CAB230 – Web Computing

server-side documentation

n10234756

2019

Contents

[Intro 2](#_Toc10410800)

[Technical, testing, limitations & security 2](#_Toc10410801)

[Module Installation Guide 3](#_Toc10410802)

# Intro

As was observed during 9-11 tutorial by the tutor, the assessment (and cloud-based backups) had corrupted, thus an attempt was made to recreate the server-side.

All unauthenticated routes and some of the authenticated routed were implemented.

The unauthenticated routes Offences, Areas, Ages, Genders and Years.

The authenticated routes for Login and Register were also implemented.

A certificate and https were also implemented. The swagger doc was set to be the root of the server.

# Technical, testing, limitations & security

The server-side component uses express as the base of its function. The express application uses various instances of middleware. This middleware acts as a chain of complementary functions which can be used to establish a connect directly to a request object without additional parameters and unneeded information.

The server-side implementation implements the use of knex to act as middleware between queries and the database. This allows for simpler queries to be used for the same result and with a reduced chance of an integrity breach. The morgan logger is used to log events that occur on the server, it is set to use the common trigger as it provides all the relevant information. The cookie parser is used to parse cookies and place the cookie information on a req object in the middleware. It also has the ability to decrypt signed cookies if the secret is known.

Knex is implemented through the knexfile.js file and is assigned to “req.db” which allows the database to be accessed by the middleware. A portion of exception handling is dealt with by the native express handlers.

Endpoints are handled in the routes folder under index.js. The swaggerUI, although created in the app.js is handles and set as the root through the index.js file using the express router.

Unauthorised endpoints can be accessed by their respective names to the cab230.hackhouse.sh variants. Successful call responses have been mimicked. However, the entry "Torres Strait Island Regional Council" (as is shown on cab230.hackhouse) is represented as “Torres Strait Island Regional Counci” from the provided SQL dump. Errors returned by these unauthorised endpoints only occur if the database is unavailable.

The Authorised end points are more complicated than unauthorised endpoints, passing knex middleware to check database entries against user input entries. The register endpoint performs an initial check against user input, if the email already exists, an error status 400 is returned, if the email and password do not exist, a new entry is created in the database. The user’s password is salted and hashed, to protect user accounts if an outside entity is able to view the content.

The login endpoint is similar to the register endpoint, first performing a check as to whether or not the user account exists, however unlike the register endpoint, a non-existent account is rejected. If the user has entered an incorrect password, an error 401 is returned, however if the correct username and password is used, a status 200 is returned, alongside an authentication token. The password is checked by resalting and hashing the input password and comparing it against the stored hash. The hash and webtoken are made available through “bcryptjs” and “jsonwebtoken” respectively.

Due to the limited amount of time and testing required to implement the search endpoint, this was not possible. As such the attempts were removed to prevent errors and allow the rest of the server to properly function.

This server is TLS protected in compliance with the X.509 standard. A cnf file is used to create the SSL certificate. This certificate is valid for 365 days from its creation and protected by sha256. An authentication key is made using RSA. This certificate is then applied to the server instance using the private key and certificate permissions, which are stores as access credentials. A https instance of the server is then created using these credentials and the channel begins to listen for traffic. Http instances are closed and redirected to the https server to protect information. Helmet is used to ensure that a HSTS header is present, this is responsible for ensuring a https connection is used.

# Module Installation Guide

NPM modules can be installed through the use of a command terminal either locally or globally. Many of the authentication modules must be installed prior to use as they are not standard libraries.

The user must first navigate to their express directory if they wish to install these modules locally. The http-server and http-errors modules must be installed as these are a basis for the client, however the https and https-server modules are then installed as these then take precedence.

Knex is a middleware used to instantiate instances of data objects directly and replaces mysql queries, however mysql must also be installed to instantiate knex for mysql.

The swagger docs can only be successfully loaded when the swagger and swagger-ui-express modules are installed, as such are required.

There are two modules used for authentication and protection, the first is the “jsonwebtoken” module, which allows a JWT token and its bearer to be instantiated, this is to allow a user to be authenticated. However, the bcryptjs module is used to salt and hash user passwords to prevent plaintext passwords being saved, preventing security issues if the passwords were to be seen.

The SSL certificate requires a .cnf file containing authentication information. When the .cnf file has been completed, the command openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout cert.key -out cert.pem -config req.cnf -sha256 can be used, this will generate a certificate that is X.509 compliant. A key is generated using RSA, this cert.key is needed alongside the cert.pem file to access and authenticate the certificate. Sha256 is used as the signature algorithm. Once the files have been created and instantiated a https connection will be available. However, this https is broken, and a user must first accept the certificate through their browser. This is often available in the security overview of a webpage; the certificate must be accepted. A valid https connecting will be available the next time the server is run.

Below is a full list of dependencies available from the package.JSON

{

"name": "expServer",

"version": "1.0.0",

"private": true,

"scripts": {

"start": "node ./bin/www"

},

"dependencies": {

"bcryptjs": "^2.4.3",

"cookie-parser": "~1.4.3",

"debug": "~2.6.9",

"express": "~4.16.0",

"helmet": "^3.18.0",

"http-errors": "~1.6.2",

"http-server": "^0.11.1",

"https": "^1.0.0",

"https-server": "^0.1.2",

"jade": "~1.11.0",

"jsonwebtoken": "^8.5.1",

"knex": "^0.17.2",

"morgan": "~1.9.0",

"mysql": "^2.17.1",

"swagger": "^0.7.5",

"swagger-ui-express": "^4.0.2",

"types": "^0.1.1"

}

}