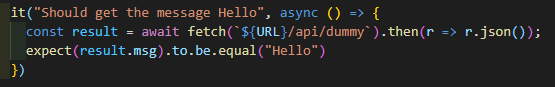
Period-2 Node, Express with TypeScript, JavaScript Backend Testing, MongoDB and Geo-location

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**  
  
Pros:  
Easier to be more specific of what you want.  
Easier to update modules.  
Fast setup for any other that needs to work on the project.  
Able to change between ECMA versions if needed.  
Able to setup your own specific scripts that you can run.  
Debatable if it’s generally faster to start/setup.  
Still being heavily developed. (There is ongoing support for this, so it’s not like a new language that could just suddenly die.)  
  
Cons:  
If you choose an ECMA version and want to use something between different builds there might be an issue of something has been removed/deprecated?  
Personally, it’s slower currently as I do not have as much expertise with it yet.  
()  
Haven’t seen a way yet / researched if there is a way to work CI with checking tests before deploying.  
  
From <https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-node-js-web-app-development/>  
  
Pros:   
Speed and performance  
Easy knowledge sharing within a team  
A huge number of free tools  
V8 engine  
Scalable technology for microservices  
Highly used  
Strong corporate support  
Seamless JSON support  
  
Cons:  
Performance bottlenecks with heavy computation tasks  
Callback hell issue (We could see other programmers that has done this)  
Immaturity of tooling

**Explain the difference between *Debug outputs* and *ApplicationLogging*. What’s wrong with console.log(..) statements in our backend code.**The reason why console.log(..) is bad is because it’s blocking. And we do not want to slow down our program.  
Debug outputs as used with for example:  
  
  
We can then use debug(..) and we’ll stop at our debug statements and can see in console some more information about this.  
ApplicationLogging is for example where we used out middleware Winston  


**Demonstrate a system using application logging and environment controlled debug statements.**  
Any of the week2 projects except geo

**Explain, using relevant examples, concepts related to testing a REST-API using Node/JavaScript/Typescript + relevant packages**   
We use Chai and Mocha for our testing  
  
We describe our test by writing:  
  
In here we can then before/beforeEach  
  
Tests that are not ignored are marked with it(..) with some info and a body  
  
If they should be ignored they are marked with xit(..)  
  
We can expect as we got from chai to expect things.  
A lot more can be found at:   
<https://www.chaijs.com/guide/styles/>   
<https://www.chaijs.com/api/assert/>   
<https://www.chaijs.com/api/bdd/>

We can also get promises with Chai.  
  
  
  
See Readyforday6

**Explain a setup for Express/Node/Test/Mongo-DB development with Typescript, and how it handles "secret values", debug and testing.**  
Secret values are kept in a .env file that should be added to the gitignore.  
With testing we can also circumvent authentication if needed.

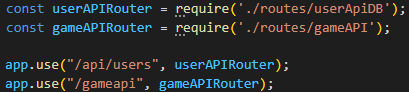
**Explain, preferably using an example, how you have deployed your node/Express applications, and which of the Express Production best practices you have followed.**  
Followed mostly <https://www.youtube.com/playlist?list=PLDbigcKhXkiW3w8RQ25QRwJD5OhuTj8HU>  
Since the Node.JS Quickstart setup for a droplet isn’t there anymore I installed these myself.  
<https://expressdemos.cphfb.codes/>

**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.**  
We would have do open several ports to several network hostnames.  
<https://www.youtube.com/watch?v=9Fe0mV9MhjI&list=PLDbigcKhXkiW3w8RQ25QRwJD5OhuTj8HU&index=4>

**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  
  Using a process manager this can be done. <https://www.youtube.com/watch?v=9Fe0mV9MhjI&list=PLDbigcKhXkiW3w8RQ25QRwJD5OhuTj8HU&index=4>
* Ensure that you Node-process restarts after a server (Ubuntu) restart  
  Used with a product manager PM2
* Ensure that you can run “many” node-applications on a single droplet on the same port (80)  
  The port in the application would have to be different from the other one, else then it’s just to add it to the PM2 manager again.  
    
  Example: <https://expressdemos.cphfb.codes/>

**Explain, using relevant examples, the Express concept; middleware.**  
<https://expressjs.com/en/guide/writing-middleware.html>   
<https://expressjs.com/en/guide/using-middleware.html>   
  
See readyForDay6

**Explain, using relevant examples, your strategy for implementing a REST-API with Node/Express + TypeScript and demonstrate how you have tested the API.**  
  
See readyForDay6  
For **rest** we have app.ts where we have our “main” routes go through.  
We can then add an extra Javascript file where we have several endpoints like  
  
And we would have middleware also that could be logging, authentication or other things.  
The package.json should look like below, so that we can easily, run our file, debug and deploy.

"scripts": {

"test": "mocha --exit -r ts-node/register test/\*\*/\*.ts",

"testDebug": "mocha --exit -r ts-node/register --inspect=5858 test/\*\*/\*.ts",

"dev": "node -r ts-node/register ./src/app.ts",

"dev:watch": "nodemon",

"devDebug": "node --inspect=5858 -r ts-node/register ./src/app.ts",

"start": "tsc && node ./build/app.js",

"build": "tsc",

"devNo-tsnode": "concurrently \"tsc -w\" \"nodemon ./build/app.js"

},

Dependencies:

"dependencies": {

"@types/winston": "^2.4.4",

"basic-auth": "^2.0.1",

"bcryptjs": "^2.4.3",

"chai-as-promised": "^7.1.1",

"concurrently": "^5.1.0",

"debug": "^4.1.1",

"dotenv": "^8.2.0",

"express": "^4.17.1",

"express-winston": "^4.0.3",

"mongodb": "^3.6.2",

"tsscmp": "^1.0.6",

"typescript": "^3.7.4",

"winston": "^3.2.1"

},

"devDependencies": {

"@types/bcryptjs": "^2.4.2",

"@types/chai": "^4.2.7",

"@types/chai-as-promised": "^7.1.3",

"@types/express": "^4.17.2",

"@types/mocha": "^5.2.7",

"@types/mongodb": "^3.3.14",

"@types/node": "^13.1.4",

"@types/node-fetch": "^2.5.5",

"chai": "^4.2.0",

"mocha": "^7.1.0",

"node-fetch": "^2.6.1",

"nodemon": "^2.0.2",

"ts-node": "^8.6.2"

}

Rest endpoints has been tested by. Creating a before where we have a timeout since we’re using mongoDB, use a test\_port if our application is running, skip authentication and point to a test database:

this.timeout(Number(process.env["MOCHA\_TIMEOUT"]));

process.env["PORT"] = TEST\_PORT;

process.env["SKIP\_AUTHENTICATION"] = "1";

process.env["DB\_NAME"] = "semester\_case\_test"

Then we do a beforeEach that nukes the database

await usersCollection.deleteMany({})

And then we add what we want again.  
Once that is done we can create tests where we can “make” a config where we can add headers and specific methods if we aren’t just sending a get request without any headers.

const config = {

method: 'POST',

headers: {

'Accept': 'application/json',

'Content-Type': 'application/json'

},

body: JSON.stringify(newPosition)

}

const result = await fetch(`${URL}/gameapi/nearbyplayers`, config).then(r => r.json());

**Explain, using relevant examples, how to test JavaScript/Typescript Backend Code, relevant packages (Mocha, Chai etc.) and how to test asynchronous code.**  
If we want to run a specific TypeScript fine we can do:

ts-node -r dotenv/config ./src/PATH\_TO\_FILE

Else with the script we have we can use npm test / npm testDebug. And then we can also use

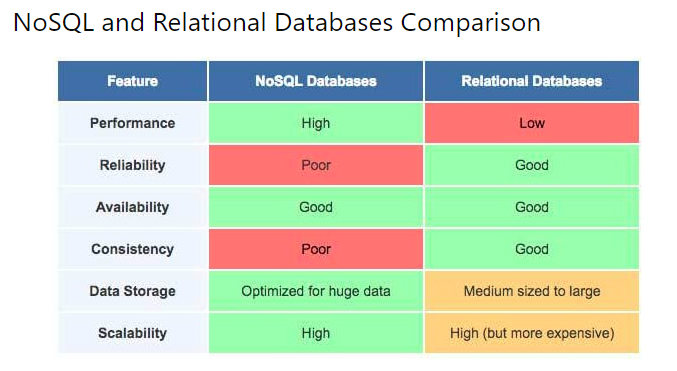
const debug = require("debug")("user-endpoint-test");

For testing asynchronous code we can use async await /.then hell and also use

chai.use(chaiAsPromised);

#### NoSQL and MongoDB

***Explain*, generally, what is meant by a NoSQL database.**  
NoSQL databases (aka "not only SQL") are non tabular, and store data differently than relational tables. NoSQL databases come in a variety of types based on their data model. The main types are document, key-value, wide-column, and graph. They provide flexible schemas and scale easily with large amounts of data and high user loads.  
<https://www.mongodb.com/nosql-explained>   
<https://www.youtube.com/watch?v=BgQFJ_UNIgw>

***Explain* Pros & Cons in using a NoSQL database like MongoDB as your data store, compared to a traditional Relational SQL Database like MySQL.**  
  
Pros:  
- Schema-less  
- No (or very few) relations. We usually would just add some fields from a user.  
- Data is typically merged/nested in a few collections  
- Both horizontal and vertical scaling is possible.  
- Great performance for mass (simple) read & write requests  
  
Cons:   
- Not recommended for very big applications/programs.  
- Not recommended for robust systems that will never change schema.  
- Lack of high performance concurrency: In many cases, Graph Databases provide multiple reader and single - writer type of transactions, which hinders their concurrency and performance as a consequence, somewhat limiting the threaded parallelism.  
- Lack of standard languages: The lack of a well established and standard declarative language is being a problem nowadays. Neo4j is proposing Cypher and Oracle is working on a language.  
  
  
<https://youtu.be/ZS_kXvOeQ5Y?t=1140>   
<https://www.youtube.com/watch?v=ZS_kXvOeQ5Y&t=20s>   
<https://dev.to/lmolivera/everything-you-need-to-know-about-nosql-databases-3o3h>

***Explain* about indexes in MongoDB, how to create them, and *demonstrate* how you have used them.**  
  
<https://docs.mongodb.com/manual/indexes/>   
Example of creating index:

//1) Create expiresAfterSeconds index on lastUpdated

positionCollection.createIndex({ lastUpdated: 1 }, { expireAfterSeconds: EXPIRES\_AFTER });

//2) Create 2dsphere index on location

positionCollection.createIndex({ location: '2dsphere' });

***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.***<https://docs.mongodb.com/manual/core/index-ttl/>   
<https://docs.mongodb.com/manual/core/index-unique/>   
See readyForDay6/src/utils/geoUtils.ts for TTL  
See readyForDay6/test/gameFacadeTest.ts for 2dsphere. Line 51-60

***Demonstrate*, using a REST-API *designed by you*, how to perform all CRUD operations on a MongoDB**  
See readyForDay6/src/routes/gameAPI.ts line 36-52 ->  
readyForDay6/src/facades/gameFacade.ts line 110-129

***Explain*, using a *relevant example*, a full JavaScript backend including relevant test cases to test the REST-API (not on the production database)**  
Already explained in “Explain, using relevant examples, your strategy for implementing a REST-API with Node/Express + TypeScript and demonstrate how you have tested the API.”

Demonstrate, using your own code-samples, decisions you have made regarding → normalization vs denormalization

#### Geo-location and Geojson

**Explain and demonstrate basic Geo-JSON, involving as a minimum, Points and Polygons**  
  
<https://docs.mongodb.com/manual/reference/operator/query/polygon/>   
<https://docs.mongodb.com/manual/reference/geojson/#point>

**Explain and demonstrate ways to create Geo-JSON test data**  
  
<http://geojson.io/#map=14/55.7904/12.5632>

**Explain the typical order of longitude and latitude used by Server-Side APIs and Client-Side APIs**  
  
On the server side we have it LAN and then LON.  
But on Client-Side we display it as LON and then LAN.

**Explain and demonstrate a REST API that implements geo-features, using a relevant geo-library and plain JavaScript**  
  
See readyForDay6/src/routes/gameAPI.ts line 36-52 ->  
readyForDay6/src/facades/gameFacade.ts line 110-129

**Explain and demonstrate a REST API that implements geo-features, using Mongodb’s geospatial queries and indexes.**  
  
<https://docs.mongodb.com/manual/geospatial-queries/>   
Same as above question.

**Explain and demonstrate how you have tested the gameFacade and gameAPI for the game-related parts of the period exercises**  
  
Already explained in “Explain, using relevant examples, your strategy for implementing a REST-API with Node/Express + TypeScript and demonstrate how you have tested the API.”

### This will come in period-3

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

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