CodeClan Final Project Backend Presentation

The backend consists of the Java based Framework, Spring and the SQL database, H2. The choice of database was because we've used it before and it was created with Java in mind. Given more time we'd have opted for a NoSQL database such as MongoDB due to the flexibility and scalability it offers. We're also using Java 8 rather than more advanced versions since its fairly stable.

The backend and frontend technology aren't heavily reliant on one another so you could in theory replace Spring with Express and H2 with MySQL and have few issues. The backend's prime role is just to provide JSON taken from a database for the frontend to manipulate as well as store information sent to it. The backend has full CRUD functionality, as for the Update part we went with PUT over PATCH just because it was simpler to implement. The GETs vary from GET ALL to GET one entry with a specific id and various other GET filters mostly returning JSON files. A rare few actually return Longs such as a count of how many reservations a specific customer has made.

It is also possible to delete an entry via a specific id or delete everything in a specific table if you need to clear out its content without resorting to a full drop and recreation of every single database table.

We're using the these dependencies "Spring Boot DevTools" to assist us with development and fast application restarts, "Spring Web" to help us build our RESTful controllers, "Spring Data JPA" to help us interact with the database and as previously mentioned "H2 Database". IntelliJ IDEA was used to write the Java.It's a superb IDE capable of rapidly generating the getters, setters and constructors in our models for the Object Relational Mapping with our database. It also generates some beautiful Java Documentation.

[Show Reservation Controller Slide.png]

[Show Models Slide.png]

We have 4 database tables, Venues, Venue Tables, Customers and Reservations. All of these tables have ids in the form of Longs which are used as primary keys.

Venues just have their venue name.

Venue Tables holds a cover (the amount of people you can seat at the table) and an id relating to which Venue it belongs to. We have to refer to them as venue tables so they aren't confused with regular database tables.

For Customers we have an email, first name, last name and phone. This is the customer that calls in to make the reservations & can be contacted if there's a change of plans rather than a list of every single person attending.

The Reservations table has a start and end date using local date-time, the size of the party attending and each reservation must contain a customer and venue table. An optional value that can be entered is called "reservationNotes" where you can add information related to the party such as allergic to "nuts" or "sing happy birthday".

The Java packages consist of "components" that holds our data loader, "models" and "controllers" whose names are self explanatory in an MVC context and "repositories" whose files are interfaces that utilise the JPA Repository so we can interact with the database. Since both our client and our servers were running locally, we had to tackle CORs errors including, in our controller classes, @CrossOrigin(origins = "http://localhost:3000")

One of the things I disliked about apis used during some of our tutorials was the limited amount of filtering you could make which inspired me to experiment with JPA so our app could be

introduced to new frontends, vanilla javascript or just html urls and still have powerful filtering capabilities. For example in there are GET requests that allow you to retrieve customers by their first name, last name, the start of their last name, some letters the last name contains for those that find spelling tricky or view all customers in descending order of their last name whilst ignoring case. There are 58 unit tests related to our tables, from making sure the setters in the models are working to checking that entries get deleted and the correct ones are getting retrieved from the database.