a server mocking and expectations library for Node.js. It can be used to test modules that perform HTTP requests in isolation.
Example: fullstack-startcode/playground/usingnock/whattodoTest.ts

Explain, using a sufficient example, how to mock and test endpoints that relies on data fetched from external endpoints
Example: fullstack-startcode/playground/usingnock/whattodoTest.ts

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.
- With supertest, we use the request object from supertest, to make requests, to get a response.
Authentication can be put on this request object (see example)
Then the response is tested, using chai.
Example: fullstack-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)**

A full javascript backend, is this general project:
Example: fullstack-startcode/
link: https://github.com/SebastianBentley/fullstack-startcode

Geo-location and Geojson (Period-4)

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.