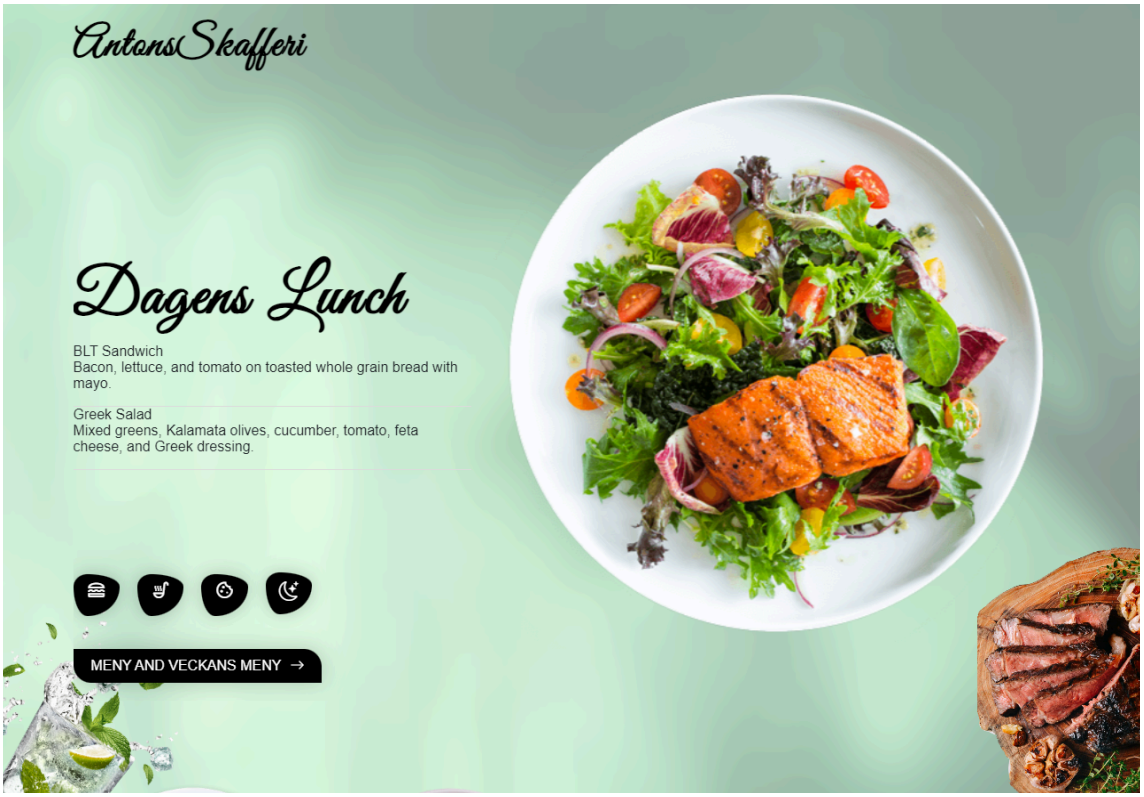
Antons skafferi



Examiner: martin.kjellqvist@miun.se Martin Kjellqvist

Presentation data: 2024-03-13

Authors:

arfr1901@student.miun.se

jaha2101@student.miun.se

luam2100@student.miun.se

Saal2107@student.miun.se

yaba2100@student.miun.se

Abstract

This report is a compilation of the work carried out by a group of five students for a comprehensive Jakarta EE project. In this project, we were tasked to work as a full stack development team, to meet the demands of a fictional client. The fictional client was a restaurant owner that expressed a need for a modern, digital solution to simplify the tasks of waiters and kitchen staff. More specifically, the client requested a set of applications that would enhance their operations and marketing; a restaurant website for the public view, a dashboard for administrative tasks, a mobile application for taking and managing orders, a mobile application for the kitchen staff to view orders, and a mobile application for visualizing the schedule of each staff member. In this report, we describe our workflow and methodology, followed by technical implementations, and the resulting mobile applications and websites that were created. Finally, we end the report by discussing the results, the quality of the teamwork, and reflecting on how the number of group members can affect productivity.

Terminology

DDL Data Definition Language

DTO Data Transfer Object

ORM Object Relational-Mapping

JPA Jakarta Persistence (Java Persistence API)

JDBC Java Database Connectivity

JPQL Java Persistence Query Language

CRUD Create, Read, Update, Delete and Execute

MVC Model View Controller

MVVM Model-View-View-Model

JSF JavaServer Faces

REST Representational State Transfer

API Application Programming Interface

Table Of Contents

1. Introduction

1.1. Bakgrund och problemmotivering

Because of the rapid digitalization of today’s society, various industries are recognizing the need for developing applications to meet their specific needs and help their business grow. The restaurant industry is no exception, with digital tools offering numerous solutions to improve service and efficiency. The owner of Antons Skafferi, a restaurant open seven days a week and serving a variety of dishes, has identified a need for technological advancement to modernize their workflow. The restaurant's owner is looking to replace outdated systems and improve communication between staff and kitchen personnel, and to improve marketing by using a website to attract more customers by establishing an online presence to showcase events and the menus.

To help solve this issue, we developed web pages and mobile applications tailored to Anton's Skafferi's requirements. This included creating a website featuring the daily lunch menu, event listings, a comprehensive menu page, and an admin section. Mobile applications were also developed for the serving staff to manage orders and for the kitchen staff to track ongoing orders and update their status, all integrated through a database and various APIs. The purpose of this report is to document the development process of these applications for Antons Skafferi and to showcase the achieved results, by highlighting the shift from traditional order-taking methods to a digital system that manages orders from creation to completion.

1.2. Future Work

1.2.1 Employee Work Shifts App

Because of time constraints, we weren't able to implement a fully functional scheduling application for the employees. In the finalized system, the staff members will utilize an integrated mobile application for viewing their individual schedules. Here they will be able to change their status, that is, if they are available or unavailable to work for the shift. Of course, the view of this application will change dynamically, according to the schedule defined by the restaurant owner in the administrator's page.

1.2.2 Restaurant App

In the restaurant app, the final version will show the selected table number at the top of the screen, where the orders are also visible for that table (see figure 22). Furthermore, we would make sure that for each menu item pressed, they are added to a single order for the given table, as opposed to the current solution, where each click results in a unique order. This would require a fundamental change to the API's structure. Lastly, the auto-refresh failure would be addressed, eliminating the need to manually reboot the entire application to see the orders made for a specific table.

1.2.3 Kitchen application

During our recent testing and debugging phase, we identified several functionality issues within the kitchen application, which we were unable to address due to time constraints.

Auto-Refresh Failure: Despite updating the status in the database after checking a box, the API and kitchen application do not automatically refresh. Specifically, after marking menu items as ready (via the corresponding checkbox), the item status fails to update in the API. A manual restart of both the web-server and the application is necessary to change the item indicator to green.

Checkbox Default Value Mismatch: The default states of checkboxes do not align with the database entries. For instance, even if an appetizer has been served (indicated by a green color), its checkbox remains unchecked by default.

Unchecked Boxes Handling: The application currently lacks functionality to properly handle the unchecking of boxes, an issue we have not yet addressed.

1.2.4 Miscellaneous

The following are the remaining tasks which are left for future work.

- Make sure the popup-screen for the A La Carte menu works and is displayed properly in the customer page.
- Remove “Dessert” and “Healthy Foods” options in the customer page.
- Ensure that the mobile app has a functioning scroller, in case of large orders.

Figure 1 shows an excerpt for some of the unfinished tasks, from Trello.

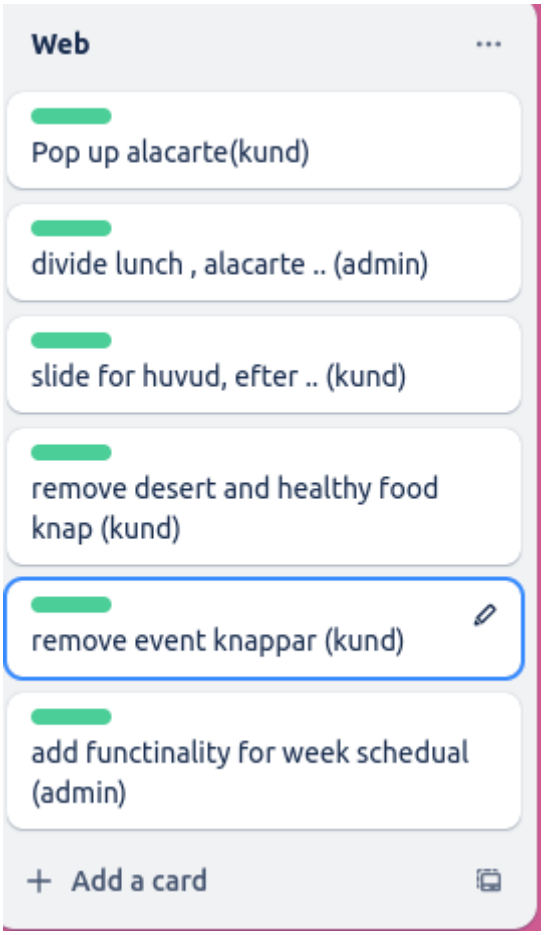


Figure 1: Excerpt of unfinished tasks.

1.3. Purpose

The purpose of this project was to simulate the experience of working as a software development team using an agile workflow. Much like out in the industry, our group aimed to meet the demands of a client. The fictional client was played by our mentor, Martin Kjellquist. His name was "Anders Svensson", and he was the owner of the restaurant "Antons Skafferi". The goal was for us to utilize the designated tech stack to deliver a product that would satisfy the needs of our client.
This would in turn require a significant amount of planning, discussion, and technical work within the group. Having said that, the core purpose of this project was to work together as a team; to face the challenges, and try to overcome them together. The results were important, but teamwork was paramount.

1.4. Concrete and verifiable goals

Our project aimed at streamlining restaurant operations and boosting its online presence with specific, measurable goals:

1. Implement a digital communication system between staff and the kitchen.
2. Launch a comprehensive website for advertising, event information, and table reservations.
3. Introduce a digital platform for shift scheduling and management.
4. Develop user-friendly interfaces for inventory oversight and database management.
5. Create dedicated web pages for the daily lunch menu, events, the à la carte menu, and administrative tasks.
6. Develop mobile applications for staff to manage orders and for the kitchen to track and update order statuses.7. Make so Anton can assign schedules to his employees.
7. Create a schedule app that employers can see and change shifts.

These objectives focused on enhancing the restaurant's efficiency, online visibility, and customer engagement, aiming for a smoother operation and better customer service.

1.5. Authors' Contributions

1. Arvid DB, Beans, Admin page forms and connection to database, restaurant app order POST request
2. Jakob DB, Webbserver, DTO
3. Luwam: Validation of Websites, Design administration and Java classes in Restaurant Application
4. Saly: Design administration and Customer websites and Restaurant Application
5. Yasan MVC, DTO, Retrofit, APIrest, classes in restaurant App, Kitchen App

2. Theory

The theory section offers a detailed exploration of contemporary software development practices, combining agile methodologies, programming languages, and development tools with database technologies, server frameworks, and design principles. It emphasizes the importance of adaptability, collaboration, and technical excellence, highlighting the role of Java, Scrum, Extreme Programming (XP), GitHub, and various design patterns in creating efficient, scalable, and user-centric software solutions. By integrating these elements, the section provides a comprehensive overview of the theoretical foundations and practical applications that underpin modern software development, underscoring the dynamic interplay between technology, methodology, and design in the creation of robust software products.

2.1 Comprehensive Agile Development Methodologies

Agile methodologies represent a paradigm shift in software development, emphasizing adaptability and customer collaboration over rigid planning and documentation. Among these, Scrum stands out for its structured approach to managing complex projects through iterative cycles called sprints, fostering team collaboration and proactive problem-solving. Each sprint is a self-contained unit of work allowing for the development, review, and adaptation of project components.

Extreme Programming (XP) complements Scrum by focusing on technical best practices and personal teamwork. XP prioritizes technical excellence and responsiveness to changing customer requirements, advocating for practices such as continuous integration, test-driven development, and pair programming. These methodologies underscore the importance of feedback loops and adaptability, ensuring that the final product closely aligns with user needs and expectations.

In our project, we embraced agile methodologies through a combination of Scrum and XP practices. For example, we conducted regular sprint planning sessions to outline goals and prioritize tasks, followed by daily stand-ups to facilitate communication and address any impediments. Continuous integration and test-driven development were integral components of our development process, ensuring code quality and responsiveness to evolving requirements. Challenges such as adapting to changing priorities were addressed through

frequent retrospectives, where the team reflected on our processes and made adjustments for future sprints. This iterative approach not only improved our development efficiency but also enhanced the overall quality of our software product. Additionally, we utilized Trello, an online project management tool, to organize and visualize our tasks across 'To Do', ‘Doing’, and 'Completed' lists, facilitating transparent collaboration and task tracking among team members, as we can see in figure 2.

Figure 2: SCRUM model, generated by ChatGPT

2.2 Communication complexity

The number of communication channels increase by the formula $\frac{n(n-1)}{2}$, where ‘n’ is the amount of team members. The n(n−1) calculates all possible pairs of communication when every member can communicate with every other member.

2.3 Social loafing

When the number of people in a group increases, social loafing tends to happen. This occurs because individual contribution becomes less hard to identify. This leads to decrease in effort as they see their input less critical to the group outcome.

2.4 Advanced Programming Languages and Frameworks

Java's role as a versatile, object-oriented programming language is undisputed in the development ecosystem. Its platform independence, robust standard libraries, and extensive community support make it ideal for a wide range of applications, from enterprise solutions to mobile applications.

In our project, we leveraged Java and its associated frameworks to develop a robust and scalable software solution. JavaServer Faces (JSF) and Jakarta Server Faces played a pivotal role in extending Java's capabilities and building user interfaces for our web applications following the MVC framework. By utilizing these frameworks, we were able to create dynamic, component-based user interfaces that enhanced the user experience and streamlined development efforts.[5]

Additionally, we utilized Payara Server to create a RESTful API with Jakarta EE 10, showcasing Java's versatility in building robust and scalable web services. Payara provided us with the tools and capabilities necessary to develop RESTful APIs that seamlessly integrated with our frontend applications, ensuring efficient data communication and interoperability.[3]

Furthermore, Retrofit, as a RESTful client library, exemplifies Java's adaptability to modern web services. By employing Retrofit, we simplified API interactions through declarative programming, reducing boilerplate code and improving code maintainability. This not only enhanced our development efficiency but also contributed to improved application performance and scalability.[4]

These technologies provided us with a solid foundation for building complex software systems while ensuring compatibility and interoperability across various platforms and environments.

2.5 Integrated Development Tools and Environments

The selection of development tools and environments significantly influences the efficiency and quality of software projects. Android Studio and IntelliJ offer rich features for Java development, including code editing, debugging, and UI design, tailored for Android apps and Java applications, respectively.

In our project, we utilized Android Studio and IntelliJ extensively to facilitate Java development, leveraging their powerful features to streamline coding, debugging, and user interface design processes. These IDEs provided us with comprehensive toolsets and robust debugging capabilities, enabling us to efficiently develop and test our software solutions.

Moreover, GitHub played a pivotal role in facilitating collaborative software development within our project. Beyond serving as a code hosting platform, GitHub provided us with tools for project management, issue tracking, and community engagement. The integration of GitHub into our development workflow promoted transparency, and seamless collaboration among team members.[6] [7] [8]

Additionally, we utilized ChatGPT for assistance in coding and a whiteboard for collaborative app design within our development workflow. These tools facilitated a user-centered design approach, enabling iterative testing and refinement of user interfaces. During the design phase, we created multiple mockups on the whiteboard and collectively evaluated them to select the most user-friendly design. By incorporating ChatGPT for coding assistance and utilizing a whiteboard for collaborative app design sessions, we were able to gather valuable feedback early in the design phase, resulting in user-friendly and visually appealing software products.

The strategic selection and integration of development tools and environments in our project played a crucial role in enhancing productivity, collaboration, and the overall quality of our

software solutions. By leveraging these tools effectively, we were able to deliver impactful software products that meet the needs of our client.

2.6 Database Management and Integration Technologies

Mariadb is a open source relation database management system.
Normalization in databases
Database normalization is one technique used to structure a database to get less redundancy and ensure data integrity. Organizing data in tables and defining the relationships between the tables so data can be managed and stored effectively.

Database views
Is a virtual tables they don't store data but present data from other tables in a structured form. Let data be presented in a simple manner.

ORM
“Technique for converting data between a relational database and the heap of an object-oriented programming language. This creates, in effect, a virtual object database that can be used from within the programming language.” according to [12]

JPA is a ORM framework where entities (Java classes) are mapped to database tables, through its EntityManager. JPA manages the lifecycle of entities within a persistence mapping context, facilitating CRUD operations without direct SQL queries using Java Persistence Query Language (JPQL). This abstraction simplifies database interactions, and make data access and manipulation more intuitive for developers, which ensures application portability across different database systems.

2.7 Server Technologies and Web Services

2.7.1 Retrofit

Retrofit is a REST client for Android and Java. This framework is used for establishing and managing API calls for the mobile applications. [10]

2.7.2 Glassfish

Glassfish is an open source application server for Java EE. It provides support for a Java based web application. Supports Servlets, JSP, JSF, EJB, JPA. It is used as a comprehensive tool to help us host, deploy and manage our web applications. [11]

2.7.3 DTO

“DTO does not have any behavior except for storage, retrieval, serialization and deserialization of its own data (mutators, accessors, serializers and parsers). In other words, DTOs are simple objects that should not contain any business logic but may contain serialization and deserialization mechanisms for transferring data over the wire.” according to [14]

2.7.4 REST API

“A REST API (also called a RESTful API or RESTful web API) is an application programming interface (API) that conforms to the design principles of the representational state transfer (REST) architectural style. REST APIs provide a flexible, lightweight way to integrate applications and to connect components in microservices architectures.” according to IBM [13]

2.8 Collaboration and Version Control Systems

During the development of the project, various applications were used for different purposes, each with their respective features and functions. These included Intellij IDEA Ultimate, Android Studio, Payara, XAMPP, GitHub, Trello and Discord. Intellij IDEA Ultimate 2023.3 and Android Studio 2023.1.1 were utilized for code development. Payara is an open-source application server based on the GlassFish server and compatible with Java EE standards, making it a suitable platform for Java EE applications. XAMPP, which stands for "X-

Cross-Platform, A- Apache, M- MySQL, P- PHP, and P- Perl," is another open-source platform used for creating local databases.

GitHub is a web platform based on cloud services, well-known among most developers for its primary function of code version control and it facilitates code sharing and collaboration within the group during the development process. For this purpose, different repositories were created for the various areas we worked on. Members created different branches under these repositories and worked together to develop the project. Trello is a web-based platform used for backlog planning, meaning the planning of tasks to be performed. It allows the user to create lists of tasks to be performed, and if necessary, it has the functionality to specify which member is to perform them. These lists can be moved between different stages of the workflow. Finally, Discord was used for communication within the group, where different servers were created for voice and text communication. To minimize confusion, servers were named according to their purpose, such as database, links, web, design, and more. The database server was used for communication regarding database tasks, links for link sharing, and the same principle applied to the others.

2.9 Design Patterns and Software Engineering Principles

The design of development is built upon parts from the SOLID principle, which stands for "S - Single Responsibility Principle, O - Open/closed Principle, L - Liskov Substitution Principle, I - Interface Segregation Principle, D - Dependency Inversion Principle". The principles that the group has utilized are: S indicating that a class should have a single responsibility, O stating that a class should be open for extension but closed for modification.

In addition to those, two different architectural patterns were used, namely MVC and MVVM. They were employed to separate and organize the code structure in Android development. The MVC model divides the code into three main components known as Model, View, and Controller. The Model represents the application's data structure and business logic, the View is the user interface displaying data to the user, and the Controller handles user interactions and updates the view based on the model's state. Similarly, MVC is used in web development to separate data management, data presentation, and user interaction handling into the model, view, and controller components. In this context, the model represents the application's data structure and business logic, the view displays data to

the user, and the controller processes user interactions and returns appropriate responses to the user through web connections.

Even though MVVM is similar to MVC, they share the same structure, with the only difference being that MVC has a Controller while MVVM has a ViewModel. ViewModel offers several advantages in modern app development. Its task is to clearly separate view logic and business logic, making the code easier to understand and maintain. Additionally, it becomes easier to test the code since ViewModel isn't dependent on the Android framework or any specific Android context. This also enables unit testing of business logic without having to deal with the complexity of Android components. Furthermore, the use of ViewModel minimizes the risk of lifecycle-related issues and allows for data management independent of the lifecycle of Android components. Lastly, ViewModel integrates well with data binding, facilitating a simpler connection between the view and ViewModel, and automatic UI updates when data in the ViewModel changes.

2.10 Java code conventions

2.10.1 Introduction

Code conventions are vital for programmers due to several reasons:

- Approximately 80% of a software's lifetime cost is dedicated to maintenance.
- Most software isn't maintained by its original author throughout its life.
- Conventions enhance code readability, enabling engineers to comprehend new code swiftly and thoroughly.
- When distributing your source code as a product, it should be packaged and organized like any other product.

Acknowledgments:
These conventions are based on the Java Language Specification by Sun Microsystems, with significant contributions from Peter King, Patrick Naughton, Mike DeMoney, Jonni Kanerva, Kathy Walrath, and Scott Hommel. For inquiries about adaptation, modification, or redistribution of this document, refer to our copyright notice [link]. Feedback on this document can be submitted through our feedback form.

2.10.2 File Names

Java follows a convention for its file types, meaning files containing Java program code end with the ".java" extension. This is deliberate to clearly indicate that the file contains Java

code. Adhering to this convention is important as it makes it easier for both developers and compilers to distinguish Java files from other file types in a project. By simply looking at the file name, one can quickly identify which files contain Java code and which do not. This simplifies the organization and management of files in a Java project and contributes to a more structured and efficient development process.

2.10.3 File Organization

In Java programming, it's important to keep your files well-organized with clear sections and avoid making lines of code too long more than 2000. Each file should contain only one public class or interface, along with any related private classes or interfaces. Start your files with a C-style comment and make sure to include package and import statements. Within classes and interfaces, maintain a consistent order and use four spaces for indentation. Keep lines of code relatively short and break them into multiple lines when needed. Group methods based on their functionality to make your code easier to understand. Alla these steps above will help you to more easily understand the code for you as developer and others, if it needs.

2.10.4. Indentation

Use four spaces as the standard unit of indentation. The specific use of spaces vs. tabs is not defined, but when using tabs, they should be set to represent every eight spaces, not four. Keep the lines under 80 characters to ensure compatibility with most terminals and tools. When an expression wont fit a single line, break it down. Break after comma before operator, prefer higher-level breaks to lower-level breaks, new line with the beginning of the expression at the same level on the previous line. If it leads to confusing code or to code that’s squished up against the right margin, just indent 8 spaces instead.

2.10.5. Comments

There are two kinds of comments: implementation (/*...*/) and documentation (/**..*/). Four styles of implementation comments: block, single-line, trailing and end-of-line

Implementation comments are meant for commenting out code or for comments about the particular implementation. There are four styles of implementation comments: block, single-line, trailing and end-of-line

Block Comments: start of files or methods for descriptions.

Single-Line Comments: brief comments, aligned with the code's indentation.

Trailing Comments: Same line as the code, spaced from the code.
End-Of-Line Comments: Utilize // for commenting out lines or parts of lines. Avoid for multiple consecutive lines of text comments but used for commenting out code.

Doc comments can be extracted to HTML files using the javadoc tool. Doc comments are meant to describe the specification of the code, from an implementation-free perspective. Provide details for Java classes, interfaces, constructors, methods, and fields, meant for API documentation. Information not suited for public documentation should be placed in implementation block comments or single-line comments immediately after the declaration.

2.9.6. Declarations

Number Per Line One declaration per line is recommended since it encourages commenting. In absolutely no case should variables and functions be declared on the same line. Do not put different types on the same line.

Placement:
Put declarations only at the beginning of blocks
Exception to the rule is indexes of for loops, which declared in the for statement.
Avoid local declarations that hide declarations at higher levels

When possible initialize local variables where they’re declared.
When coding Java classes and interfaces, the following formatting rules should be followed:
No space between a method name and the parenthesis.
Open brace “{” appears at the end of the same line as the declaration statement.
Closing brace “}” starts a line by itself indented to match its corresponding opening statement, except when it is a null statement the “}” should appear immediately.

2.10.7 Statements

In Java, the manner in which statements are written needs to precisely reflect the convention. Each line should have only one statement at most. `if` statements must always have curly brackets, for-loops should adhere to the *initialization; condition; update*` signature, and

return statements must be descriptive. `while`, `do-while`, switch-statements, and `try-catch` blocks need to follow a strict form.

2.10.8. White Space

Blank lines conventions are important, since they improve readability by logically separating segments of code. Two blank lines should always be used between sections of a source file, and between class and interface definitions. One blank line should always be used between methods, between the local variables in a method and its first statement, and before a block or single-line comment.

2.10.9 Naming conventions

Naming conventions helps with easier comprehension of a code, where differentiating functions, constants, packages and class names does actually highlight the structure much faster of a code we are trying to understand. In table 1 we can see the naming convention rules for Java.

Table 1: Naming convention rules for Java

Identifier type	**Rules for Naming**	**Examples**
CLASSES	Nouns, uppercase letters at first and each internal word as well, descriptive, and non acronyms (usually) unless widely used like URL or HTML.	class Raster; class ImageSprite;
INTERFACES	Same as classes.	interface RasterDelegate; interface Storing;

METHODS	Verbs, lowercase letters at first but with uppercase letters for each internal word.	run();
		runFast();
		getBackground();

VARIABLES	Short, indicate the intent of its usage, avoid	int age;

	one-character names (temporary variables only) and lowercase letters at first but with uppercase letters for each internal word.	int i; char *startingLetter; float myWidth;

CONSTANTS	All uppercase letters with underscores separation between internal words.	int MIN_WIDTH = 4;
		int MAX_WIDTH = 999;
		int GET_THE_CPU = 1;

2.10.10 Programming practices

1. Only make **public** instances or class variables if it is an absolute necessity, like having a struct within a class where we need to use the class variable or instance inside the struct. Otherwise it is private access.
2. Try to use class name instead of creating an object of the class, if an access for the class methods or variables (static) is needed. For example:
    a. OK
        i. classMethod();
        ii. AClass.classMethod;
    b. AVOID
        i. ClassName anObject;
        ii. anObject.classMethod();
3. Avoid coding direct numerical constants. An exception can be counter values inside a loop like “-1”,”0” and “1”.
    a. For (int i = 0; i < size() ;i++) {
        …
        }
4. Avoid several variable assignments.
    a. fooBar.fChar = barFoo.lchar = 'c';.
5. Avoid using assignment operations in confusing places for a reader.

a. if (c++ = d++) {
   …
   }

6. Avoid embedded assignments.
   a. AVOID
      i. d = (a = b + c) + r;
   b. OK
      i. a = b + c;
      ii. d = a + r;
7. Avoid operator precedence problems by using parentheses liberally in expressions.
   a. AVOID
      i. If (a == b && c == d)
   b. OK
      i. If ((a == b) && (c == d))
8. Avoid unnecessary extra returning values codes.
   a. AVOID
      i. if (booleanExpression) {
            return TRUE;
         } else {
            return FALSE;
         }
   b. OK
      i. Return booleanExpression;

   Similarly

   c. AVOID
      i. if (condition) {
            return x;
         }
         return y;
   d. OK
      i. return (condition ? x : y);
9. Parenthesized expression that contains a binary operator before the “?” in the ternary “?:” operator.

a. (x >= 0) ? x : -x

10. Use XXX in a comment to flag something that is bogus but works. Use FIXME to flag something that is bogus and broken.
    a. function addNumbers(a, b) {
        let sum = a + b; #XXX: This addition may not handle edge cases
        return sum;
    }
    b. function divideNumbers(x, y){
        result = x / y; #FIXME: Division by zero may occur here
        return result;
    }

3. Methodology

This section details the organizational structure, collaborative processes, and communication strategies adopted during the project's development phase. It emphasizes the use of agile methodologies, such as Scrum and Extreme Programming (XP), and highlights the significance of tools like Trello, GitHub, Discord, and Google Drive in facilitating teamwork and coordination. Additionally, it discusses the experimentation with pair programming and mob programming to enhance collaboration and skill-sharing among team members. Overall, it underscores the importance of effective communication and cohesive teamwork in achieving project success.

3.1. Division of work

In our project, we adopted a flexible and collaborative approach to development, heavily inspired by agile practices, particularly Scrum and elements of Extreme Programming (XP). Our work process was structured around the creation of a Scrum board at the outset, with one of our team members stepping into the role of Scrum Master to guide the workflow.

Our team was organized into smaller groups, each focusing on a specific aspect of the project—mobile apps, web and admin interfaces, and the database. This division allowed us to tackle the project from different angles simultaneously while maintaining a cohesive development process. We used Trello, an online project management tool, to build our Scrum

board, which served as a dynamic to-do list. This board listed all our goals and tasks, which were continuously updated as we completed old tasks and identified new ones. It was a crucial tool for keeping the team aligned and focused on our objectives. As shown in Figure x

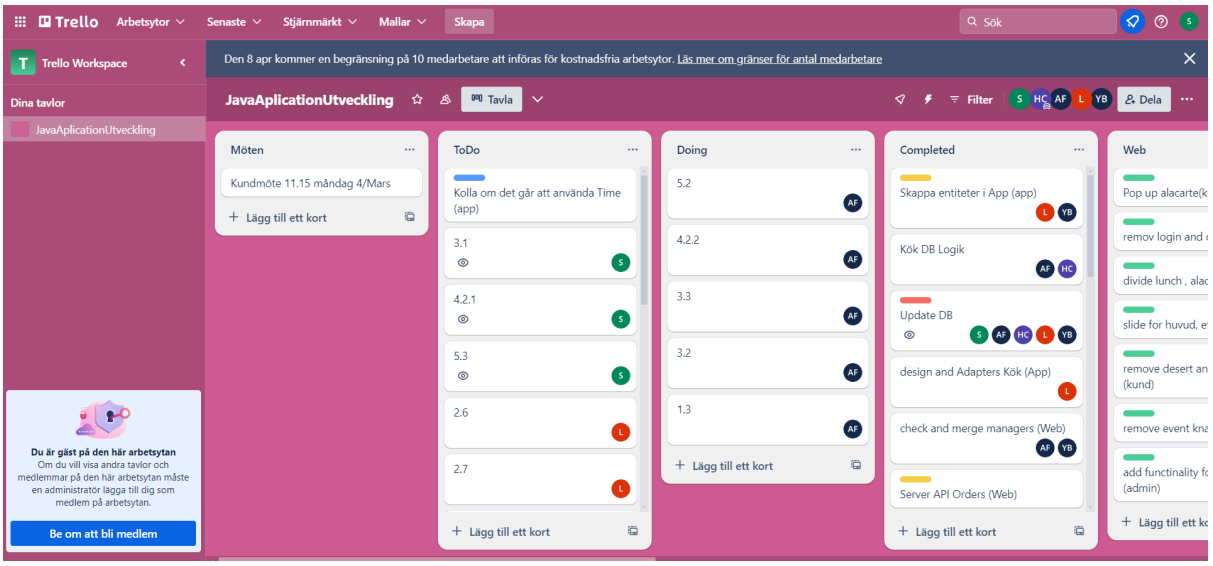


Figure 3 - Example of Trello, to plan and distribute the work

During our initial group meeting, we developed a learning matrix to evaluate our individual capabilities and interests, ranging from 1 to 5, in various technical skills and teamwork abilities as shown in figures x and x. This approach was designed to understand each member's strengths and preferences, enabling us to allocate tasks effectively. However, we encountered a limitation as we lacked formal personality and technical assessments to accurately gauge each member's skill level. Consequently, we relied on self-assessment, which introduced a degree of subjectivity to our evaluation process.

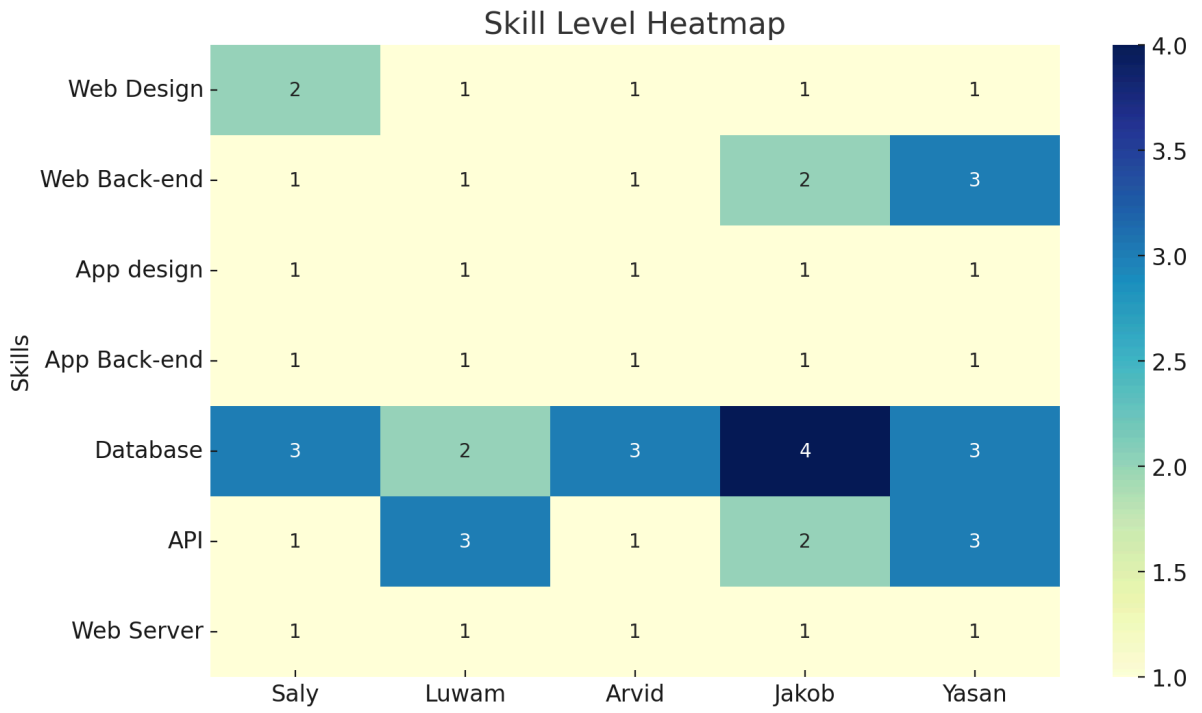


Figure 4: Technical abilities of each member

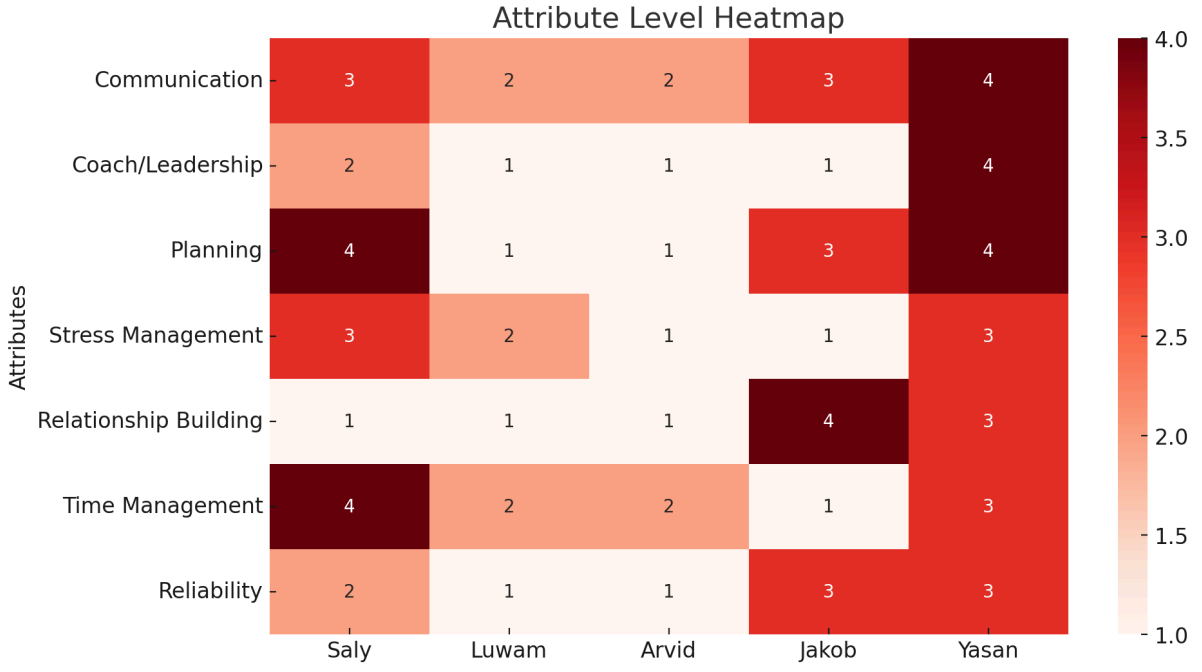


Figure 5: Personal abilities for each group member

Based on the self-reported outcomes, we strategically distributed tasks, ensuring that individuals with higher proficiencies in specific areas took the driver, while those keen on enhancing their skills in those areas served as observers or navigators. This arrangement not only optimized our workflow by leveraging each member's strengths but also fostered a learning environment that encouraged skill development and knowledge sharing within the group. Through this method, we aimed to balance project efficiency with personal growth and team collaboration.

For code management and version control, we utilized GitHub, creating specific branches off the main branch for different tasks (e.g., `design-admin-page` for admin page designs). This strategy facilitated smooth integration of our work into the main codebase, even when multiple team members worked on the same files.

Throughout the project, mob programming emerged as a pivotal technique, engaging the entire team in working on the same task simultaneously. This approach not only fostered enhanced collaboration and understanding among team members but also allowed us to navigate complex coding challenges collectively and benefit from mutual learning experiences. Communication stood as the foundation of our success, underpinned by daily meetings that established a supportive network for addressing any challenges encountered. These gatherings were crucial not just for problem-solving, but also for aligning on progress and strategizing future tasks. Following these collaborative sessions, we transitioned to remote work, focusing on completing outstanding tasks independently. This hybrid model of in-person and remote collaboration maintained our project's flexibility and efficiency, ensuring a cohesive and productive team dynamic.

We utilized Discord for our daily communications, leveraging its features to share important documents, conduct virtual meetings, and create events for scheduling meetings and key project milestones. The ability to have real-time discussions, coupled with the flexibility of asynchronous communication, allowed us to maintain a high level of coordination and stay aligned on our goals. Additionally, we made use of a shared Google Drive folder for document and report writing. This centralized repository ensured that all project-related materials were easily accessible to every team member, facilitating seamless collaboration

and information sharing. This integrated communication strategy was instrumental in navigating the complexities of the project and achieving our objectives.

3.2. Design and Prototype

The design and prototyping decisions were largely influenced by the client's preferences, as well as the functionality it should support. For example, our client expressed a dislike for buttons. As such, they were generally avoided, instead favoring views where most elements are present within the same page and accessible through scrolling.

The client conveyed his vision for the customer page, wanting it to look aesthetically pleasing and elegant. These descriptive words served as the framework for our design. Later on in the project, we also took into consideration the client's ideas regarding the overarching design of the restaurant application's views.

Early on in the project, mockups and low-fidelity sketches of the application and website were created using whiteboards and marvelapp. As the course progressed, the whiteboard became a consistent tool used for quickly brainstorming ideas about basic designs. Discussions were had about the design and how it should look. The group members who were assigned the responsibility of the design would then simply implement the stylesheet or XML directly during development of the application or website, turning it into a high-fidelity view.

3.3. Crafting Apps and Websites: Tools and Techniques

As previously mentioned in the "Purpose" chapter, the tech stack to be used for this project was not of our choosing; it was predetermined by our mentor. Jakarta Enterprise Edition was at the heart of software development for both the applications and the websites. The integrated development environments (IDEs) used were Android Studio for mobile development, and IntelliJ Ultimate Edition for web development.

For the android applications, the Retrofit framework was used to establish and manage REST API calls. For the website applications, the MVC design pattern was employed, together with JPA, which allowed for a smooth interface between the view, the managed bean, and operations with the database. Payara was used as the web server, which offered a rich suite of tools to facilitate the connection and management of the database pool and credentials.

The database management system used was MariaDB. Some group members leveraged XAMPP with phpMyAdmin, while others only used MariaDB's integrated CLI for managing the DBMS.

3.3. Generative AI

Throughout this project, we have thoroughly leveraged the LLM, ChatGPT, for a wide range of tasks. It was used as an auxiliary tool, assisting us in learning about foreign topics. This includes technical concepts within the tech stack, and how to fulfill various design patterns programmatically, within the boundaries of convention. ChatGPT helped in generating advice and ideas with respect to how to structure code according to the product we wanted to deliver. The LLM was used extensively for debugging purposes, and for providing insight into best practices for coding, front-end to back-end.

In this report, ChatGPT was utilized as a way to swiftly generate ideas about how to structure or formulate certain paragraphs. Once generated, the texts were analyzed and modified to ensure it's aligned with the intent of the author. Finally, the generative AI model was used as an effective tool for translating Swedish texts into English.

4. Construction

4.1. Database

4.1.1. Database Development and Integration

We chose to use Mariadb as a database management system cause we all had experience using it before. We also chose to use the database local on host so we could work modularly on different tasks so we could work even if we had different database versions.

The foundation of our project was the development of a robust database system.

We created multiple ER diagrams, based on our interpretation of the business logic from the first meeting with the client. See figures in Appendix B.

The group had discussions with the client during the second meeting for more clarifications and to make sure our business logic aligned with the client's needs. From that we merged our ideas we had and with what the client clarified.

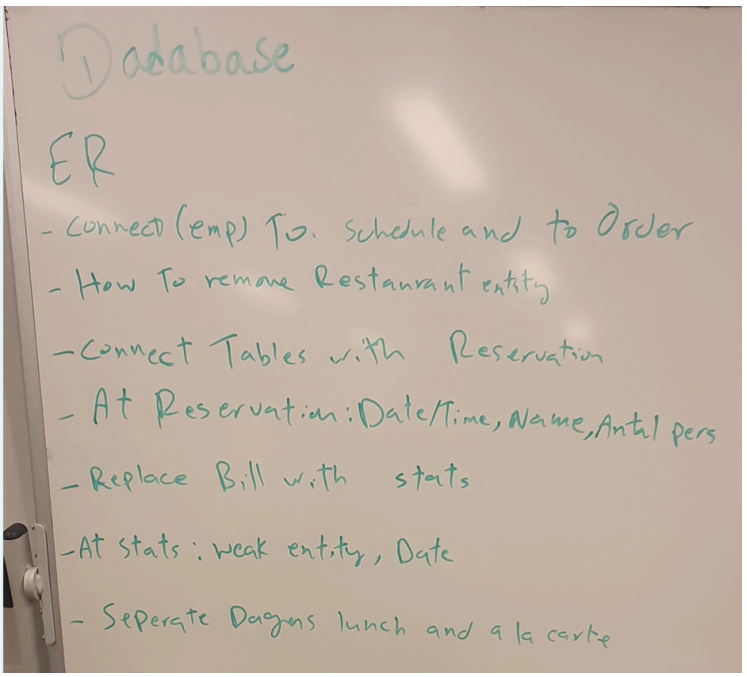


Figure 6: What needed to be added and fixed.

Compared the different ideas and what needed to be changed and merged them to this prototype ER.

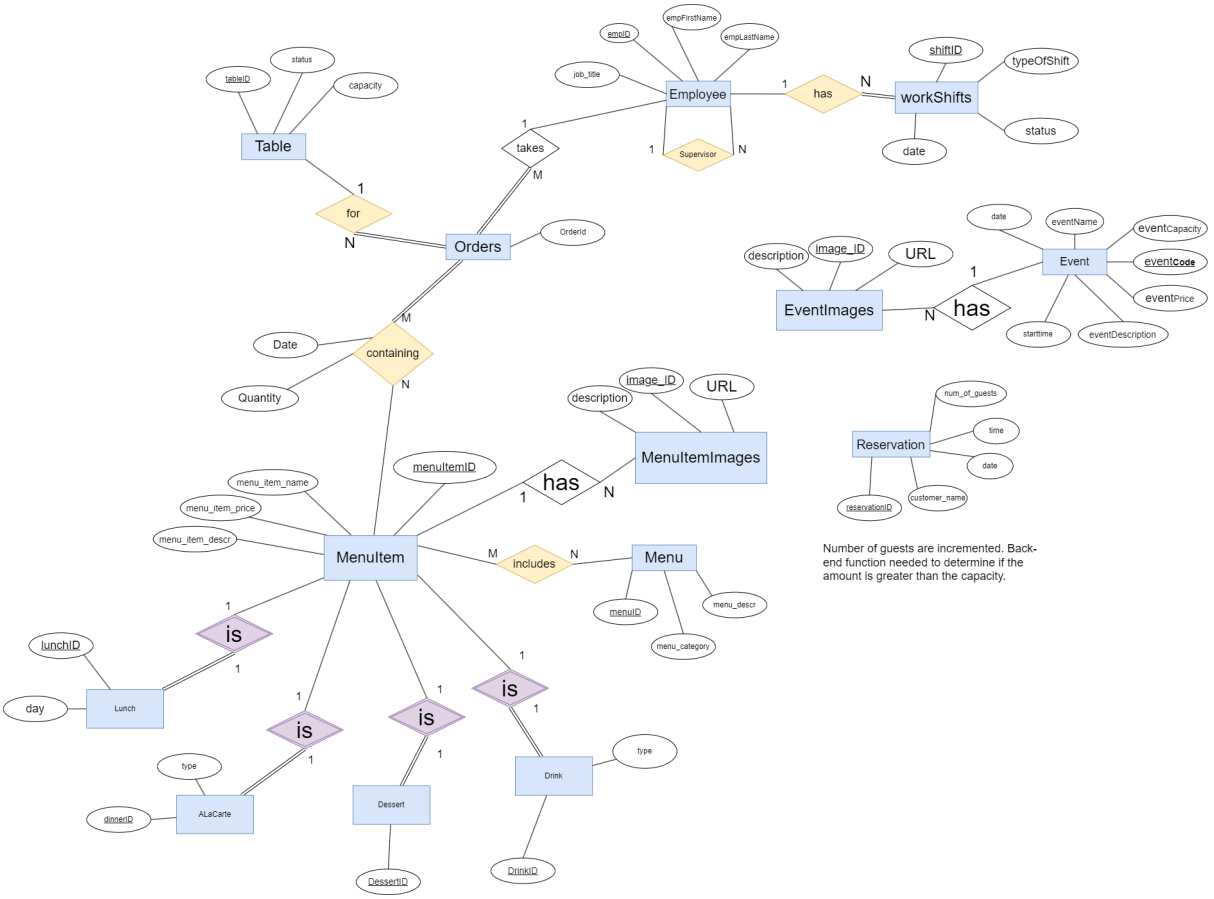


Figure 7: Prototype ER diagram

This became our database ER after the meeting with the client and merging our ideas and after Normalizing.

Then this diagram guided us to the creation of our DDL statements we saved to a text file to be shared so it was easy to import it to our mariadb server.

To set up the mariadb connection to payara we needed multiple steps. We needed to download the datasource module “mariadb” and put it in glassfish modules.

Then we needed to setup JDBC pool and resources as follows:

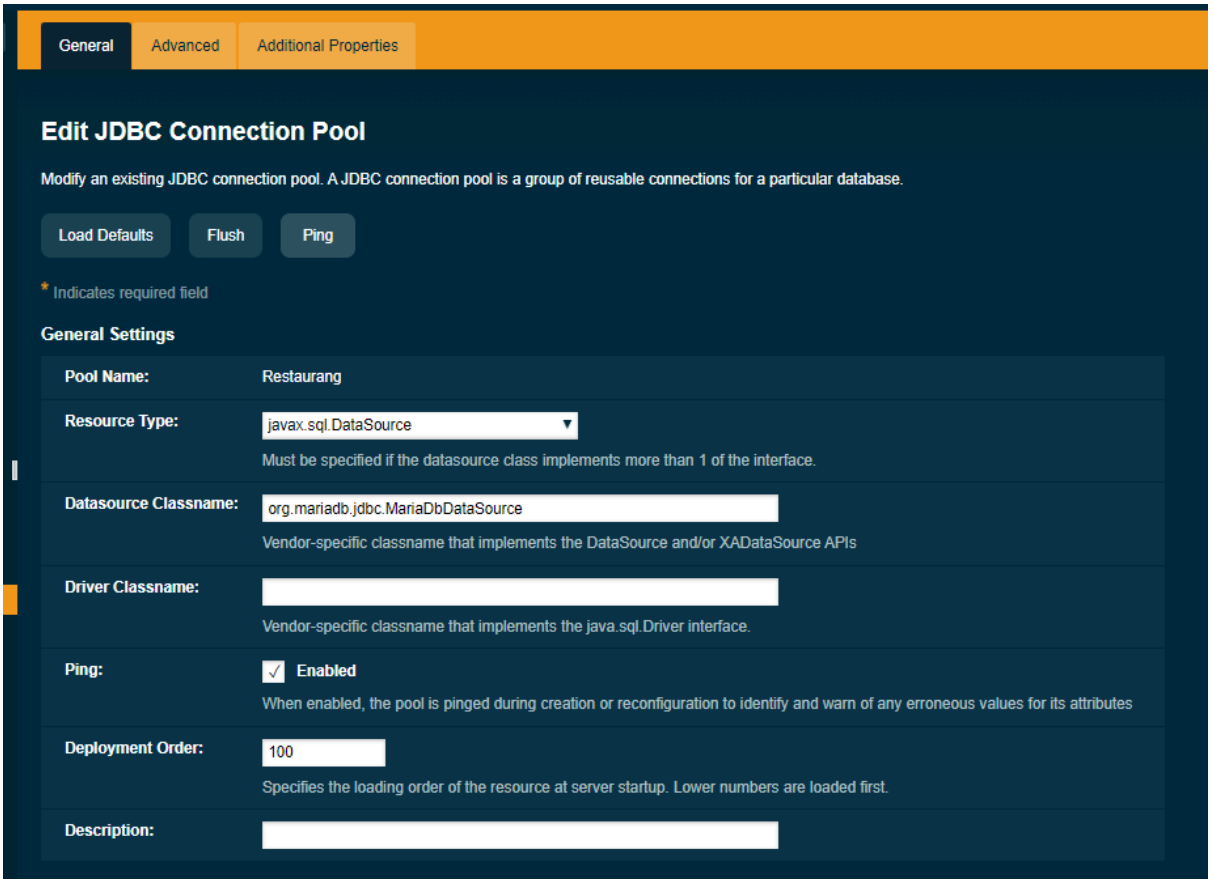


Figure 8. JDBC Database Pool Setup (Payara Server).

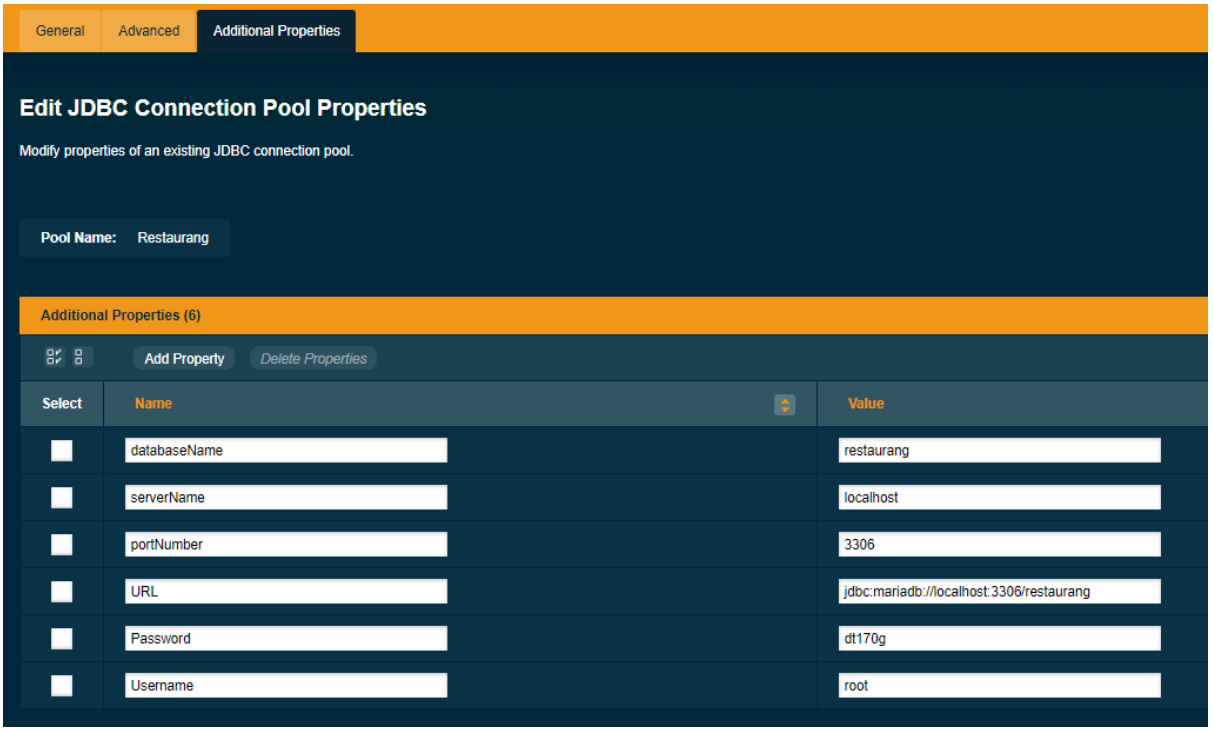


Figure 9: JDBC connection properties.

We saw some problems here, the reason being that case being sensitive for unix systems that made us all need to have the same name on the database. And this problem would occur more which lead us to set a standard for the database. That being all tables should be in capital letters and the database being in small letters being that windows would not allow them to set the database name in capital letters.
After succeeding establish our database structure and integrating it with mariadb we turned to data accessibility. First we tried doing all sql queries but the complexity and scalability and time it would take was not sustainable so we looked for a different solution. The solution was implementation of ORM by using JPA. This made it so we could use persistence mapping to generate entities of tables which was way faster than doing everything individually which made it so we could work with java objects directly. But there were still some problems with this. We needed to make annotations for one to many, many to one and many to many but this came more from our database's more complicated structure. To solve many to many and table view we made an ID class for that entity.

entity manager?

4.1.2. REST API and Client-Server Communication

To make sure our applications could get the data from the database we chose to use Rest API. Our REST API played a crucial role in connecting our mobile applications to the database. Implemented through GET,POST and PUT requests over HTTP, the API allowed clients to retrieve information as JSON objects or strings and to input specific data, such as creating orders or updating inventory. On the server side, each API function was encapsulated within a servlet, accessible via both GET and POST methods and PUT for updating the data.
For JSON data handling, we employed DTO classes to convert objects to and from JSON. But to convert our entities to DTO classes we made a mapper class that converted DTO to and from entities. We had a controller that handled https responses and requests and called the mapper. We did encounter multiple problems where we could send the data to the api so we could grab it from the api but when we wanted to add it to the database we would have

infinite loops caused by bidirectional entity references and more. We solved it by using DTO to precisely control the data structure.
We got up the api but we didn't have time nor resources to figure out the best solution or how the optimized api should look like we just wanted to get up data as fast as possible so we could get the data to the app. This was a mistake all it did was make it harder for the app. If we had put some more time into how we wanted the api data the apps would have it way easier. For future time a documentation for the api would be good for an easy handover to the client side in this case it was the kitchen and restaurant app.

4.2. Website

4.2.1. Customer page

In the project, developing a user-friendly and functional client-side interface for "Antons Skafferi" was of most importance, reflecting the restaurant's priority to provide a digital experience for its guests. The development phase began with the team dividing into smaller groups, each tasked with designing a sketch for the client-side layout. These initial low-fidelity prototypes were presented to our fictional client, Anders, portrayed by our mentor, Martin Kjellqvist. Anders selected the most appealing design, which set the direction for our development efforts.
Members were then assigned, sometimes in pairs, to translate the chosen design into HTML code, focusing on creating pages for lunch, à la carte menus, and events, ensuring consistent navigation and footer across the site. Initially, due to the absence of a database, the website content remained static, presenting a challenge in environment setup and prompting some members to work directly with HTML and CSS without GlassFish server integration.
As development progressed, the goal of the team shifted towards making the site dynamic. This involved establishing database connectivity and implementing JSP for page development, later transitioning all pages to JSF to accommodate database interactions more efficiently. The admin side development and database setup proceeded concurrently, with a dedicated group for each, ensuring that the client-side could dynamically display information like the day's lunch specials directly from the database.

Challenges such as session errors in Intellij and Glassfish, along with the cumbersome process of configuring a development environment, prompted the strategic choice to use standard text editors for HTML and CSS coding. Media queries in CSS ensured the site's responsiveness across different screen sizes. Once client approval was obtained, the code was integrated into Intellij, converting HTML to XHTML for JSF compatibility.
The client-side design aimed to balance aesthetics with functionality, offering sections for daily specials, a full menu, upcoming events, a reservation system, and contact information. All content was fetched from the database, with the restaurant owner able to update data through an admin side detailed in a separate chapter. This approach not only ensured that the website remained up-to-date with the restaurant's offerings but also provided a system for highlighting special events and facilitating table bookings, all aimed at enhancing the guest experience.

4.2.2. Administrators page

The MVC design pattern laid the foundation for the administrator’s page’s construction. Hence, the code follows a simple structure. First, we have the XHTML file for the administrator’s page, applying the JSF (JavaServer Faces) API. These facelet files represent the `View`, and support both traditional html as well as xhtml code.

The admin.xhtml file served as the central translation unit for the entire administrator page. It includes all forms and submit buttons, ranging from the A La Carte menu, to employee work shifts, and events. While this approach is not ideal with respect to “clean code”, or separation of concerns, it was nevertheless an effective solution considering the time constraints and imminent deadlines.

Directly interfacing with the JSF file are the `Bean`-classes. These classes represent the `Controller` of the MVC pattern. As controllers, their responsibility is to handle the business logic of the entities. The entities, representing the `Model`, are the Java classes generated from Java Persistence Mapping. They are simply containers for the data associated with the given entity.

The Bean classes receive input from the forms in the XHTML file, and will then assign that data to the corresponding entity. Once this step is completed, the bean class performs the necessary database operation. Using the EntityManager class object, it can execute any CRUD operation that is needed, interfacing directly with the database.

4.3. Applications

Our team developed two distinct applications to streamline operations and enhance customer experience at "Antons Skafferi": a restaurant application and a kitchen application.

Communication between both mobile and web applications was facilitated through a shared database. The GlassFish server within Netbeans hosted the web applications and provided connectivity to the database. Additionally, a RESTful web service was established to expose the database functionalities to Android applications. These Android applications utilized the Retrofit API to interact with the RESTful web service, enabling essential CRUD operations (Create, Read, Update, Delete) on the database. Real-time synchronization capabilities ensured that changes made in the applications were instantly reflected across all devices, maintaining accurate and up-to-date information.

Both applications facilitated efficient order handling processes, allowing staff to create, modify, and track orders seamlessly. Real-time synchronization ensured that order statuses were updated across all devices, improving coordination between front-of-house and kitchen staff.

The focus was on providing staff with the capability to efficiently manage orders and track their status. While the primary function wasn't to update menu items, staff could promptly receive notifications when the kitchen began working on an order and when it was completed. Additionally, staff could show customers their order details, ensuring transparency and enhancing the dining experience.

The streamlined communication between different roles within the restaurant, improved overall efficiency and service delivery. By centralizing critical restaurant operations into

intuitive applications, we empowered "Antons Skafferi" to deliver exceptional service and enhance the overall dining experience for customers.

4.3.1. Restaurant

The restaurant application development process begins with building our API in IntelliJ IDEA through our Payara server, where we define and implement the necessary API endpoints. We then fetch this API to Android Studio using the Retrofit framwork.To establish communication with the Payara Server API, we use Retrofit, a powerful HTTP client, to fetch data in JSON format from the API Endpoint. Retrofit simplifies the process of making HTTP requests and handling responses by providing a high-level abstraction layer.When the data is fetched from the server, our priority is to ensure its integrity and consistency within our application. We map the raw data from the API responses to corresponding Java classes or objects, enabling efficient manipulation and handling of the data using object-oriented principles such as encapsulation, inheritance, and polymorphism.

Adaptation mechanisms played a central role in preparing data for presentation in the application's user interface. By using adapters, especially RecyclerView:s, we transformed the necessary java objects into a format suitable for display (.xml). This process includes transforming data fields, applying formatting rules, and aggregating related information to enhance the user experience and readability.

RecyclerView serves as our main user interface component for efficiently displaying large datasets. Its modular architecture separates data management from view rendering, allowing smooth scrolling and dynamic content loading. By customizing adapters and layout managers, we can tailor RecyclerView to meet our specific use cases and requirements.

Finally, we develop the application's user interface with XML, providing a standardized and flexible structure for web-based content. XML enables the creation of engaging and interactive user interfaces by supporting multimedia content, forms, and styling, ensuring compatibility across different platforms and devices.

4.3.2. Kitchen

As part of our project, we developed dedicated applications for both the serving staff and kitchen personnel of "Antons Skafferi," with a particular focus on optimizing order handling and preparation processes. The kitchen application served as a vital tool in this comprehensive system, designed to efficiently manage incoming orders and facilitate seamless communication between the front-of-house and kitchen teams.

Kitchen Application Development

The development of the kitchen application was centered around several key functionalities. Utilizing the Retrofit library, the application interacted with a shared database managed by the GlassFish server within the Netbeans environment. To gather the necessary information from the database, we implemented a view table that consolidated data from multiple tables, ensuring that only relevant information was transmitted to the kitchen application. This enabled the application to efficiently retrieve orders sent by the waitstaff for preparation, displaying them in a structured manner within the app. Each order, identified by attributes from multiple database tables, underwent sorting and presentation to ensure an efficient cooking flow. The user interface was meticulously crafted to allow kitchen staff to easily distinguish between different types of orders, including appetizers and main courses.
A significant aspect of the kitchen application was its order handling mechanism. Orders were updated as completed by the kitchen personnel only when both main courses and appetizers were served. The web server sends only active orders with uncompleted items to the kitchen application. Consequently, when the kitchen application refreshes its data according to the API, orders with completed statuses automatically disappear. This streamlined the kitchen workflow by focusing attention on active orders that required attention.
Additionally, a dedicated view within the app catered specifically to the kitchen staff, listing active orders in a RecyclerView format. This view provided real-time updates on new or fulfilled orders, ensuring that kitchen personnel always had access to the latest order information. Orders were visually represented as CardItems, with status updates for each order card managed by kitchen personnel.

The collaborative effort invested in the development of the kitchen application significantly enhanced communication and efficiency between the serving staff and kitchen personnel. From the initial design prototypes to user testing and iterative improvements, the development process exemplified our team's dedication to delivering a practical and effective solution to meet the needs of "Antons Skafferi".

5. Resultat

The customer website is designed using XHTML and supported by a local database, which is shared with our administrator website. This architectural choice enables smooth handling and updating of the content on the customer website directly from the administrator interface. Every aspect of the content, except for the layout, is managed and updated through the administrator website, reflecting the latest and most relevant content for our customers.

By employing a local database sharing system, our administrators can easily and efficiently manage and update all content on the customer website. Using the unified interface on the administrator website, they can add new products, adjust prices, modify descriptions, and handle other parts of the website's content in an intuitive and user-friendly manner. When an administrator makes a change on the administrator website, whether it's adding a new description to the menu or updating the price of an existing product, the altered information is synchronized directly with the local database shared with the customer website. This ensures that the latest version of the content is available to customers on the customer website in real-time.

5.1. Customer Page

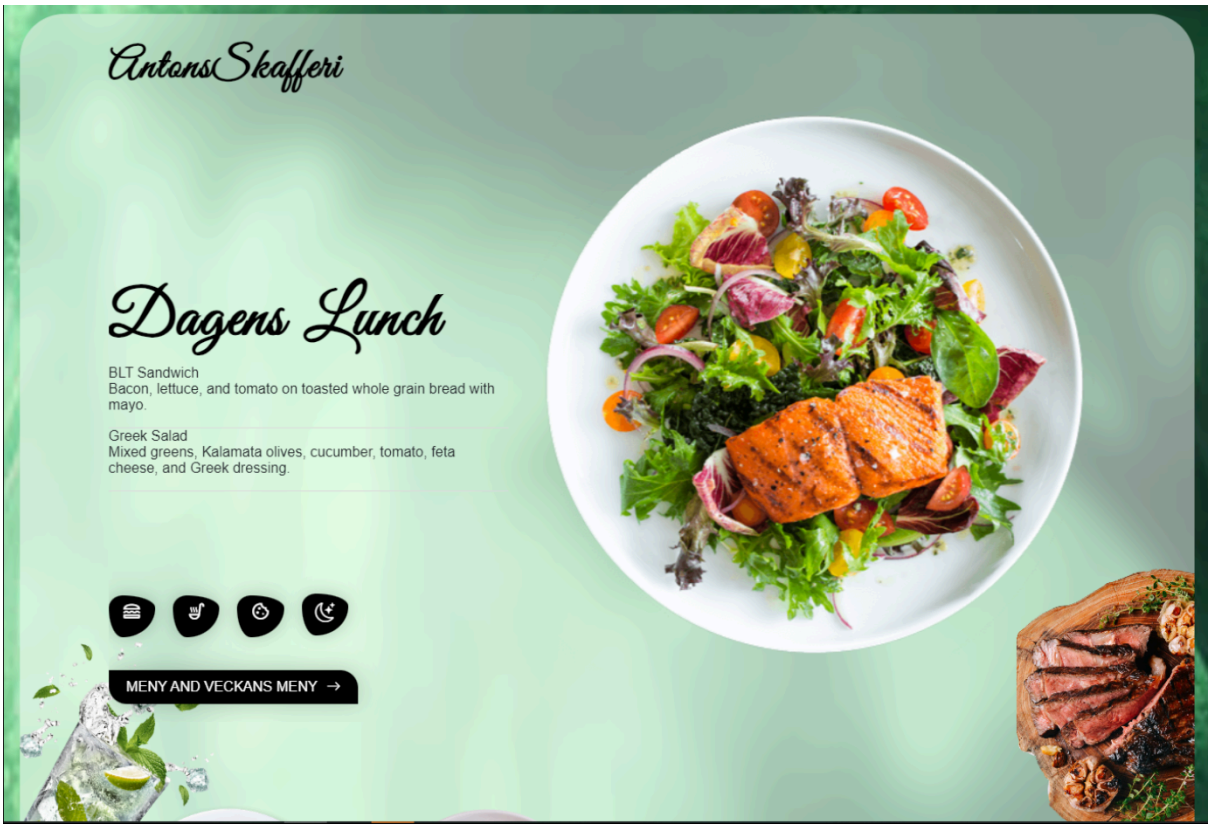


Figure 10: the start page of the customer page

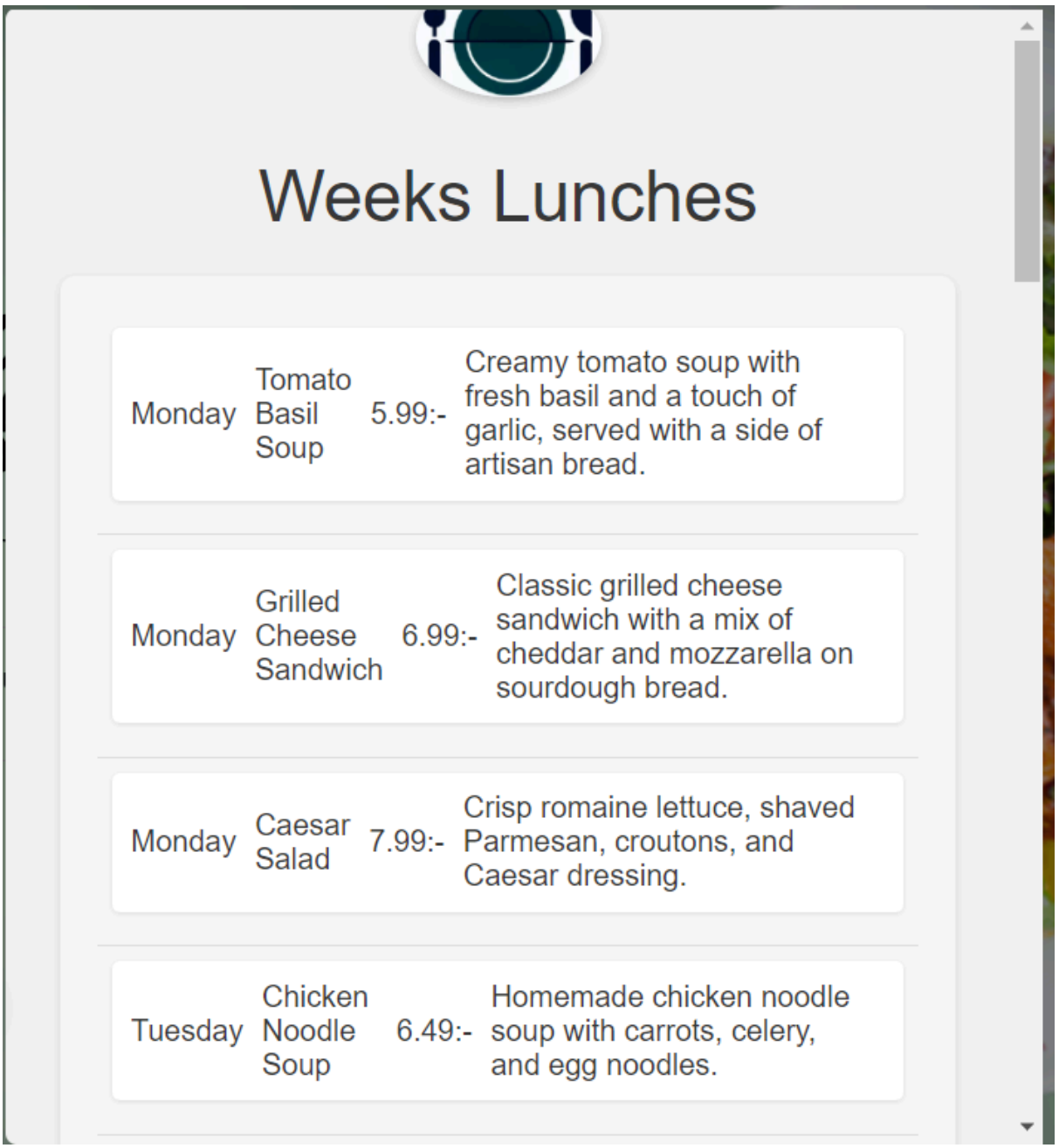


Figure 11: the weekly lunch menu

Figure 10 showcases the “Dagens Lunch” which stands for “Lunch of the day” feature, offering two options: a buffet of warm dishes and a soup menu. Users can access this information by clicking on icons below the 'Lunch of the Day' section. The buffet provides diverse choices for a customizable meal, while the soup menu offers comforting options. Clicking on the icons allows users to explore and select their desired lunch options effortlessly, enhancing their dining experience. However if users wish to view the entire

week's lunch menu, they can press “meny and veckans meny”, positioned below the icons. As depicted in Figure 11, users can then explore the full weekly menu along with prices. The website have the same popup function for the “A La Carte” also.

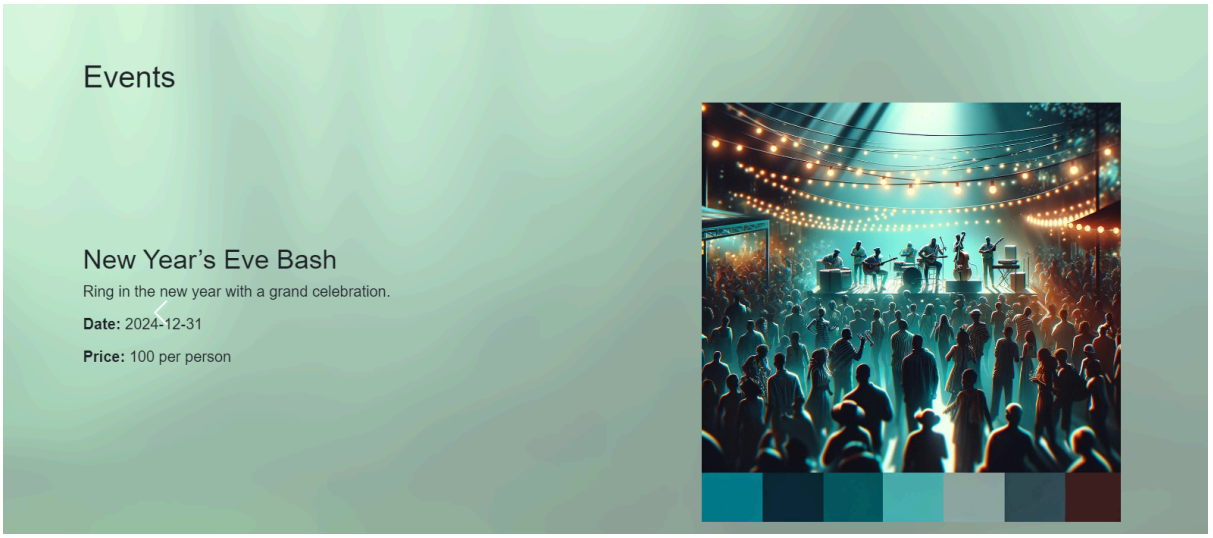


Figure 12: The event list

In the figure 12, a list of available events is displayed. For users interested in exploring and planning for upcoming events, it's easy to do so by clicking on the subtle white arrow faintly visible next to the date in the figure. Utilizing this navigation feature, users can quickly and smoothly browse through the various events and gain an overview of what's planned. This makes it easier for users to stay updated on upcoming events and participate in the activities that interest them the most.

Figur 13: Reservation

Our customers have the opportunity to book a table conveniently directly from our platform. The information needed to complete the booking is clearly presented in Figure Y, making the process accessible and understandable for users. By following the clear steps in the figure, customers can quickly and efficiently enter the name, date, time, month and number of guests for their reservation. But if for any reason the booking cannot be completed, customers need not worry. They still have access to contact information for our staff, either by sending an email or making a phone call. This way, they can easily receive assistance and support from our dedicated team to resolve any issues or answer questions related to the booking process.

5.2. Admin Page

The final result of the administrator's page displays a list of the relevant forms needed to specify the information that is important for the restaurant owner. This includes setting the lunches of the week, all items of the a la carte menu (starters, mains, desserts, and drinks), adding upcoming events, and setting the schedule for employees. Rather than dividing the sections into different views, they are all on the same page. This makes it easy to access through scrolling, with minimal button presses required.

A snapshot of the administrator's website can be seen in figure 14:

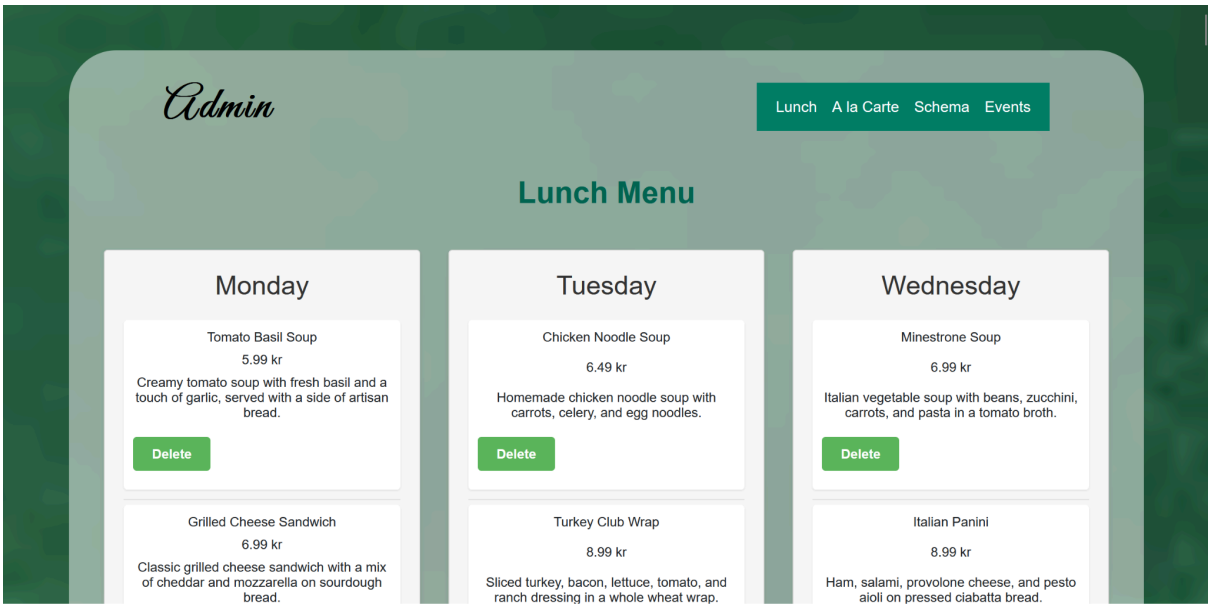


Figure 14: excerpt of the admin page

In figure 14, we can see the first snapshot of the administrator's website. In the upper right corner, there is a menu for quick navigation to each section of the administrator page. The first section is reserved for adding or deleting lunches, which appear as a grid for all the weekdays. Lunches can easily be deleted in case of errors made. Figure 15 shows the form used for adding lunches.

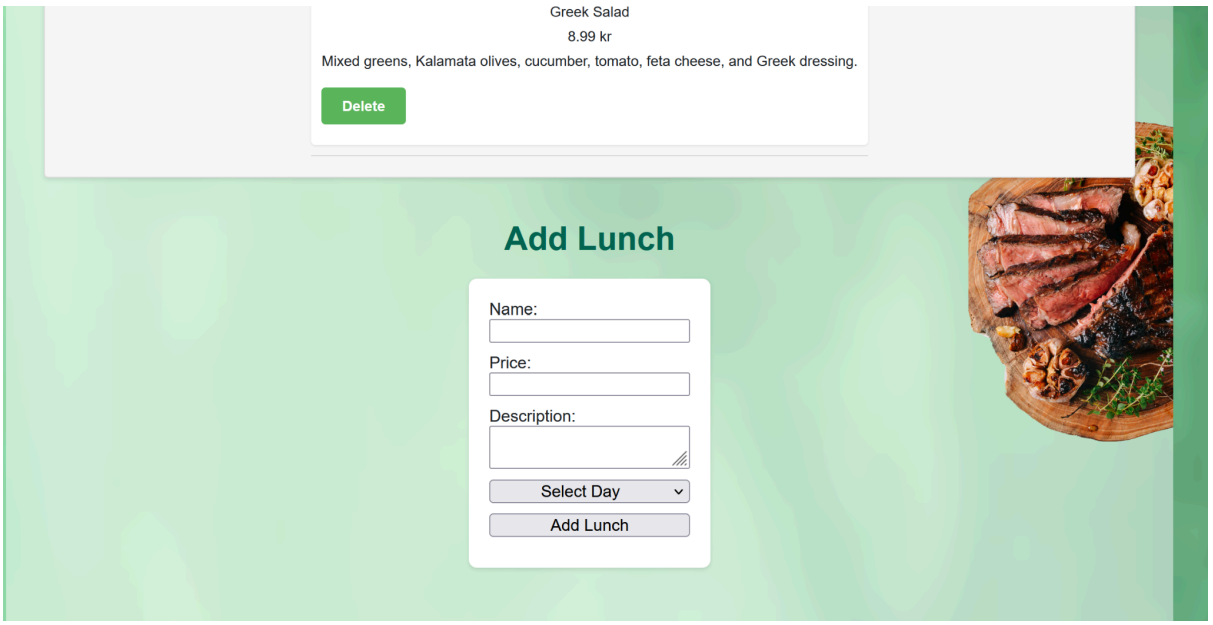


Figure 15: Form used for adding a lunch

Figure 15 illustrates the form used for adding a lunch. By simply filling out the form and selecting the correct day of the week, the lunch will appear under the day in the grid above.

The rest of the page follows the same pattern; the list of items the administrator has added (with the option to delete one of those items), followed by a form to add a new item. For example, figure 16 shows the “Mains” dishes followed by the form used for adding a main course:

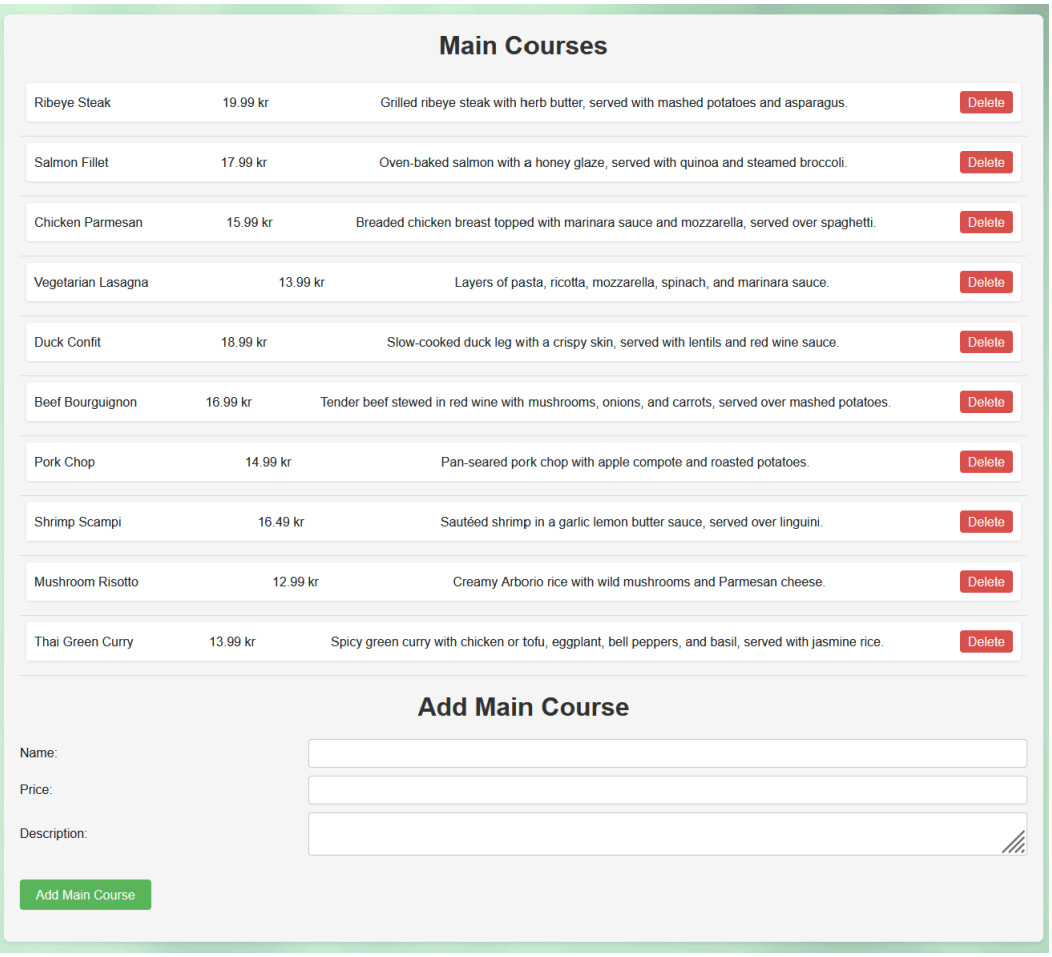


Figure 16, Section for adding and deleting main courses

In figure 16, we can see the section for manually adding or deleting main courses. The same design applies for the rest of the a la carte submenus. Lastly, we have the sections for the employee schedules and events, see figures 17 and 18.

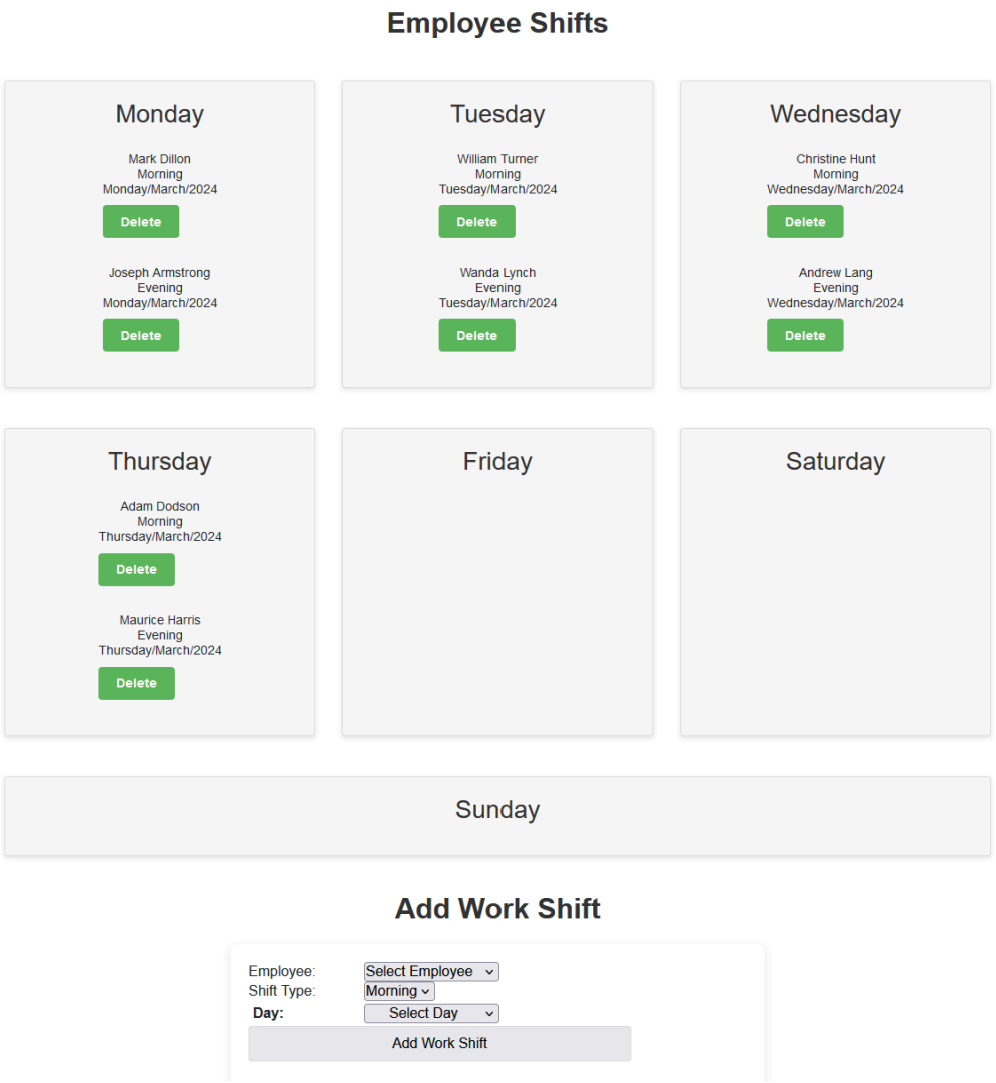


Figure 17, Section for adding work shifts for specific employees

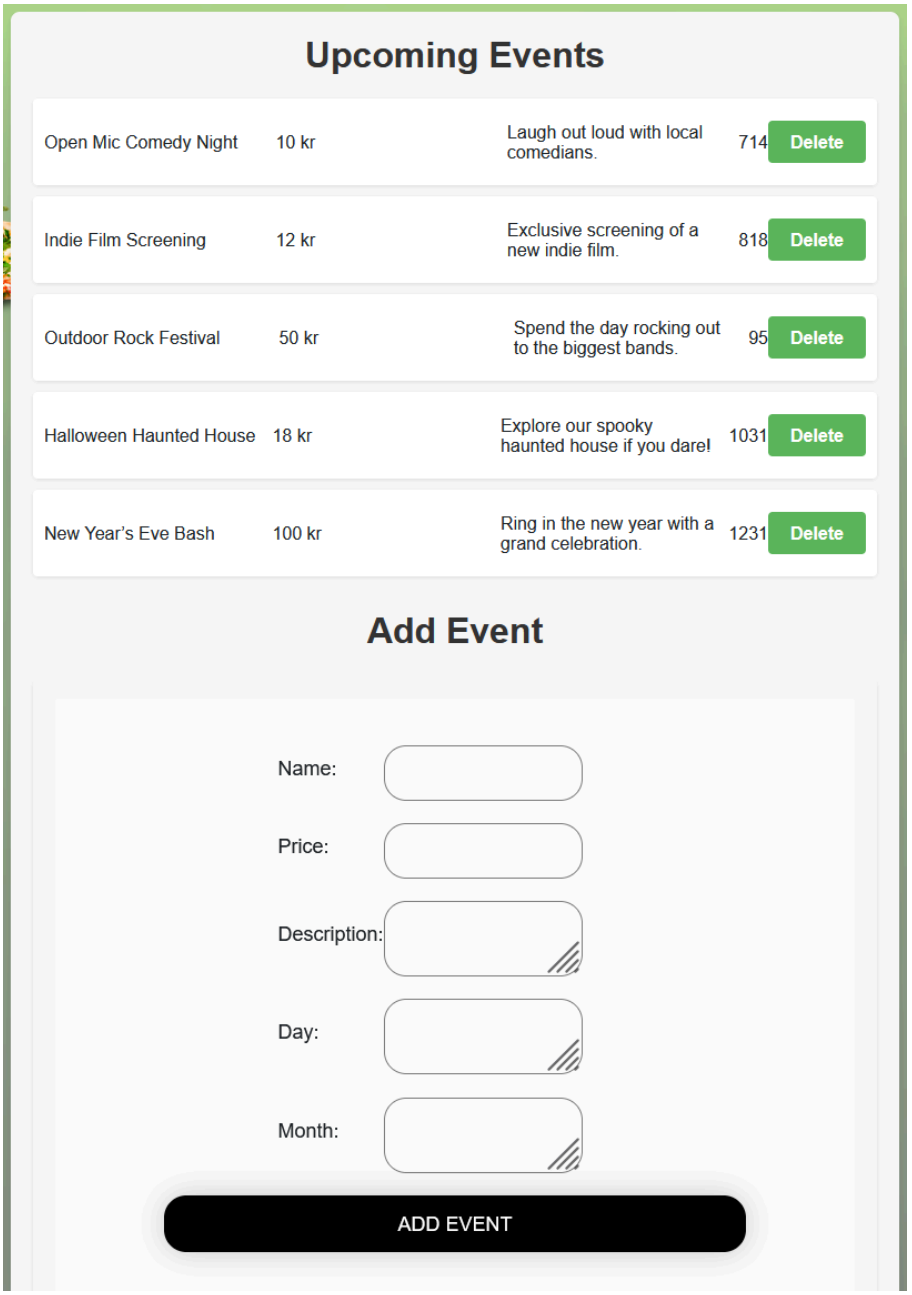


Figure 18, section for managing events.

Due to time constraints, and since the admin page is only intended for the restaurant owner, the primary focus of the admin page was to ensure it was functional. This led to a lack of CSS styling, resulting in a simple design. But while the aesthetics may require some polish given enough time, the administrative tasks themselves work as intended, as all items are fetched and updated instantly. This is because of the successful interface with the database supporting this admin.xhtml web page.

5.3. Restaurant Application

In this section, the outcomes are presented from the deployment of the innovative restaurant staff application designed to streamline the ordering process in a busy restaurant environment. The application is made with a user-friendly interface distributed across three distinct screens or activities, each utilizing a recyclerView to systematically display relevant data to the user. These screens collectively contribute to an efficient ordering workflow, from employee selection to finalizing customer orders.

5.3.1. Employee Activity Screen

The initial screen, named the Employee Activity Screen, is the gateway for staff to interact with the app. It sets out all restaurant employees in a recyclerView, facilitating an effortless selection process. Upon selecting their name, staff members are navigated to the next pivotal screen in the ordering sequence.

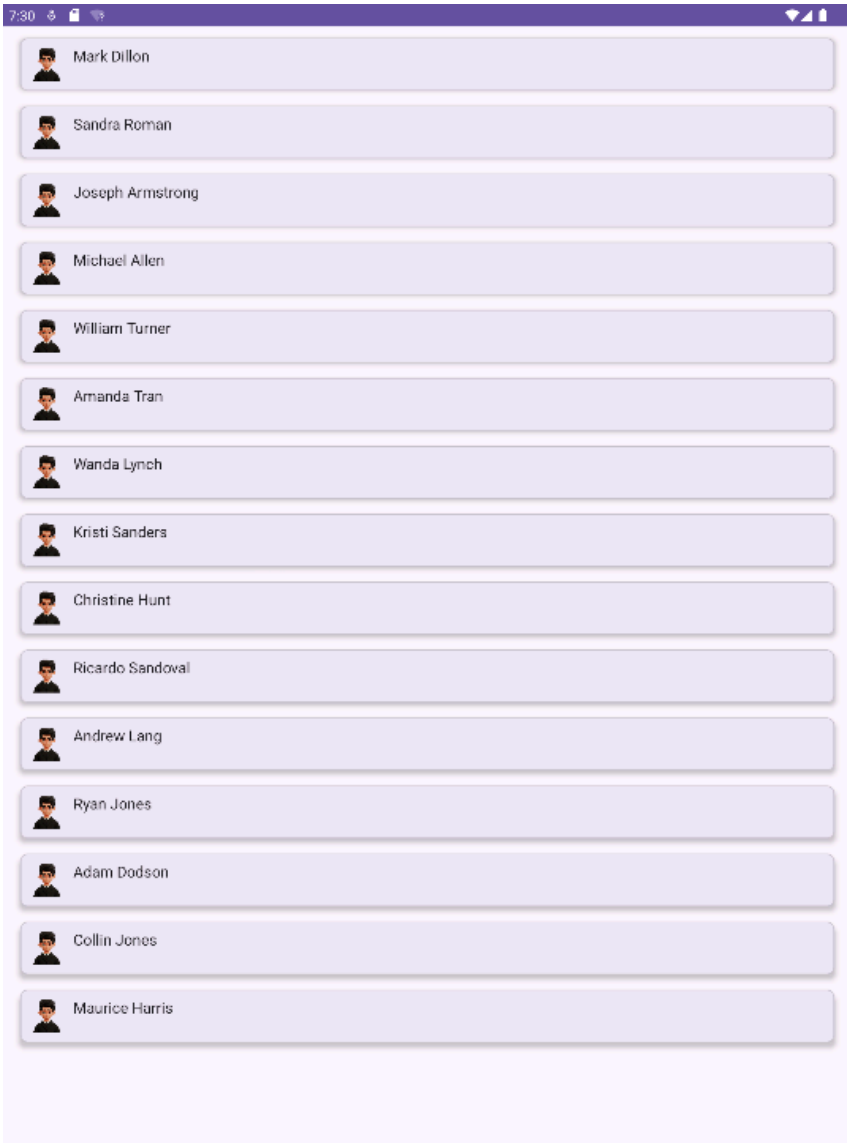


Figure 19: Employee list

Figure 19 illustrates this screen, highlighting the user-friendly layout and the smooth navigation that leads to the subsequent screen upon employee selection.

5.3.2. Table Activity Screen

The second screen, known as the Table Activity Screen, is integral for managing table statuses within the restaurant. It visualizes each table as an individual entity, with the background color serving as a status indicator: green signifies availability, and red denotes occupancy. This boolean color scheme enables staff to quickly assess table availability at a glance. Staff members have the flexibility to update table status through a long click to mark a table as available post-service or a simple click to proceed to the Menu and Order details

screen for the tables. Figure x showcases the intuitive design of this screen, emphasizing the color-coded table statuses and the user interaction mechanisms for updating these statuses.

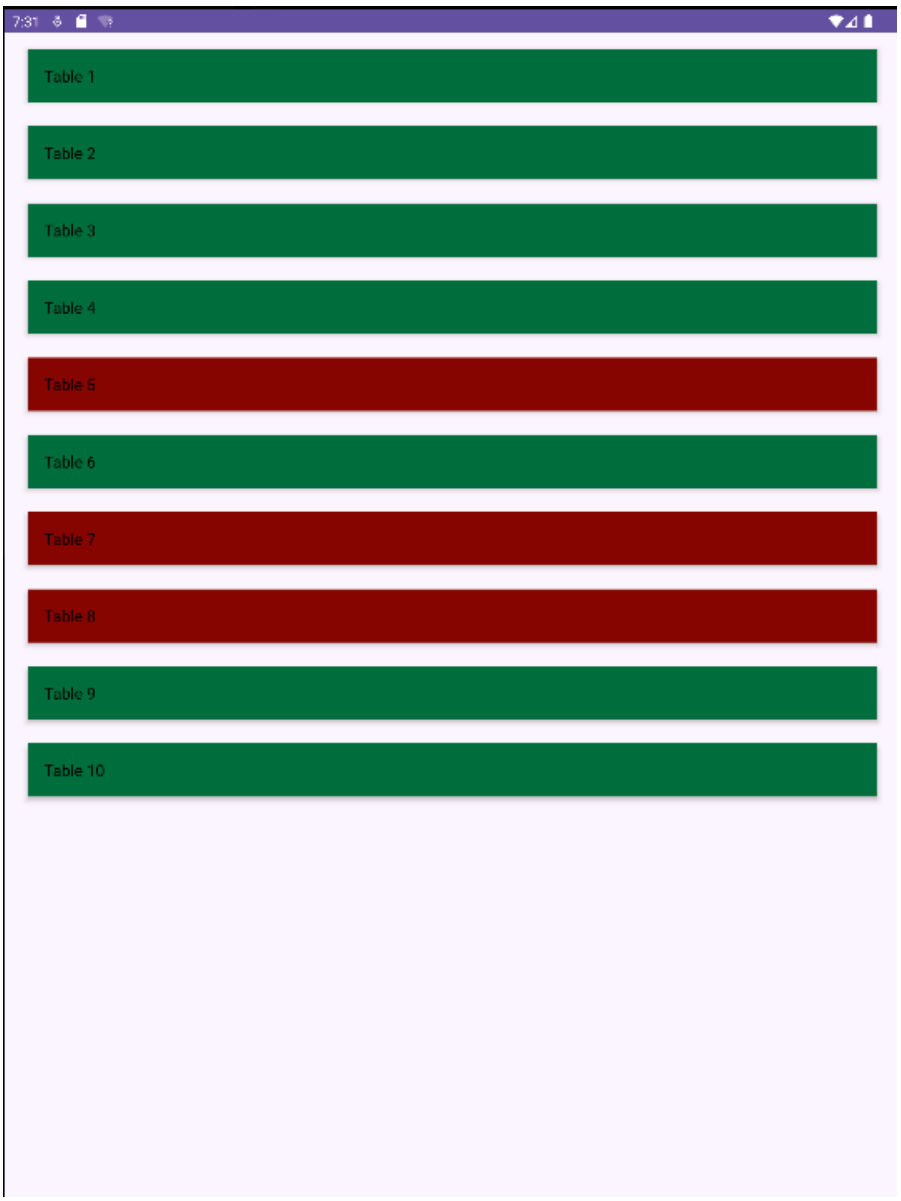


Figure 20: tables list where tables 5,7 and 8 are unavailable

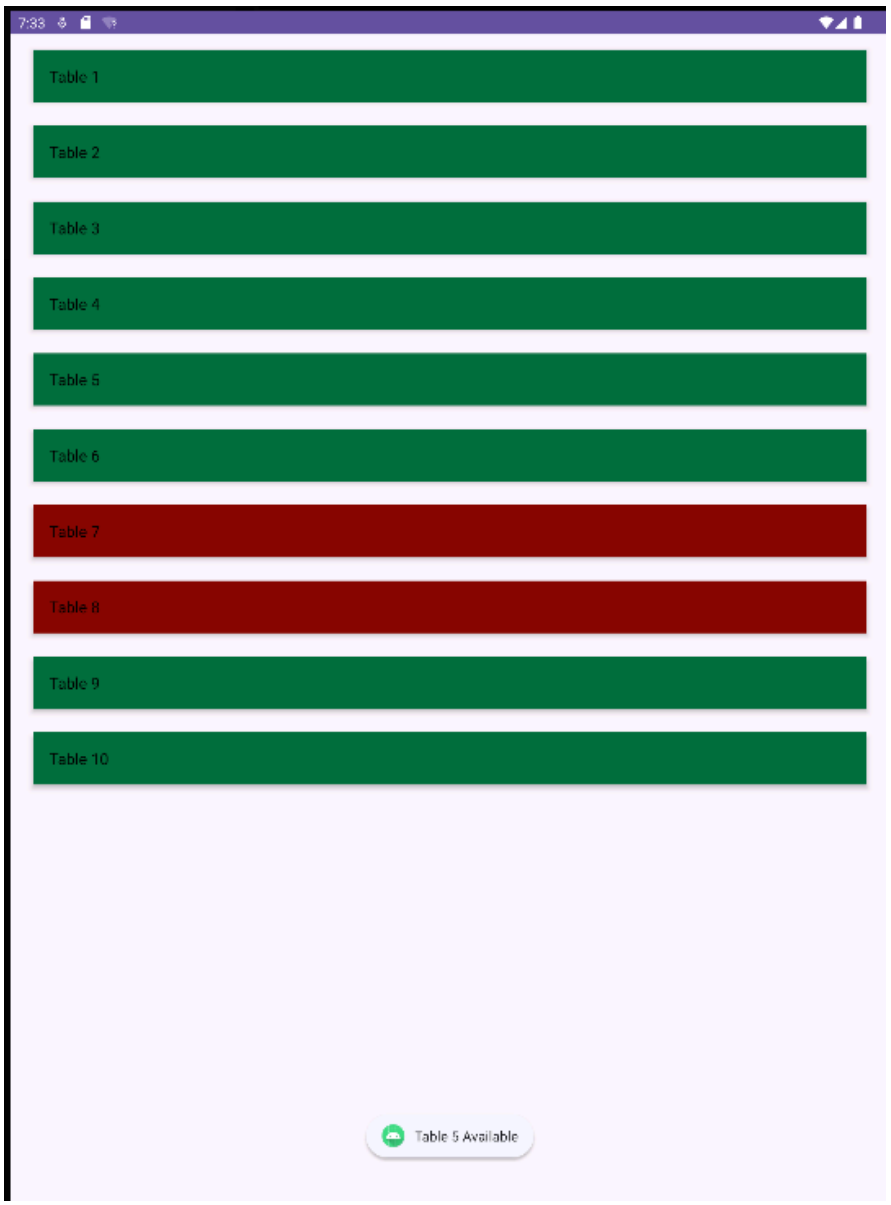


Figure 21: after long clicking on Table 5 to change status and a toast that notifies the change made

Figure 20 and 21 showcases the intuitive design of this screen, emphasizing the color-coded table statuses and the user interaction mechanisms for updating these statuses.

5.3.3. Menu and Orders Activity Screen

The final screen in the application's workflow is the Menu and Orders Activity Screen, which is structured around a parent recyclerView that hosts five distinct headers. These headers

categorize the restaurant's offerings into four à la carte menus and a section for table-specific orders. Interactivity within this screen is highly dynamic; staff can select any item from the à la carte menus, which then populates the orders section for the active table. Completion of an order triggers an update in the database, subsequently altering the associated table's background color from green to red on the Table Activity Screen, indicating the table's occupancy.

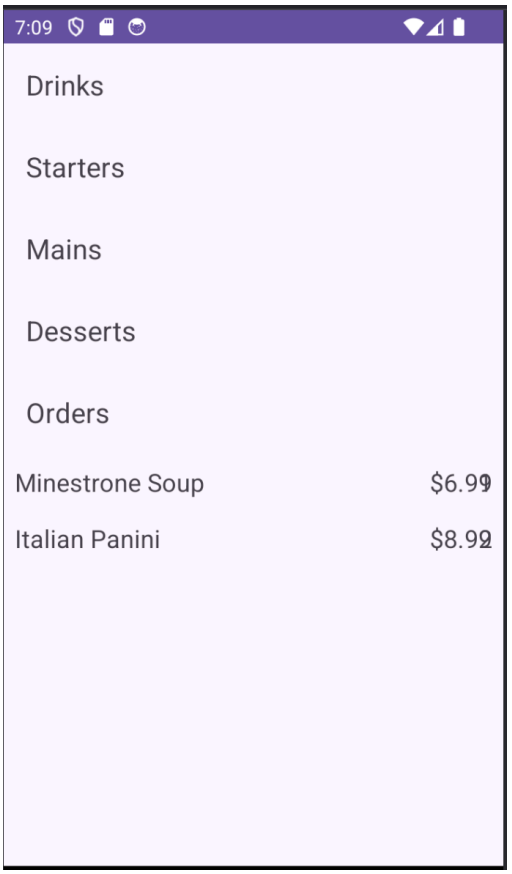


Figure 22: Current orders for table 4

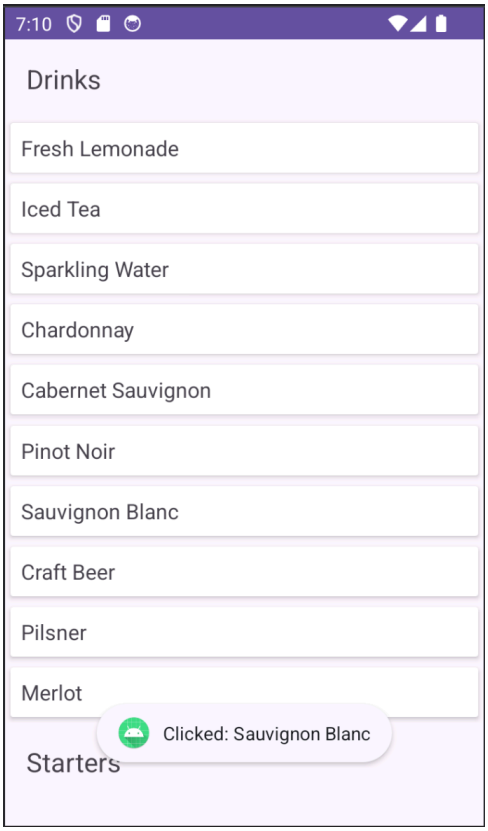


Figure 23: Adding Sauvignon Blanc to table 4's order

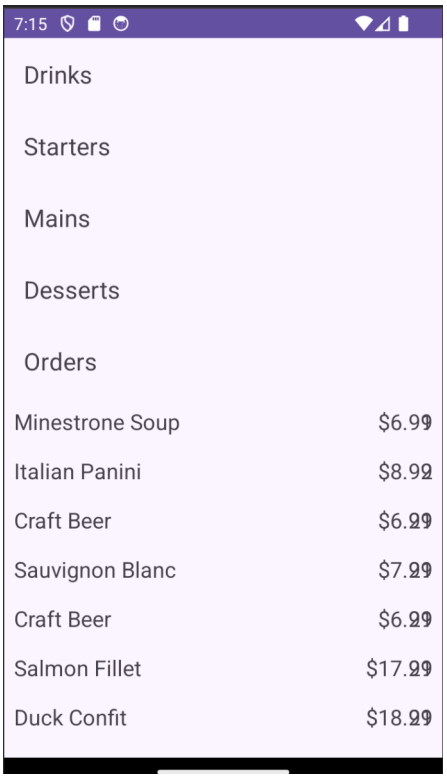


Figure 24: Seeing all the orders made for table 4.

Figure 22 provides a clear visual guide on the user interface of our application, presenting the list of expandable parent headers. These headers allow users to navigate through different categories seamlessly.

Following that, Figure 23 takes us through the process of adding an item from the à la carte menu. It captures the intuitive design of the interface and demonstrates the confirmation mechanism with a toast notification, informing users of the specific item they have selected.

For a more detailed exploration of the functionality, Figures X, Y, and Z, located in Appendix A, depict the procedure by which users can add items from various à la carte lists. These figures collectively illustrate the fluidity and range of choices available within the application.

Concluding this visual journey, Figure 24 showcases the culmination of the user's actions: a comprehensive list of all the orders that have been added. This final list serves as a summary of the user's selections and is an integral part of the order and checkout process within our system.

5.4. Kitchen Application

The implementation of the kitchen application at "Antons Skafferi" is expected to significantly improve order processing, potentially reducing the average time from order placement to service with the implementation of the application. Facilitating a smoother and faster dining experience.

Improvement in Order Accuracy and Customer Satisfaction

The application's design emphasizes clear communication via real-time updates between front-of-house and kitchen staff, as well as notably decreasing order errors. The intuitive interface helps to mitigate common communication pitfalls in a high-paced restaurant environment, where improvement in service quality and customer satisfaction is expected, based on the system's ability to accurately relay order details and updates.

Enhanced Communication Channels

The integration of real-time updates is anticipated to bridge the communication gap between serving and kitchen staff effectively. This feature has the potential to manage customer expectations more accurately and handle peak service times with increased efficiency.

Interface Design and Usability

The kitchen application's interface was crafted for ease of use in a high-paced restaurant setting. It presents orders in a structured manner, prioritizing them as starters or mains. Color-coded status indicators and a clean layout aim to reduce the cognitive load on kitchen staff, ensuring focus on active orders.

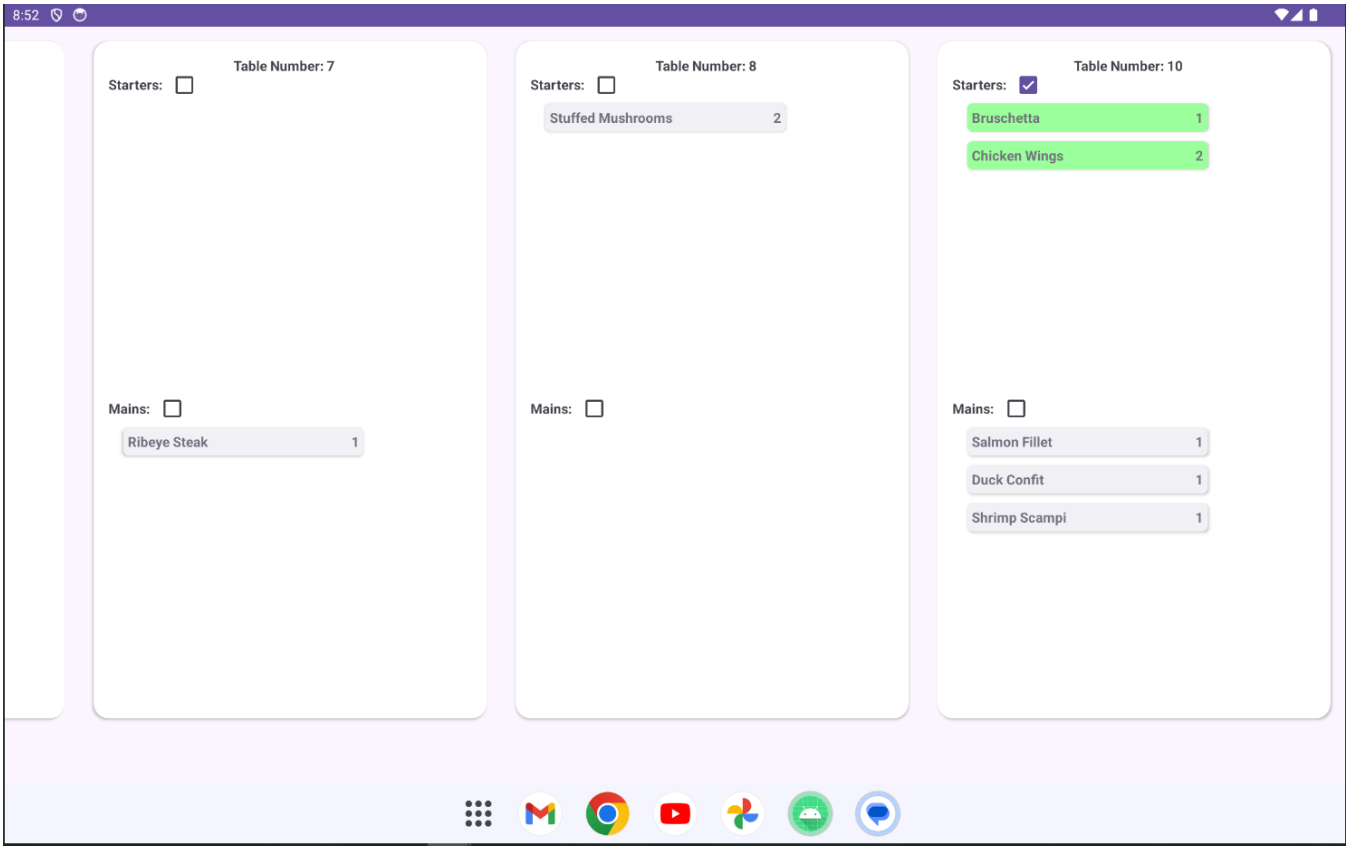


Figure 25: Screenshot of the Kitchen Application Interface

The UI shown in figure 25 showcases a simple and intuitive layout that separates orders into starters and mains for each table, allowing kitchen staff to easily identify and prioritize meal preparation. Orders are presented in real time, with color-coded status indicators to distinguish between pending and completed items. The UI's clean design, featuring clearly labeled sections and a concise presentation of order details, reflects our commitment to creating a practical tool for fast-paced kitchen environments.

5.5. Teamwork

Throughout the project, the group engaged in regular comparisons and discussions about each week, aiming to not only address technical results and issues but also to enhance team dynamics and collaboration. The team placed a strong emphasis on identifying key characteristics that underpin effective teamwork.

To concretize these efforts, the team established a weekly rating system for each identified characteristic, providing a clear metric for improvement in subsequent weeks. These evaluations are visually displayed in the graphs, where the x-axis represents the timeline of the project (week by week) and the y-axis reflects the performance scores (ranging from 1 to 5).

Each line graph tells the story of how the team perceived their collective performance on a particular attribute. A higher score suggests better performance or satisfaction regarding that characteristic for the given week. The visual trend lines serve as a reflection of the group's self-assessment, enabling them to pinpoint strengths and areas for further development as the project progressed.

The collected data and ensuing discussions were integral to the project's iterative improvement process. They provided tangible benchmarks for the team's growth in areas crucial for collaborative success, ultimately contributing to the project's evolution and the team's ability to work together more effectively.

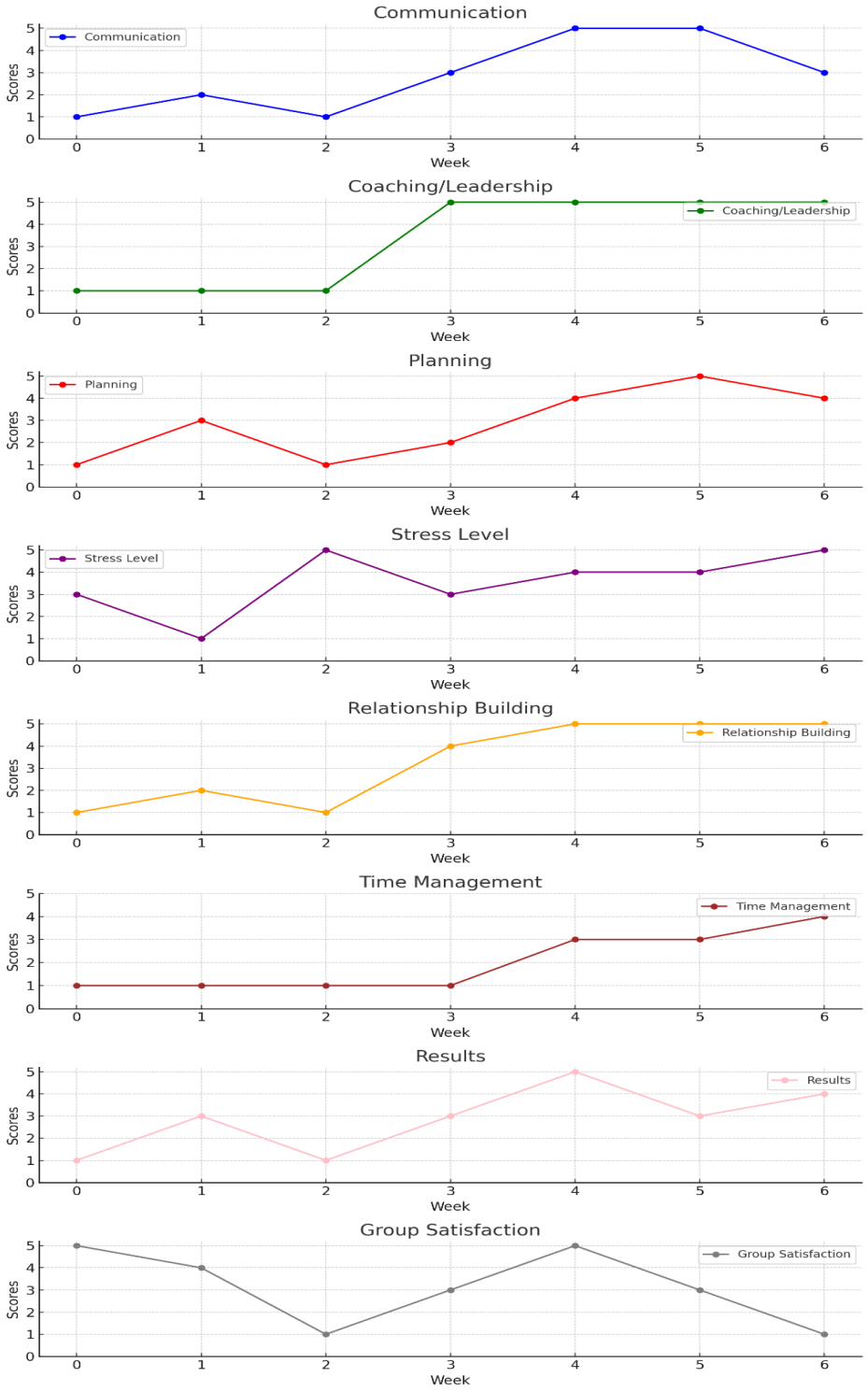


Figure 26: illustrates series of line graphs of teamwork characteristics

Figure x showcases a series of line graphs representing the progression of various teamwork and project management characteristics over a span of six weeks. These characteristics include Communication, Coaching/Leadership, Planning, Stress Level, Relationship Building, Time Management, Results, and Group Satisfaction.

6. Discussion

In this section, we reflect on our project, evaluating both our successes and the challenges we faced. This analysis includes what aspects went well, what did not meet our expectations, and considerations on what could have been approached differently. We also share personal insights into our collective work.

At the outset, our task was to develop a website, segmented into admin and customer pages, alongside three distinct mobile applications: a waiter application, a kitchen application, and a schedule application for employee shifts. We successfully implemented the backend functionality for both the admin and customer sections of the website, with only minor design adjustments remaining due to time constraints.

Regarding the mobile applications, we managed to partially complete two of them. Unfortunately, the third application, intended for managing employee schedules, was not started due to time limitations.

The waiter application saw the implementation of all three essential views. However, both its frontend design and backend functionality could have benefited from further refinement to enhance the overall outcome.

Our development of the kitchen application was significantly hampered by time restrictions, leaving us with just three days for its completion. Consequently, it retains a default design that falls short of our expectations. Despite these design shortcomings, the application's backend functionality meets our basic requirements, even though it was subject to certain limitations detailed earlier in our report. Its simplicity, focused on a singular view and fundamental features, was achieved as planned.

Reflecting on the final results, our sentiments are mixed. Given the constrained timeline and our small team size of five members, we acknowledge a level of satisfaction with the extent of our project's completion. However, it's evident that our aspirations exceeded the finished product.

Despite these challenges, our team dynamic was a notable success. We exceeded our initial expectations in terms of collaboration, maximizing our performance and efficiency, and fostering a positive working environment. This experience has been invaluable, teaching us the importance of teamwork and the collective effort required to navigate and overcome project obstacles. Our journey through this project has not only enhanced our technical skills but also our ability to work, learn, and grow as a unified group.

If we were more members we would have more resources to allocate for finding tools, research and completing tasks. But a bigger team would also lead to a bigger communication complexity. More people would improve innovation and enchanted problem solvings but when teams grow too big it will kill innovation and according to social loafing the individuals will give less effort to the project.

7. Reference

[1] Monash University, ”Citing, Vancouver Style” http://www.lib.monash.edu.au/tutorials/citing/vancouver.html Publicerad 2006-04-13. Hämtad 2012-01-25 (Exempel på referens till webbsida)

[2] Svenska Datatermgruppen, ”Information om datatermer”, http://www.nada.kth.se/dataterm/ Publicerad 1998-08-20. Hämtad 2005-04-11. (Exempel på referens till webbsida)

[3] De Busscher, Rudy. 2021. “Getting Started with Jakarta EE 9: How to Create a REST API with Jakarta EE 9.” Payara Blog. https://blog.payara.fish/getting-started-with-jakarta-ee-9-how-to-create-a-rest-api-with-jakarta-ee-9.

[4] “Retrofit.” n.d. Square Open Source. Accessed March 10, 2024. https://square.github.io/retrofit/.

[5] “Jakarta Faces.” n.d. YouTube: Home. Accessed March 10, 2024. https://jakarta.ee/specifications/faces/4.0/jakarta-faces-4.0.

[6] GitHub web application link: https://github.com/ybond93/Website

[7] GitHub restaurant application link: https://github.com/ybond93/Application_Restaurant

[8] GitHub kitchen application link: https://github.com/ybond93/Application_Kitchen

[9] Robbins, S. P., & Judge, T. A. (2021). Organizational Behavior (18th ed.). Pearson.

[10] A. Author, "Retrofit Library in Android," Topcoder, [March 2024]. [Online]. Available: https://www.topcoder.com/thrive/articles/retrofit-library-in-android. [Accessed: 11-03-2024]

[11] "Eclipse GlassFish," Eclipse Projects, [Online]. Available: https://projects.eclipse.org/projects/ee4j.glassfish. [Accessed: 11-03-2024].

[12] Hejlsberg, Anders. n.d. “Object–relational mapping.” Wikipedia. Accessed March 12, 2024. https://en.wikipedia.org/wiki/Object%E2%80%93relational_mapping.

[13] “What is a REST API?” n.d. IBM. Accessed March 12, 2024. https://www.ibm.com/topics/rest-apis.

[14] “Data transfer object.” n.d. Wikipedia. Accessed March 12, 2024. https://en.wikipedia.org/wiki/Data_transfer_object.

8. Appendix A

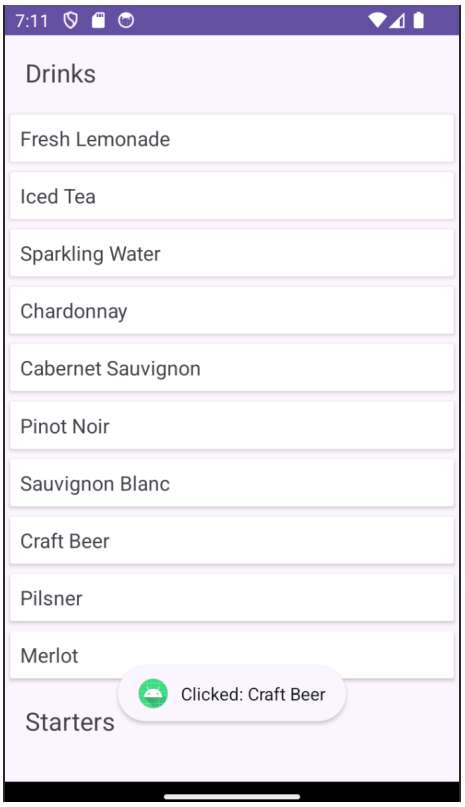


*Figure X: adding **two** craft beers for table 4's order*

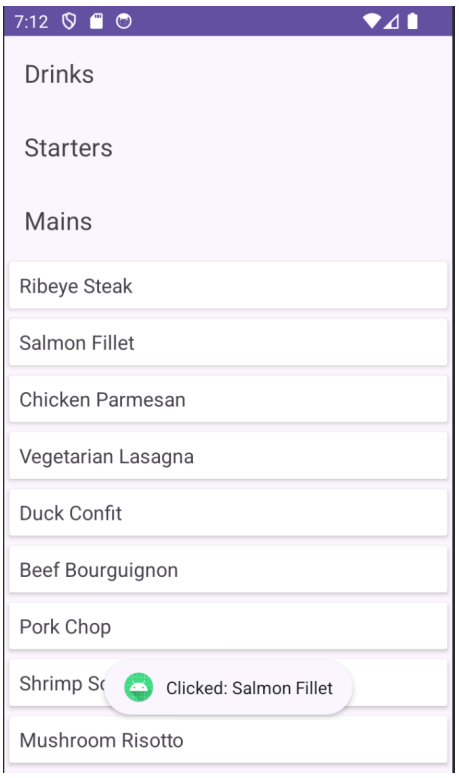


Figure Y: Adding a Salmon Fillet to table 4's order

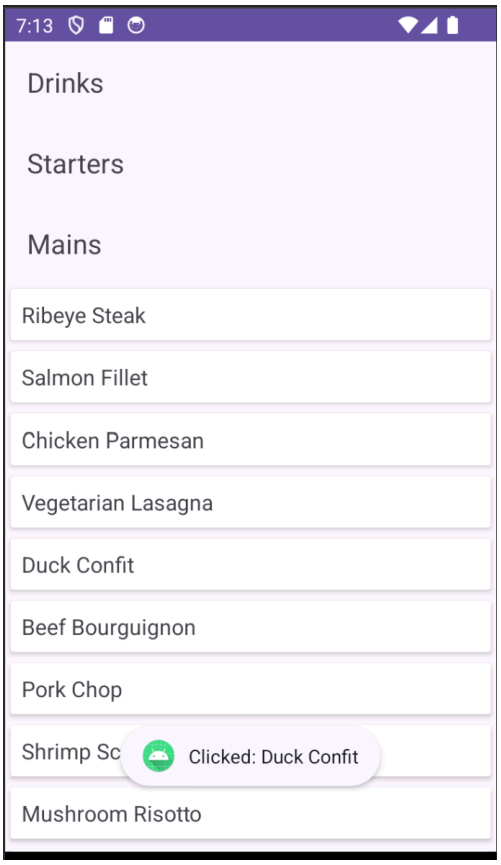


Figure Z: Adding a Duck Confit to table 4's order

9. Appendix B

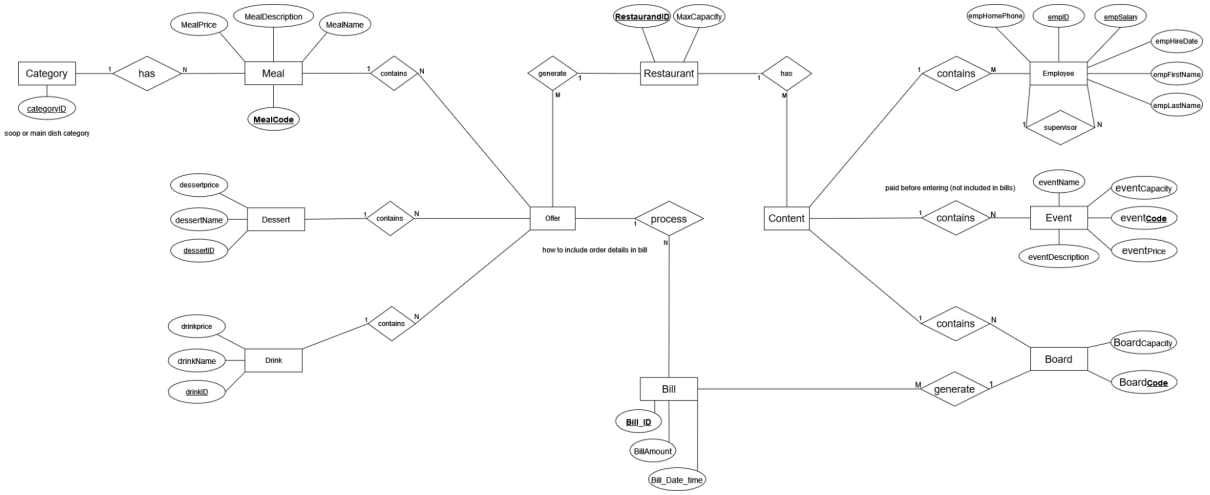


figure x: image of a d

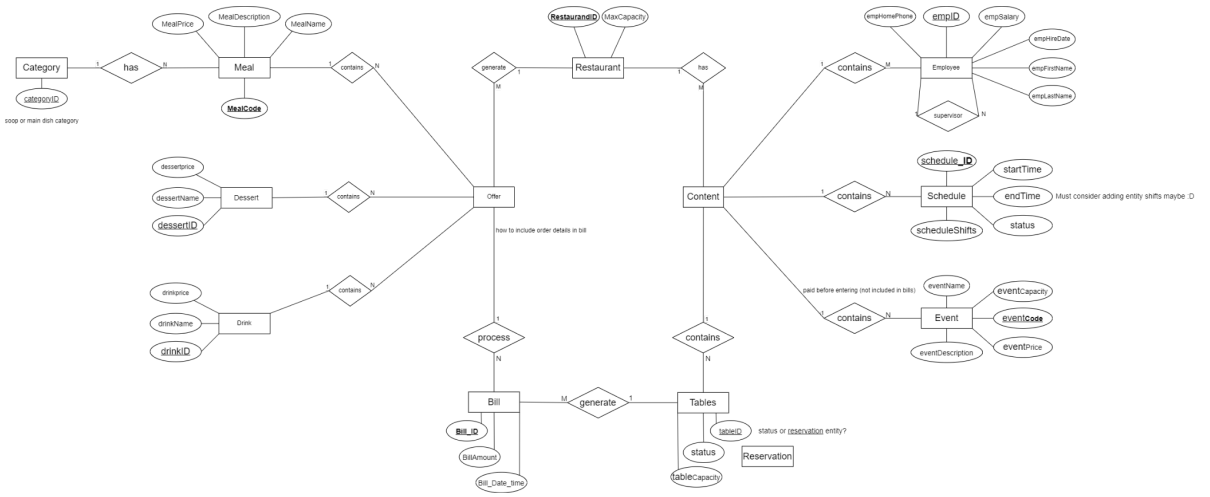


figure x:

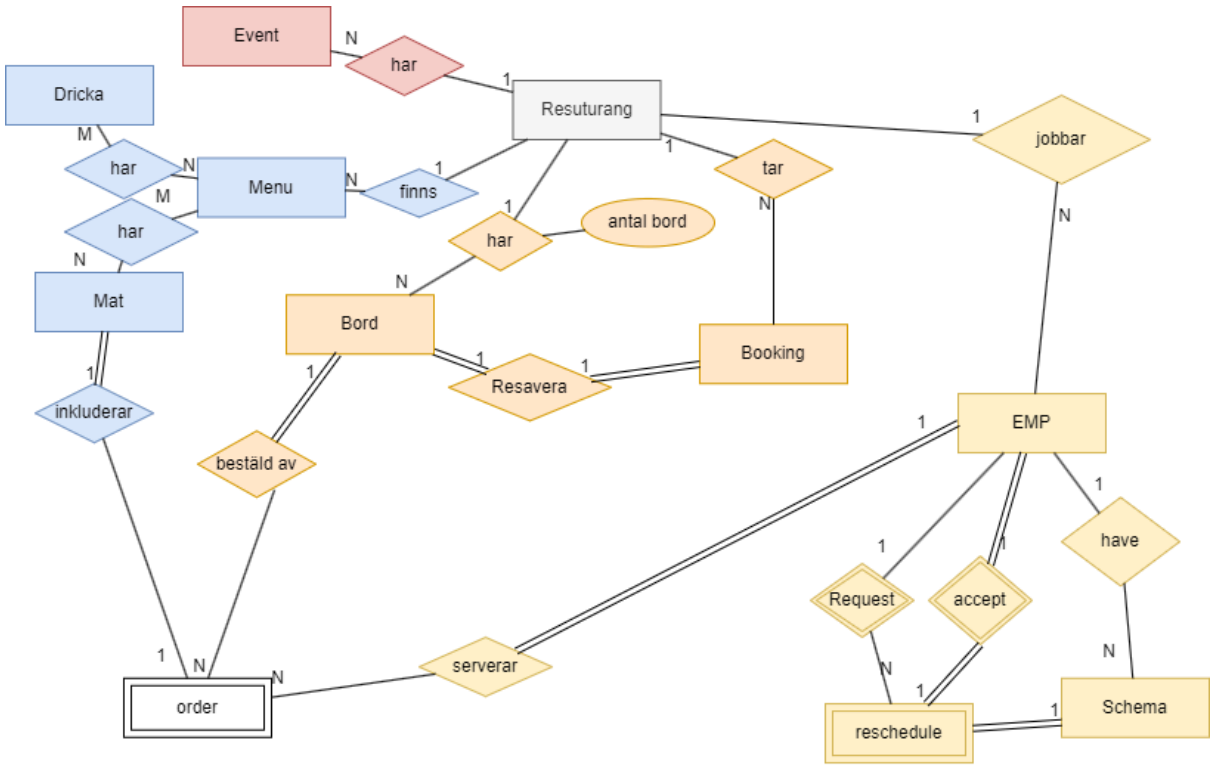

Event
har
Dricka
Resuturang
jobbar
har
Menu
finns
tar
har
antal bord
Mat
Bord
Booking
Resavera
inkluderar
beställ av
EMP
have
Request
accept
serverar
order
reschedule
Schema

figure x: