Finalization Phase: Abstract of the Family Shopping List project

The goal of the Family Shopping List (FSL) project is to enable families to better organize their shopping needs in a collaborative way. Shared shopping lists among family members, shared inventory items per stores, and facilitating the shopping process while in a real store are the main objectives of this project. The high-level design of the application consists of a web application using HTML, CSS, and JS/TS, which interacts with an application server using Node.js, HTTP, and JavaScript. The server itself uses a relational database, MySQL, as a persistent storage system.

The chosen technologies focus on a web application that runs in a web browser as single-page application (SPA) in a responsive way with a mobile-first design in mind using an Ubuntu Linux distribution as target platform. The mobile-first approach is a firm requirement considering the goal to support real-world shopping in a brick-and-mortar store. The Angular 17 framework (Google, n.d.), which supports TypeScript, bootstrap CSS (Bootstrap team, n.d.-a), bootstrap icons (Bootstrap team, n.d.-b), HTML, RxJS, etc., and its organization of component-oriented development approach, provides a good environment to encapsulate and reuse functionality. For example, the inventory-edit component is used in different contexts without replicating the code. By calling the component from the Inventory as well as from the Shopping List-Add component, it makes it easier for developers to follow the process flow and it avoids implementing the same functionality again.

The server side of the application uses Node.js as an HTTP server (Node.js, n.d.) leveraging additional components and packages, for example, Express.js (Zanini, 2023), to handle the web applications requests. Using MarketSplash Team (2023) analysis on how Node.js and Angular can work together, provides a good starting point to create a web application scaffolding to get a web application and its application server working. Installing Node.js (Boucheron, n.d.) and the Google Chrome Web Browser (Linuxize.com, 2020) on Ubuntu can be challenging to make this initial state work. To implement the server's HTTP functionality, the documentation of the HTTP API (Node.js, n.d.) can be very helpful.

After this initial foundation is laid, the database, MySQL, can be used with Node.js using Sequelize ORM (Object-relational Mapper). Sequelize facilitates the database model creation, the seeding of test data, and the implementation of business logic with the database. The underlying ERM (Entity Relation Model) needs to be designed and coded using Sequelize within the Node.js server as exemplified in Dedigama & Lee (n.d.). Once this groundwork exists, an end-to-end test is possible by using a web browser that runs the web app, making calls to the Node.js server, which then communicates with the MySQL database via Sequelize.

However, HTTP requests might not reach the Node.js server due to security reasons and are blocked by CORS (Cross-Origin Resource Sharing) policy (Modzilla, 2023). Kumar (2020) also describes this in the context of Node.js and how to add a CORS middleware to the server. Hobbs (n.d.) demonstrates how to use CORS in the context of Node.js and Express.js, which is also used for this project and as default only requests from `http://localhost:8089/` are allowed in this implementation[1].

Now that the CORS challenge is resolved, the data model becomes the focus. Identifying real-world entities and how they are related is the next step in the development process. Durante (n.d.) provides a good and helpful introduction into Sequelize and how to use it with Node.js[2]. It certainly helped with this project to get started and move from a conceptional ERM to a physical model including foreign key dependencies and creating a seed dataset, which enabled testing the core functionality of the application: sharing inventory items from different stores, managing shared shopping lists per store and date, and simulating multiple family members accessing the application. Using the

[1] More allowed callers can be added to the 'allowedlist' in the server.js file (lines 14-16)
[2] Kadwill (2020) provides a shorter introduction and overview of Sequelize ORM with Node.js and Express.js

Sequelize CLI (Command-Line Interface) by sequelize.org (2023) is very helpful. It allows to create the desired components as empty shells, which then can be filled with the desired logic.

The next iteration of the project addresses the core functionality. For example, let family members access and display a shopping list, let family members add items to a list, or remove them, or just change their quantity. Other core functionality includes the shopping process itself and addresses scenarios like what if the family member is in the store and wants to add an item, taking a picture and just added for everyone to see. These are some of the basic operations for this phase. In the initial design, there was only one Shopping List component and during this phase most of the functionality is developed using the initial base version.

While working on the shopping list and the process of being in a store and looking for items, adding new inventory items is an obvious action. To do that, taking a picture of the item and adding it to the list seems to be obvious. Taking a picture with a mobile device can be challenging due to different picture formats. Mobile devices using iOS take pictures in Apple's proprietary format HEIC/HEIF and deviate from the internet standard PNG. Also, transferring large size pictures between client and server seems to be too expensive, especially, when a mobile-first approach is aimed at. Faure (2023) provides an effective utility, ngx-image-compress, to compress pictures in an Angular environment. Unfortunately, it does not support Apple's format, but Saleem (2023) provides a converter, heic2any, to JPEG, which is ultimately used in this application to store picture at around 100 kB in JPEG format.

The pictures of inventory items are stored in the database as blobs. Tachyon (2019) provides some hints on how to handle pictures coming from a database in Angular 7, which also applies to Angular 17. To minimize requesting pictures of inventory items, the client maintains a cache of pictures and inventory items. It is a simple map data structure that uses the `inventory_id` as key, without this approach, the app has significant performance, response, and scalability issues. Similarly, the backend server also maintains various caches to minimize database access. For example, the server has a cache for all shopping lists, so that family members have fast access to this data. The shopping states for a shopping list are also kept in process-memory to quickly respond to requests. These caches are initialized during start-up of the server.

The final iteration of this project restructures the code into more well-defined components, moves functionality into better places and overall streamlines the implementation. The more detailed architecture diagram of this project (Sassin, 2024, p. 4) provides a good overview of how the front-end and the back-end are structured in regards to components and what they share.

Lessons Learned

The project faced some issues to use the new version of Angular 17 instead of the older version 14. The new version has some new control elements for HTML (Chautard, 2023; Gongora, 2023) and more importantly `app.module.ts` is no longer the default (rippleytrigger, 2024) and people, including myself, were wondering how to move from NgModule to standlone components. It took some effort and additional reading to apply the new model, which this project has accomplished. Another interesting aspect was the usage of a Sequelize ORM and its limitations. Many of the examples in the manual (sequelize.org, n.d.) were simple cases but this application had the need for more complex queries that require sub-queries, which is a known weakness of Sequelize[3]. The usage of set operators such as MINUS or EXCEPT are not supported and require native SQL queries[4].

Overall, the project accomplishes what it was thought to do. Multiple family members can access the app, share shopping list, modify them and get updates in near real-time[5]. Personally, I gained more Angular and Node knowledge and enjoyed developing this web-based full-stack application.

[3] Refer to ~/PJWD/FamilyShoppingList-server/controllers/shopping_list.controller.js lines 91-117 and 119-142
[4] Refer to ~/PJWD/FamilyShoppingList-server/controllers/family_member.controller.js lines 11-32
[5] Polling is set to 5000 ms; ~/PJWD/FamilyShoppingList-client/src/app/shoppinglist/shoppinglist.service.ts

Bibliography

Bootstrap team. (n.d.-a). Bootstrap. Retrieved December 28, 2023, from https://getbootstrap.com/

Bootstrap team. (n.d.-b). Bootstrap Icons. Retrieved December 28, 2023, from https://icons.getbootstrap.com/

Boucheron, B. (n.d.). How To Install Node.js on Ubuntu 20.04 | DigitalOcean. Retrieved December 11, 2023, from https://www.digitalocean.com/community/tutorials/how-to-install-node-js-on-ubuntu-20-04

Chautard, A. (2023, November 7). Angular 17: New control flow syntax. Medium. https://blog.angulartraining.com/angular-17-new-control-flow-syntax-4fbec4772d04

Dedigama, M., & Lee, R. (n.d.). How To Use Sequelize with Node.js and MySQL | DigitalOcean. Retrieved December 17, 2023, from https://www.digitalocean.com/community/tutorials/how-to-use-sequelize-with-node-js-and-mysql

Durante, D. (n.d.). Supercharging Node.js Applications with Sequelize. Packt Publishing. Retrieved December 19, 2023, from https://learning.oreilly.com/library/view/supercharging-node-js-applications/9781801811552/

Faure, D. (2023, June 23). Ngx-image-compress. Npm. https://www.npmjs.com/package/ngx-image-compress

Gongora, D. (2023, December 7). Angular 17: Things you should know about the new version. Medium. https://medium.com/@dgongoragamboa/angular-17-things-you-should-know-about-the-new-version-940f00fcd221

Google. (n.d.). Angular—The modern web developer’s platform. https://angular.io/

Hobbs, S. (n.d.). What is CORS? Complete Tutorial on Cross-Origin Resource Sharing. Auth0 - Blog. Retrieved December 11, 2023, from https://auth0.com/blog/cors-tutorial-a-guide-to-cross-origin-resource-sharing/

Kadwill, T. (2020, February 27). Using Sequelize ORM with Node.js and Express. Stack Abuse. https://stackabuse.com/using-sequelize-orm-with-nodejs-and-express/

Kumar, D. (2020, November 7). How CORS (Cross-Origin Resource Sharing) Works? The Startup. https://medium.com/swlh/how-cors-cross-origin-resource-sharing-works-79f959a84f0e

Linuxize.com. (2020, April 24). How to Install Google Chrome Web Browser on Ubuntu 20.04. https://linuxize.com/post/how-to-install-google-chrome-web-browser-on-ubuntu-20-04/

MarketSplash Team. (2023, August 25). How To Build Applications With Node.js And Angular.js. MarketSplash. https://marketsplash.com/tutorials/node-js/node-js-and-angular-js/

Modzilla. (2023, August 10). Cross-Origin Resource Sharing (CORS)—HTTP | MDN. https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

Node.js. (n.d.). HTTP | Node.js v21.4.0 Documentation. Retrieved December 15, 2023, from https://nodejs.org/api/http.html

rippleytrigger. (2024, April 15). Why doesn’t App Module exist in Angular 17? [Forum post]. Stack Overflow. https://stackoverflow.com/q/77454741

Saleem, A. A. (2023, March 29). Heic2any. Npm. https://www.npmjs.com/package/heic2any

Sassin, O. (2024). Family Shopping List.

sequelize.org. (n.d.). Manual | Sequelize. Retrieved December 15, 2023, from https://sequelize.org/v5/manual/getting-started.html

sequelize.org. (2023, December 15). Sequelize CLI | Sequelize. https://sequelize.org/docs/v7/cli/

Tachyon. (2019, March 24). Answer to “Angular 7—Display Images from byte in Database.” Stack Overflow. https://stackoverflow.com/a/55322285

Zanini, A. (2023, December 6). Express.js adoption guide: Overview, examples, and alternatives. LogRocket Blog. https://blog.logrocket.com/express-js-adoption-guide/