Egg Presentation

Total of 20 minutes (at most) per group

Spend the first 5 minutes to (in no specific order)

- **Introduce what your project is, including what technologies you use in the project and why you choose them**
  We decided to work on a website egg store where users can buy, review, order eggs.

  Next.js

  Because it is a fullstack framework which we have experience of using in two course projects previously. We mainly chose it because allows us to create the backend but also also build a frontend for the website.

  tRPC

  Is a library for creating typesafe TypeScript based API's supporting virtually any TypeScript runtime like Node.js, Bun, Deno, etc. We chose it to be able to write the API with full type safety between the server and the website.

  Postgres

  Open source and free SQL relational database which we have used before.

  Drizzle ORM

  An orm to able to write type safe queries when working with a SQL database, makes it easier to write correct queries. And also supports prepared queries to protect against SQL injection and to give ~30% better performance. We chose it over Prisma since we've used Prisma before and wanted to try Drizzle as it's supposed to have better performance and have better support for serverless environments.

  Argon2

  For hashing passwords we use Argon2 and we chose it because it's supposed to be one of the most secure password hashing algorithms.

  RabbitMQ

  For the microservices mainly used for queuing emails to be sent

GO-email
We used GO-email to be able to send emails using SMTP in Go. GO-email is flexible and allows for high throughput email flows without any downtime or hiccups.

JWT
We use JWT and refresh tokens with HTTP only cookies to authorize users so that they only have to login once and the password only has to be sent once. With an authorized JWT token you can use it for all services that can verify the token. And we don't have to send the password all the time like in HTTP basic authentication. The refresh tokens are stored in the database and are used to allow the user to get a new JWT access token when the old one has expired. This way they don't have to go through the login step every 15 minutes when the JWT expires.

- **Briefly explain the database design, how it evolved over the project timeline, and what changes you made from the first design you came up with**

  The user table stores information about each user, for example name, password, role, email.

  The refresh token table has an expiry column to tell when the token has run out, it is used to issue users with new JWT access tokens.

  We a product table with product name, price and stock. Then we have a review table to allow users to create reviews of products.

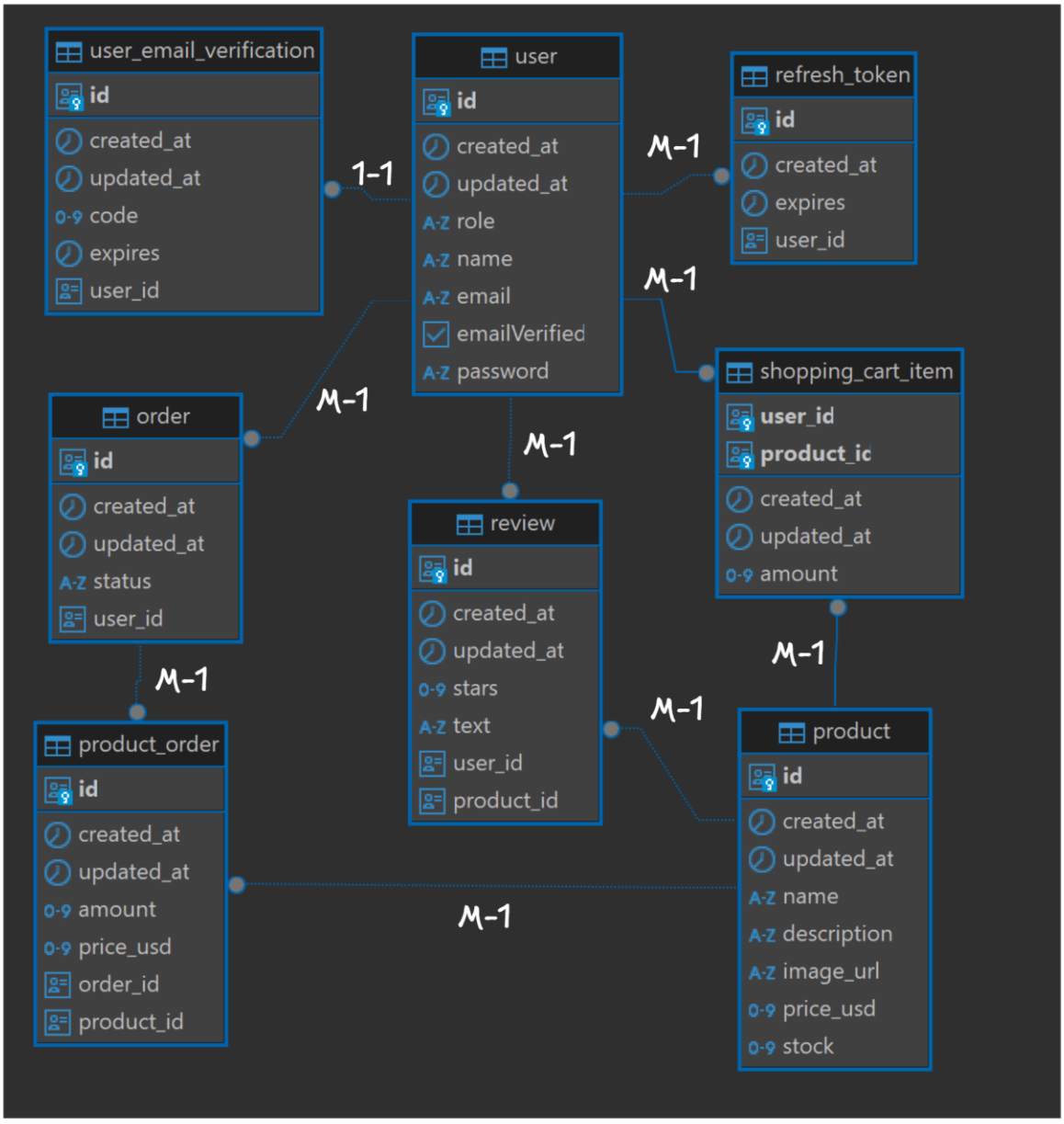
  The user can add products to their shopping cart by creating shopping cart items which collectively represent the users shopping cart.

  When the users have filled their cart with all the eggs, they can press purchase to create an order row in the order table where each order has multiple product orders which stores the products they've ordered, for which price and what amount. This way if the price changes later the order price remains the same. The order has a status for pending, processing, shipped and delivered.

  We use prepared SQL statements as a countermeasure against SQL injection attacks. This protects users' data against malicious actors, Prepared statements are also faster

to run. We also hash the user's password using argon2 hashing; this means the database never stores the actual password in case the database gets leaked.

We used to have a table with shopping carts but realised that you rarely want more than one shopping cart per user, so we changed only having a shopping_cart_item table where each item is linked to a user.



- **Explain who in the group did what and what your target grade**

Because the project is fairly small consisting of only two services we decided to work together in a VSCode live share session. This way we wouldn't have to deal with commit merges every 10 mintes like we've dealt with before.
We would often have one person making changes and another testing to make sure it works as intended.

Spend the next 10 minutes demonstrating the project

- Make sure to demo in such a way that satisfies/showcases the needed requirements

(run drizzle studio in background to be able to show things like email verified)
(create another admin account to be able to show changing user roles)

- Create account
  - Email john@doe.com, password: johndoe
  - Open email at localhost:1080
  - Enter 2FA code and show that the user is verified

- Log out
- Log in with "john@doe.com", password "johndoe", and show 2FA again

- Add 3 different products to shopping cart
- Click purchase to create order
- Make a review of a product
- Delete the review
- Create it again why not

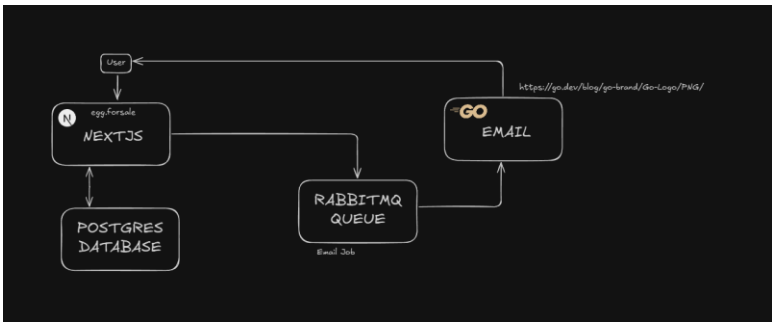
(change user role from user to admin)

- Change order status
- Create a new product as admin
- Edit the same product with new price etc

(change user role from admin to SUPER ADMIN)

- Change the role of the made admin from before
- Delete that admin

- Show that we are using microservice architecture
  - Show code / Diagram of architecture



- Run the test (✅)

Show code where the RBAC is done (procedures file)

The following 5 minutes is for Q and A

- Everyone on the team must present and answer the questions

Things to (not) do

- Collect Data Without Consent
- Misrepresent Data Usage
- Sell and Share Data Improperly
- Manipulate Data and Misrepresent it
- Neglect Data Security
- Retain Data Longer Than Necessary
- Spy or Monitor Without Justification
- Fail to Report Data Breaches
- Exploit personal data to target vulnerable groups
- Breach Confidentiality Agreements