In this template, all italic text should be removed and replaced with your own text (which should not be italic); the italic text is just a placeholder letting you know what to write there.

On the cover page, change to your own project name, your own name and your own JU email address.

You have a lot of freedom when it comes to writing this report. You do not have to use any part of this template, but the report you write should in the end somehow (in a good way) provide the same information as indicated in this template. Most students trying to do it in their own way usually fail, so if you try that, be sure to know what you are doing!

Use proper sentences, paragraphs, lists, tables, figures, etc. The more figures you use, the less text you need to write, so use many of them.

This page should be removed.

Find more tips on writing a report at https://peppel-g.github.io/course-material/lectures/report-writing/.

(it's amazing (scary) how many students that delete this text before they have read it...)

Table of Contents

Introduction

Introduce your project work. Write text that ***indirectly*** *(look up what indirectly means if you don't know it) answers questions like:*

- *Why does the project exist?*
- *What is the project about?*
- *Who are involved in the project?*
- *What will the project result in?*
- *Who are interested in the outcome of the project?*
- *How will the outcome of the project be used?*

Have you ever been hungry spending 40 minutes with your friends deciding which place to eat out at? I know I have and that's exactly why I created "foo revoo", a place where you can read great reviews about great restaurants across Sweden. By using foo revoo you can quickly and efficiently find new and exciting restaurants, making you the savior of hunger for both your family and your friends.

// QUESTION : Should the introduction section be longer?

It is good if you can add a UML use case diagrams that visualizes how the end users will use the website, such as the one at https://www.conceptdraw.com/resources/images/solutions-screens/diagramming/UML_Use_Case_Diagram.png.

After having read this chapter, those that have never heard of the project before should a have a good understanding of what it is about. If they would like to learn how it has been implemented, they just need to continue reading the rest of the report.

Architecture

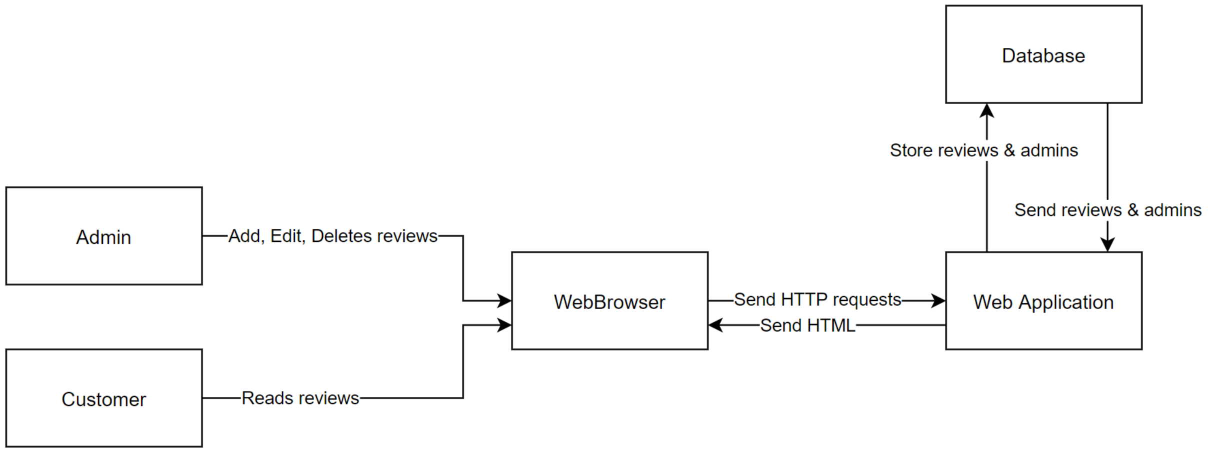
Give an overview of the components the website consists of (web application, database, web browsers, end-users, etc.). Visualize this in a figure and show how the different components make use of each other.

After having read this chapter, the reader should have a broad (but shallow) understanding of the website's internal structure.

Database : The primary function of a database is to store data in an application. In Foo revoo admins will for an example create a restaurant reviews and then store said review in the database. A customer can then at any point in time access the website and get the review from the database for reading.

Web Application : The web applications main responsibillity is to dynamically create HTML code using data from the database and then send the created HTML code to the user's web browser. However the web application also handles things such as routing, meaning that different data should be sent depending on what URL is used. “Foo-revoo.com/contact” should send the contact page whilst “Foo-revoo.com/about” should send the about page.

Admin : An admin authenticates himself by logging in, the admin has the authority to create, update and delete reviews. The admin can also create, update and delete other admins if the admin is an owner.



Database

Describe the database and the resources on the website in detail. What attributes do they consist of? How are they related? Where are they stored (in files on the hard-drive? In a relational database? Etc.). Visualize the resource in an ER diagram, such as the one found at https://www.lucidchart.com/pages/templates/er-diagram/er-diagram-example-template.

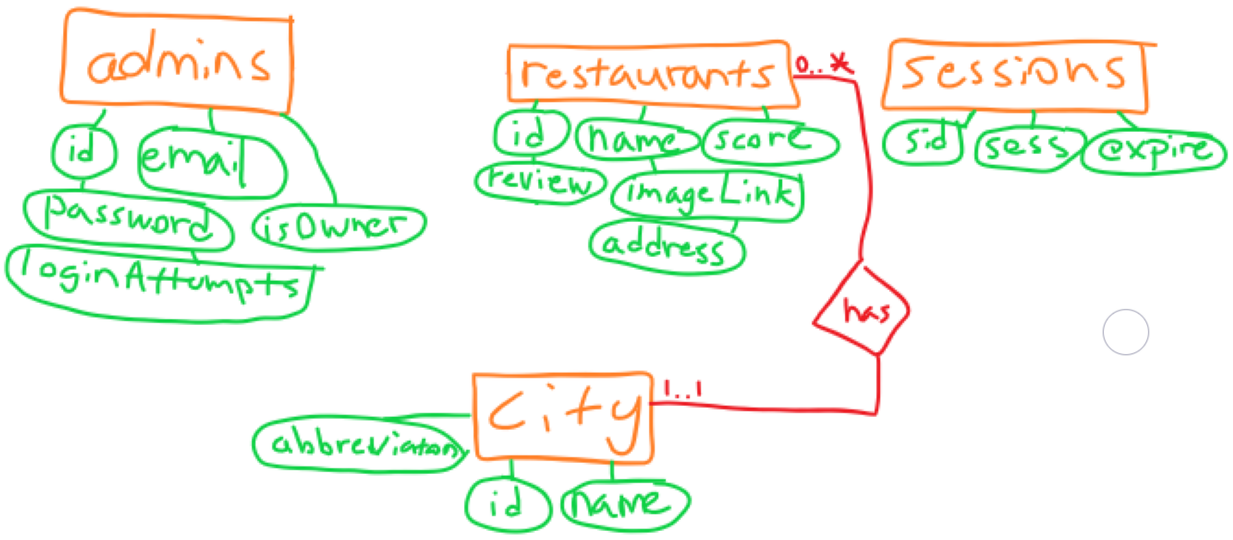
After having read this chapter, the reader should understand how the data (the resources) in the website is stored.

"If the reader is a new programmer that should start working on the website, she should now know what she needs to do if she wants to change the resources or add more type of resources (e.g. know how to add a new table to the database with a relation to an existing table in the database)." 1?

1? QUESTION : Should you make a step by step tutorial on how to operate SQLite? Like how to create tables and connect them with foreign keys?

Since the project is relatively small I've opted for SQLite which basically is a more compact version of SQL, this offers clear advantages. For one SQLite doesn't need to setup an independent server for the database. Not only is this time consuming but you will not be able to access this independent server remotely if you have hosted the server locally. SQLite is simply saved as a small file inside the project folder which can easily be accessed remotely via GitHub since the project folder is tracked with Git.

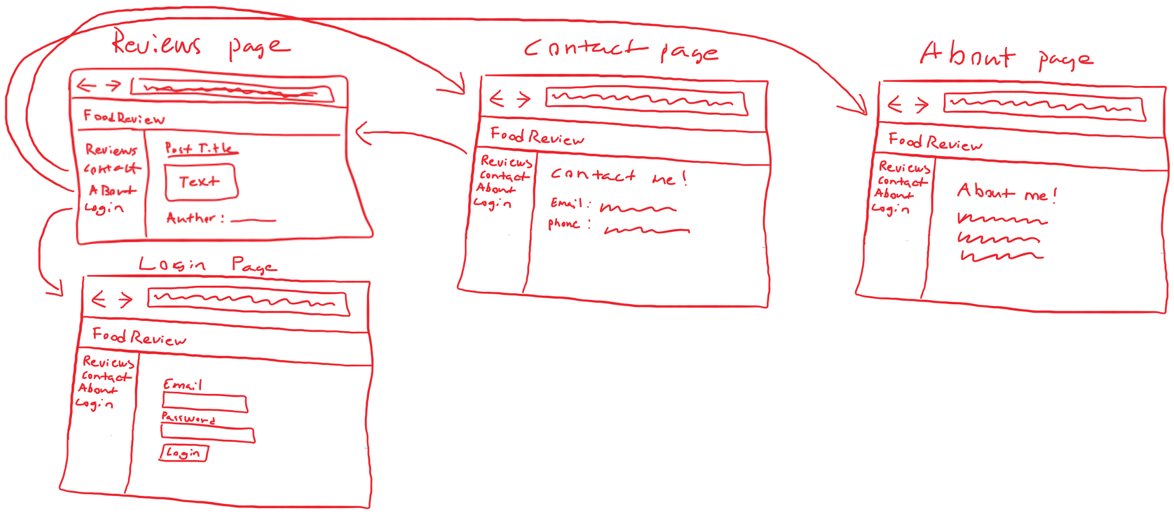
The database only contains one direct relation, which is between restaurants and cities. Removing a city will cascade onto all restaurants containing that specific cityId. Meaning that removing the city "Jönköping" will remove all restaurants that have the cityId which corresponds to "Jönköping".

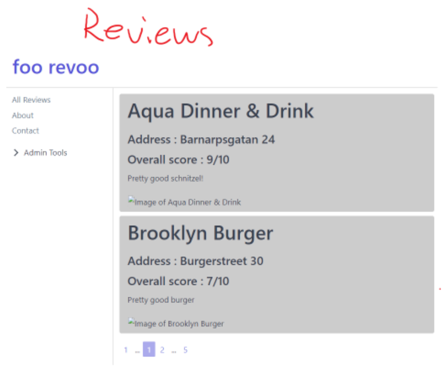


Graphical User Interface

After having read this chapter, the reader should understand how the end users will be able to accomplish their goals through the graphical user interface.

Users that has not logged in as admins only have access to the reviews, contact, about and login page. As seen on the figures below there is a navigation bar on the left side of the website to access the forementioned pages.





Aqua Dinner & Drink

Address : Barnarpsgatan 24

Overall score : 9/10

Pretty good schnitzel!

Image of Aqua Dinner & Drink

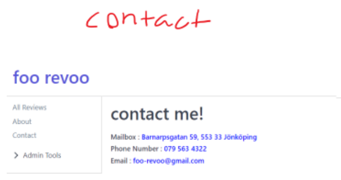
Brooklyn Burger

Address : Burgerstreet 30

Overall score : 7/10

Pretty good burger

Image of Brooklyn Burger

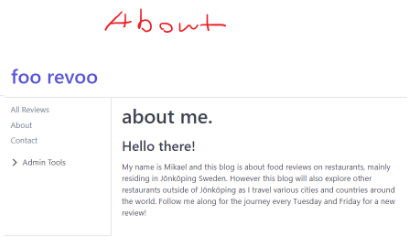


contact me!

Mailbox : Barnarpsgatan 59, 553 33 Jönköping

Phone Number : 079 563 4322

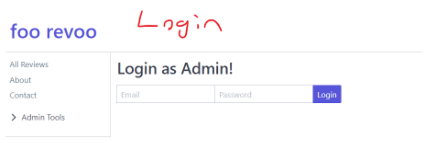
Email : foo-revoo@gmail.com



about me.

Hello there!

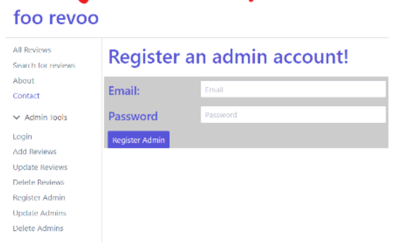
My name is Mikael and this blog is about food reviews on restaurants, mainly residing in Jönköping Sweden. However this blog will also explore other restaurants outside of Jönköping as I travel various cities and countries around the world. Follow me along for the journey every Tuesday and Friday for a new review!



Login as Admin!

Once a user has authenticated himself as an admin he has the ability to create, edit and delete restaurant reviews. If the user has authenticated himself as an owner, he also has the ability to create, edit and delete other admins.

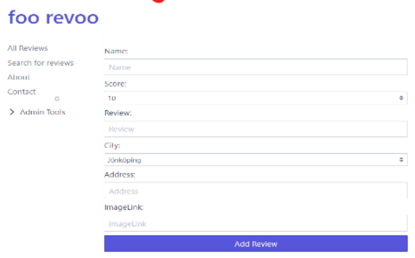
Register Admin



Register an admin account!

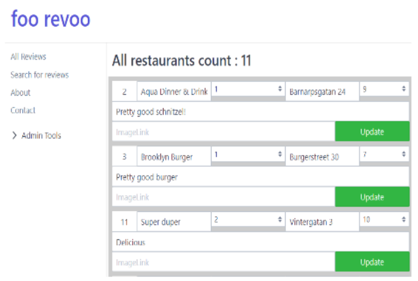
Email: Email
Password Password

Adding Review



Name: Name
Score: 10
Review: Review
City: Jönköping
Address: Address
ImageLink: ImageLink

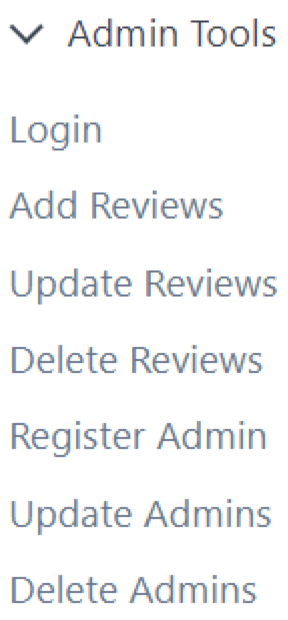
Updating Reviews



All restaurants count : 11

2	Aqua Dinner & Drink	1	Barnarpsgatan 24	9
Pretty good schnitzel!				
ImageLink				Update
3	Brooklyn Burger	1	Burgerstreet 30	7
Pretty good burger				
ImageLink				Update
11	Super duper	2	Vintergatan 3	10
Delicious				
ImageLink				Update

Which he can navigate through with the help of the “Admin Tools” navigation tab.



Web Application

Describe implementation details of the web application. Which language have you used? Which framework have you used? Which libraries/packages have you used, and for what purpose? Has all code been written in one file? Or have you somehow structured it in multiple files? Are you using some design patterns (e.g. MVC)? Are you using middlewares? Etc...

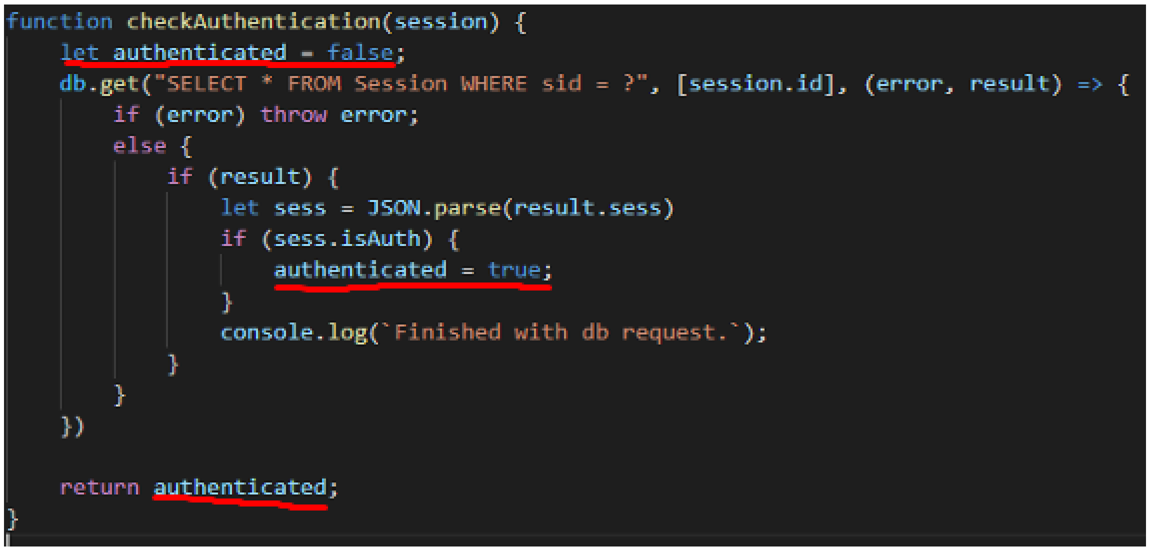
The main language of the project is of course Javascript, as it is a web application. The application is split up into two main parts, the front-end and the back-end. The front-end is handled by HandlebarsJS whose responsibility is to dynamically generate HTML code. NodeJS and Express is used in conjuncture for the back-end. Express which is probably the main package of the project sets up the server and serves primarily the generated HTML by HandlebarsJS, it also handles the POST requests from users and fetches the data from our database for HandlebarsJS to use.

All the Javascript code for the project is written in 'app.js', as most things this has both it's upsides and downsides. It's easier and quicker to set up the project, there's no need to deal with exporting and importing files/variables inside the project. The major downside however is that if the project starts growing you're going to need to eventually split up the project into smaller components. Since navigating a Javascript file containing 4000 lines of code is going to be time draining.

As far as design patterns go, there is none. App.js simply acts as the API for the application, bridging the database with the front-end. The project uses the following middlewares session, csrfProtection, express.urlencoded and the last middleware catches faulty routes and routes them to an error page. Both session and csrfProtection are discussed in Security down below.

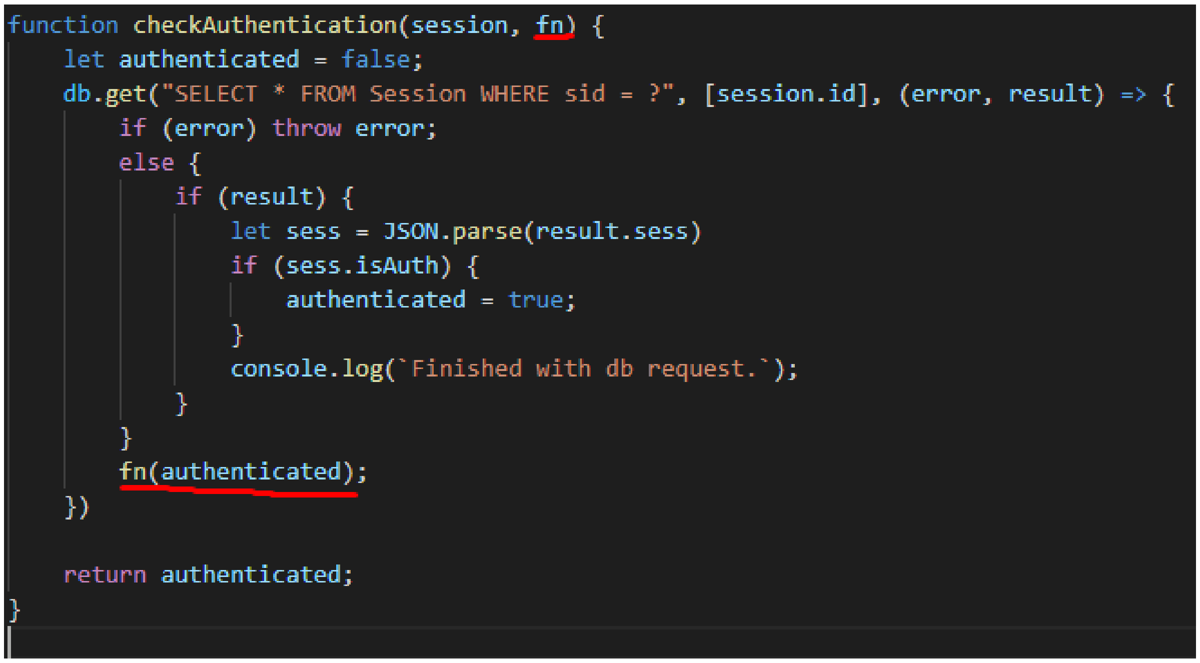
better-sqlite3 vs sqlite3

I started out using the “sqlite3” package, however the flaw with sqlite3 was that instead of using Promises for asynchronisity, it used callbacks. At first this wasn't a big problem, until I needed to return values from inside the callback to the parent function.

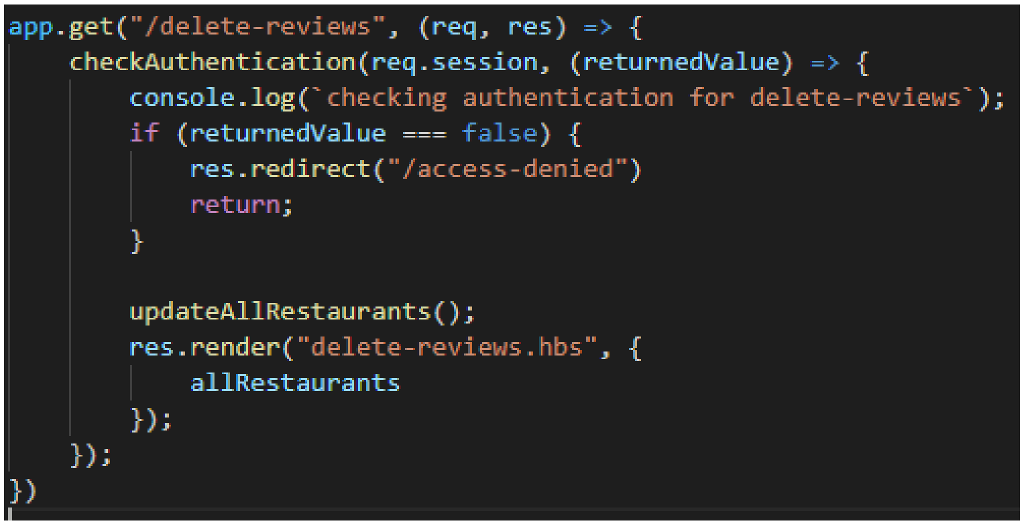


```
function checkAuthentication(session) {
    let authenticated = false;
    db.get("SELECT * FROM Session WHERE sid = ?", [session.id], (error, result) => {
        if (error) throw error;
        else {
            if (result) {
                let sess = JSON.parse(result.sess)
                if (sess.isAuth) {
                    authenticated = true;
                }
                console.log(`Finished with db request.`);
            }
        }
    })

    return authenticated;
}
```

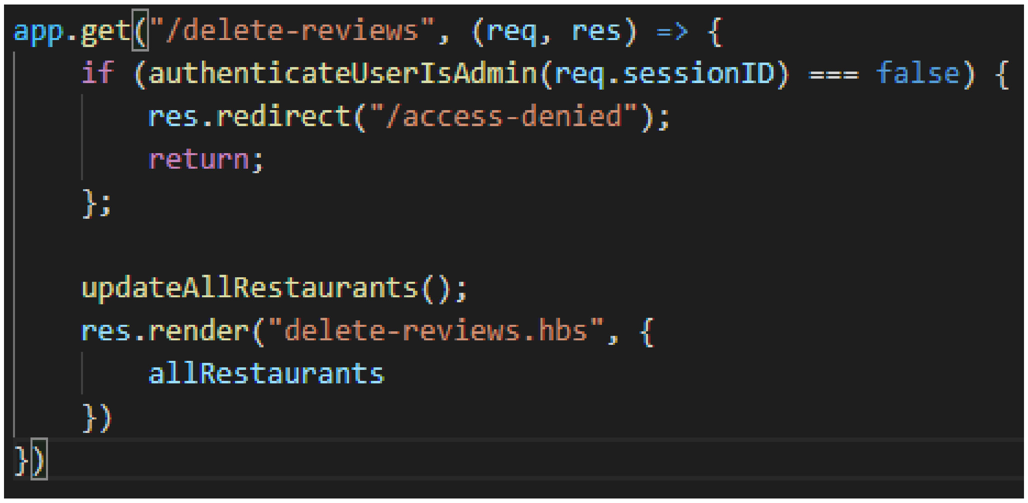
Here authenticated gets returned as false before the callback of db.get() has the oppurtunity to run. You can't await db.get() since it's not asynchronous and you can't return the value to the checkAuthentication function inside the callback. A solution to this problem would be to add ***another*** callback to the function checkAuthentication.



```
function checkAuthentication(session, fn) {
    let authenticated = false;
    db.get("SELECT * FROM Session WHERE sid = ?", [session.id], (error, result) => {
        if (error) throw error;
        else {
            if (result) {
                let sess = JSON.parse(result.sess)
                if (sess.isAuth) {
                    authenticated = true;
                }
                console.log(`Finished with db request.`);
            }
        }
        fn(authenticated);
    })

    return authenticated;
}
```



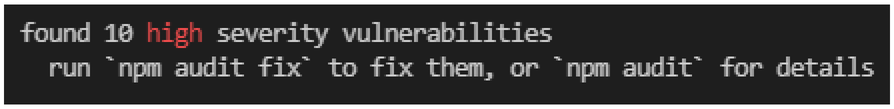
```
app.get("/delete-reviews", (req, res) => {
    checkAuthentication(req.session, (returnedValue) => {
        console.log(`checking authentication for delete-reviews`);
        if (returnedValue === false) {
            res.redirect("/access-denied")
            return;
        }

        updateAllRestaurants();
        res.render("delete-reviews.hbs", {
            allRestaurants
        });
    });
})
```



```
app.get("/delete-reviews", (req, res) => {
    if (authenticateUserIsAdmin(req.sessionID) === false) {
        res.redirect("/access-denied");
        return;
    };

    updateAllRestaurants();
    res.render("delete-reviews.hbs", {
        allRestaurants
    })
})
```

However if you compare the two code blocks, the one with callbacks is just plain uglier. Later on in the project I decided to check if the user was an owner of the website with a “authenticateUserIsOwner” function. Had I been using the sqlite3 package, this would have resulted in yet ***another*** callback nesting, making it even more hideous than It already is.

Additionally, sqlite3 together with connect-sqlite3 came with:



```
found 10 high severity vulnerabilities
  run `npm audit fix` to fix them, or `npm audit` for details
```

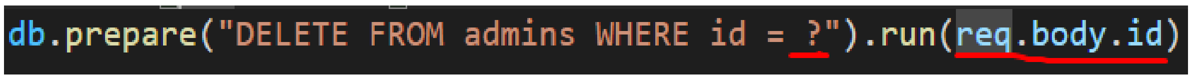
Switching from sqlite3 to better-sqlite3 removed the callback hell and removed package vulnerabillities, which is why I decided to refactor the project to use better-sqlite3 instead.

Security

When it came to security I decided to use “bcryptjs” to create hashes which I would store as admins passwords inside the database. This is mainly done to prevent passwords leaks in the event that the database is comprimised.

To defend against XSS attacks in Handlebarsjs you simply have to use the standard double brackets “{{}}” instead of triple brackets “{{{}}}”.

SQL-injections are defended against by using “?” instead of injecting variables into the string.



```
db.prepare("DELETE FROM admins WHERE id = ?").run(req.body.id)
```

Cross-Site Request Forgery is defended against by using the npm package “csurf”, which in turn will create and send a csrfToken when a user GETs a page containg a form. When the user POSTs the form, they will send the csrfToken, If the POSTs csrfToken matches the previously generated csrfToken we know that the user sent the POST request using our form and not via a CSRF-attack.

To authenticate the admins I've used sessions to keep admins logged in once they've logged into their account. A new session is created when a user succesfully logs into their admin account. The session gets the property “.isAuth = true” added onto the session object. The session then gets added to the database inside the sessions table. The user then gets a cookie containing the sessionId, which will reference the session inside the table sessions.

Try to use many figures. They are much easier to read than a wall of text. Use text to explain details that can't easily be visualized in a figure.

After having read this chapter, the reader should have a very good understanding of how the web application has been implemented. If the reader is a programmer who should start working on the web application, she should now know where to start when she should implement new features to the web application.