

Project documentation

**Computer Science**

DishDash - meal planner

Course: Computer Science

Team Members:

Wiktoria Duraj

Filip Bajbak

Jakub Ryszczyk

Location: Gliwice

Academic Year: 2024/2025

Contents

[**Team information** 2](#_Toc201401197)

[**0.1 Roles** 2](#_Toc201401198)

[**0.2 Planned Tasks** 2](#_Toc201401199)

[**0.3 Tasks actually completed** 2](#_Toc201401200)

[**0.4 Contribution** 3](#_Toc201401201)

[**1 Project goal** 3](#_Toc201401202)

[**Project name: DishDash meal planner** 3](#_Toc201401203)

[**2 Project requirements** 4](#_Toc201401204)

[**2.1 Technical and Non-Technical Requirements** 4](#_Toc201401205)

[**What our project must have?** 4](#_Toc201401206)

[**What our project could have?** 4](#_Toc201401207)

[**2.2 SWOT** 5](#_Toc201401208)

[**2.3 Expected Project Results** 5](#_Toc201401209)

[**3 Design overview** 6](#_Toc201401210)

[**3.1 Project Management** 6](#_Toc201401211)

[**3.2 Methodology** 6](#_Toc201401212)

[**3.3 Back-End** 6](#_Toc201401213)

[**3.4 Front-End** 6](#_Toc201401214)

[**3.5 Database** 6](#_Toc201401215)

[**4.1 Project Management** 7](#_Toc201401216)

[**4.1.1 Jira** 7](#_Toc201401217)

[**4.2 Management** 8](#_Toc201401218)

[**Sprint 1** 8](#_Toc201401219)

[**Sprint 2** 8](#_Toc201401220)

[**Sprint 3** 8](#_Toc201401221)

[**Sprint 4** 9](#_Toc201401222)

[**Sprint 5** 9](#_Toc201401223)

[**4.3 Back-End** 10](#_Toc201401224)

[**Libraries:** 10](#_Toc201401225)

[**Classes / Modele:** 10](#_Toc201401226)

[**Functions:** 10](#_Toc201401227)

[**Global variables:** 10](#_Toc201401228)

[**Design Pattern:** 10](#_Toc201401229)

[**4.4 Front-End** 11](#_Toc201401230)

[**Libraries:** 11](#_Toc201401231)

[**Components:** 11](#_Toc201401232)

[**Functions:** 11](#_Toc201401233)

[**Design Pattern:** 11](#_Toc201401234)

[**4.5 Database** 12](#_Toc201401235)

[**Tables:** 12](#_Toc201401236)

[**Relationships Between Tables:** 12](#_Toc201401237)

[**5.1 Results** 13](#_Toc201401238)

[**5.2 Code Example** 17](#_Toc201401239)

[**6 Deployment and Maintenance** 18](#_Toc201401240)

[**7 Results** 18](#_Toc201401241)

[**Observations** 18](#_Toc201401242)

[**Achievements** 18](#_Toc201401243)

[**Development Potential** 19](#_Toc201401244)

**Team information**

Division of roles, tasks and contribution of each team member.

**0.1 Roles**

* Jakub Ryszczyk – Back-End Developer, Code Reviewer
* Filip Bajbak - Front-End Developer, Tester
* Wiktoria Duraj - Scrum Master, Analytics, Presentation, Database

**0.2 Planned Tasks**

* Jakub Ryszczyk –
  + Create and configure the backend project structure
  + Implement API endpoints for meal, ingredient, fridge, and calendar management
  + Set up database structure and relations
  + Handle authentication and authorization logic
  + Conduct code reviews
* Filip Bajbak –
  + Create the frontend application using Vue.js
  + Implement views for login, registration, meals, fridge, recipes, and calendar
  + Integrate the frontend with the backend API
  + Perform application testing and usability checks
* Wiktoria Duraj -
  + Manage the Scrum process and sprint planning
  + Prepare project documentation and system analysis
  + Design and document the database schema
  + Create presentations summarizing sprint results and final project delivery
  + Monitor Jira board, assign tasks, and track progress

**0.3 Tasks actually completed**

* Jakub Ryszczyk –
  + Successfully implemented backend endpoints for all key features
  + Set up JWT-based authentication
  + Created database tables and handled migrations
  + Integrated meal management, fridge, and calendar logic
  + Reviewed frontend-backend integration and API functionality
* Filip Bajbak –
  + Connected the frontend with backend API endpoints
  + Implemented dynamic calendar, meal selection, and fridge views
  + Tested application functionality and reported integration issues
* Wiktoria Duraj –
  + Managed sprint meetings and task prioritization
  + Created full project documentation and final report
  + Designed the database schema and reviewed its implementation
  + Prepared presentation materials for project demonstrations
  + Oversaw Jira board, task assignment, and workflow management

**0.4 Contribution**

Contribution of each team member to the execution of the project:

* Jakub Ryszczyk – 44%
* Filip Bajbak - 23%
* Wiktoria Duraj - 33%

**1 Project goal**

**Project name: DishDash meal planner**

**The Project Goal** is to implement a unique meal planning and recipe management application.  
To create an application for users to store recipes, plan their weekly meals, and easily manage their cooking schedule.

We aim to implement several features that are not commonly available in existing apps on the market, such as:

* an intuitive meal calendar for assigning recipes to specific days
* the ability to categorize and search recipes by meal type, preparation time, or ingredients
* a personal recipe database fully customizable by the user

The project will include a clean, user-friendly interface and an object-oriented backend connected to a reliable database for storing and managing recipes and meal plans.

**2 Project requirements**

**2.1 Technical and Non-Technical Requirements**

**📌 What our project must have?**

* **registration, logging**  
  → User account creation and authentication system
* **recipe database**  
  → Basic storage of recipe information: title, ingredients, preparation steps, and category
* **adding, editing, deleting recipes**  
  → Full CRUD (Create, Read, Update, Delete) functionality for recipes
* **meal plan calendar**  
  → Ability to assign recipes to specific days of the week
* **viewing and managing meal plans**  
  → Display current weekly plan and easily replace or remove recipes
* **managing fridge stock** → Display current state for the contents of the fridge

**📌 What our project could have?**

* **recipe detailed view with images and preparation time**  
  → Display additional recipe info like images, difficulty level, and estimated cooking time
* **shopping list generator**  
  → Automatically generate a shopping list based on selected recipes for the week
* **user friend list and meal plan sharing**  
  → Add friends and share meal plans or favorite recipes
* **recipe rating system**  
  → Allow users to rate and comment on recipes
* **nutrition facts integration**  
  → Optional nutrition info per recipe (calories, macros)
* **notifications for planned meals**  
  → Reminders for upcoming planned meals

**2.2 SWOT**

|  |  |
| --- | --- |
| **Strengths** | **Weaknesses** |
| Fully customized, user-controlled meal planner | No established brand awareness |
| Clean, modern, and responsive UI built with Vue.js | Limited initial user base |
| Backend flexibility with self-hosted PostgreSQL and Node.js | No dedicated marketing or outreach strategy |
| Supports meal ingredients, fridge inventory, and calendar scheduling | Limited resources for professional UX/UI optimization |
| Clear separation of frontend and backend via Dockerized services | App scalability untested under higher traffic |
| Potential to quickly extend features (nutrition tracking, AI recommendations) | No mobile-native app version yet |

|  |  |
| --- | --- |
| **Opportunities** | **Threats** |
| Growing demand for personalized meal planning and healthy eating tools | Competition from established meal planning and diet apps |
| Integration of AI-powered suggestions and nutrition analytics | Data privacy and security risks (especially with personal meal data) |
| Gamification: cooking streaks, calorie goals, community challenges | Risk of negative reviews if early bugs or UI flaws appear |
| Partnering with local chefs, dieticians, or food influencers | Shifting tech trends and evolving user expectations |
| Expanding to support shopping list generation and fridge auto-restocking | Open-source forks or clone apps diluting your audience |

**2.3 Expected Project Results**

Creating a **desktop or web-based application** capable of safely gathering user-generated recipe and meal plan data, storing it securely in a structured database, and presenting it through a clean, user-friendly interface.

The application will allow:

* Adding, editing, deleting, and viewing recipes
* Planning meals for specific days of the week
* Viewing an organized, calendar-based weekly meal plan
* Refrigerator content management

Future iterations may introduce:

* Meal plan sharing options
* Nutrition data integration
* Recipe ratings and comments
* Automatic shopping list generation

**3 Design overview**

Project related IT tools and systems.

**3.1 Project Management**

* Jira – basically for everything
* GitHub

**3.2 Methodology**

* Scrum

**3.3 Back-End**

* Node.js
* Express.js
* REST API
* JSON

**3.4 Front-End**

* Vue 3
* Vite
* JavaScript
* Fetch API
* CSS3

**3.5 Database**

* Type of database – Relational Database
* Postressql
* dbdiagram.io – ERD for database

**4 Detailed design**

**4.1 Project Management**

**4.1.1 Jira**

In the project, **Jira** was used for task management and sprint organization.

* The project board included:
  + **Columns:**
    - Backlog
    - To Do
    - In Progress
    - Code Review
    - Done
* **Sprints:**
  + **Sprint 1** — started at the beginning of the project work
  + **Sprint 2** — User authentication and login functionality
  + **Sprint 3** — Meal and ingredient management
  + **Sprint 4** — Meal calendar implementation
  + **Sprint 5** — Optimizations and MVP improvements

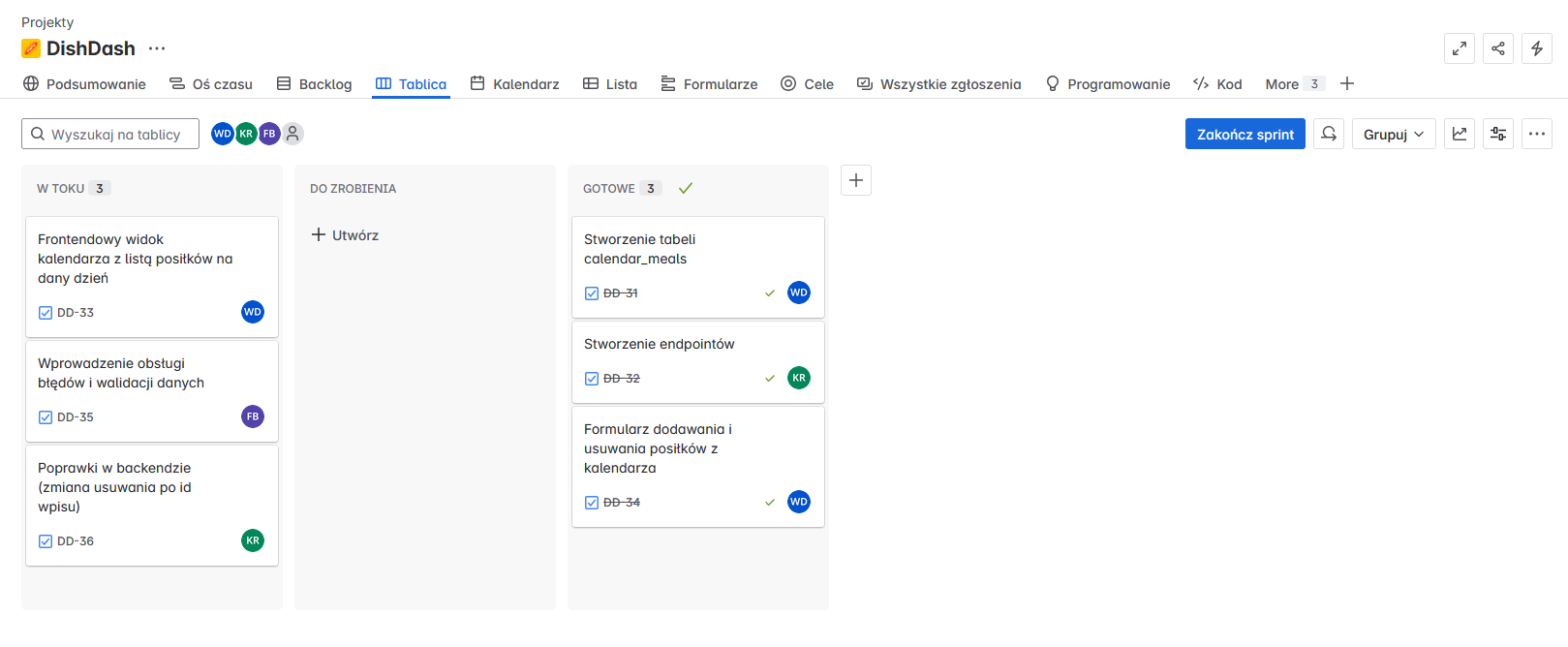


Figure 1- Jira Sprint Board

**4.2 Management**

**Methodology:** Scrum

The project was developed using the Scrum methodology. Work was divided into five sprints, each with defined goals, duration, and working mode. Regular stand-ups and sprint reviews were held to monitor progress and improve workflow.

**Sprint 1**

* **Duration:** 4 hours
* **Tasks:**
  + Set up the backend project structure
  + Configure Docker and PostgreSQL
  + Create initial database schema with migrations
  + Implement basic API endpoints (e.g., user registration, login)
* **Stand-up:** At the start of the session
* **Mode:** Pair programming

**Sprint 2**

* **Duration:** 6 hours
* **Tasks:**
  + Build the frontend application using Vue.js
  + Set up routing and authentication handling
  + Integrate frontend with backend via REST API
  + Implement meal listing and add-to-calendar functionality
  + Perform integration tests between frontend and backend
* **Stand-up:** At the start of the session
* **Sprint Review:** Final 15 minutes of the session

**Sprint 3**

* **Duration:** 6 hours
* **Tasks:**
  + Extend the calendar functionality (fetch meals by date)
  + Implement meal removal from the calendar
  + Build views for adding and managing ingredients (fridge feature)
  + Improve error handling and feedback messages in the UI
* **Stand-up:** At the start of the session
* **Sprint Review:** Final 15 minutes of the session

**Sprint 4**

* **Duration:** 6 hours
* **Tasks:**
  + Implement meal-ingredient relations and add ingredients management
  + Polish the calendar view, adding date pickers and UI refinements
  + Secure the backend routes with JWT-based authentication
  + Conduct functional and security tests
* **Stand-up:** At the start of the session
* **Sprint Review:** Final 15 minutes of the session

**Sprint 5**

* **Duration:** 6 hours
* **Tasks:**
  + Final bug fixing and UI improvements
  + Write project documentation
  + Prepare and run final acceptance tests
  + Project deployment on Dockerized environment
  + Final project presentation
* **Stand-up:** At the start of the session
* **Sprint Review & Retrospective:** Last 30 minutes of the session

**4.3 Back-End**

**Libraries:**

* **Node.js**
* **Express.js**
* **CORS**
* **JWT**
* **Baza danych SQL** (SQLite / PostgreSQL / MySQL — w zależności od środowiska)

**Classes / Modele:**

* User
* Meal
* CalendarEntry

**Functions:**

* authMiddleware — autoryzacja po JWT
* getMeals() — pobieranie listy posiłków
* getCalendar(date) — pobieranie zaplanowanych posiłków dla danego dnia
* addMealToCalendar(mealId, date) — dodawanie posiłku do kalendarza
* removeMealFromCalendar(mealId, date) — usuwanie posiłku z kalendarza

**Global variables:**

* JWT\_SECRET
* DATABASE\_URL
* PORT

**Design Pattern:**

* **Repository Pattern** — dla warstwy dostępu do danych
* **MVC (Model-View-Controller)** — przy rozdziale logiki w backendzie

**4.4 Front-End**

**Libraries:**

* **Vue 3 (Composition API, <script setup>)**
* **Vite**
* **Fetch API**
* **CSS3 z custom variables**
* **Local Storage** (przechowywanie tokenów JWT)

**Components:**

* Login.vue
* Register.vue
* Calendar.vue
* MealList.vue

**Functions:**

* authFetch() — funkcja fetch z tokenem JWT w nagłówku
* fetchCalendarMeals() — pobieranie posiłków na dany dzień
* fetchAllMeals() — pobieranie wszystkich dostępnych posiłków
* addToCalendar() — dodawanie posiłku do wybranego dnia
* removeFromCalendar() — usuwanie posiłku z kalendarza

**Design Pattern:**

* **Component-based architecture** — Vue 3
* **Composition API**

**4.5 Database**

The application uses a **PostgreSQL** relational database consisting of **6 main tables**:

**Tables:**

* **users** — stores application users with unique usernames and hashed passwords.
* **meals** — contains meal information such as name, description, calories, and creation date.
* **ingredients** — holds a list of available ingredients by name.
* **meal\_ingredients** — a linking table connecting meals and their ingredients via foreign keys.
* **fridge\_ingredients** — tracks which ingredients a user currently has in their virtual fridge.
* **calendar\_meals** — schedules meals for users on specific calendar dates.

**Relationships Between Tables:**

* **meals** and **meal\_ingredients** have a **one-to-many** relationship via meal\_id. Each meal can have multiple ingredients.
* **ingredients** and **meal\_ingredients** have a **one-to-many** relationship via ingredient\_id. An ingredient can belong to multiple meals.
* **users** and **fridge\_ingredients** have a **one-to-many** relationship via the username field. A user can have multiple ingredients in their fridge.
* **meals** and **calendar\_meals** are related via meal\_id. A meal can appear multiple times in the calendar.
* **users** and **calendar\_meals** are related via the username field, allowing each user to manage their own meal calendar independently.

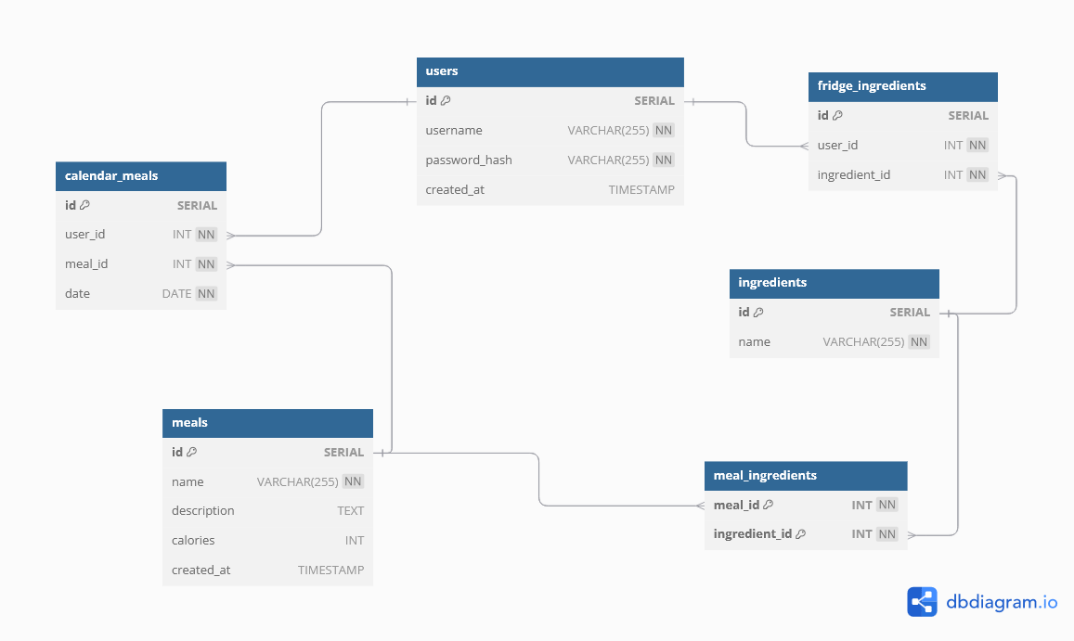


Figure 2- Databse ERD

**5.1 Results**

After completing **Sprint 5**, the project reached a fully operational state.  
The final application enables users to **register and log in**, manage their **personal fridge ingredients**, **browse a list of meals**, and **plan meals in a calendar interface** for specific dates.

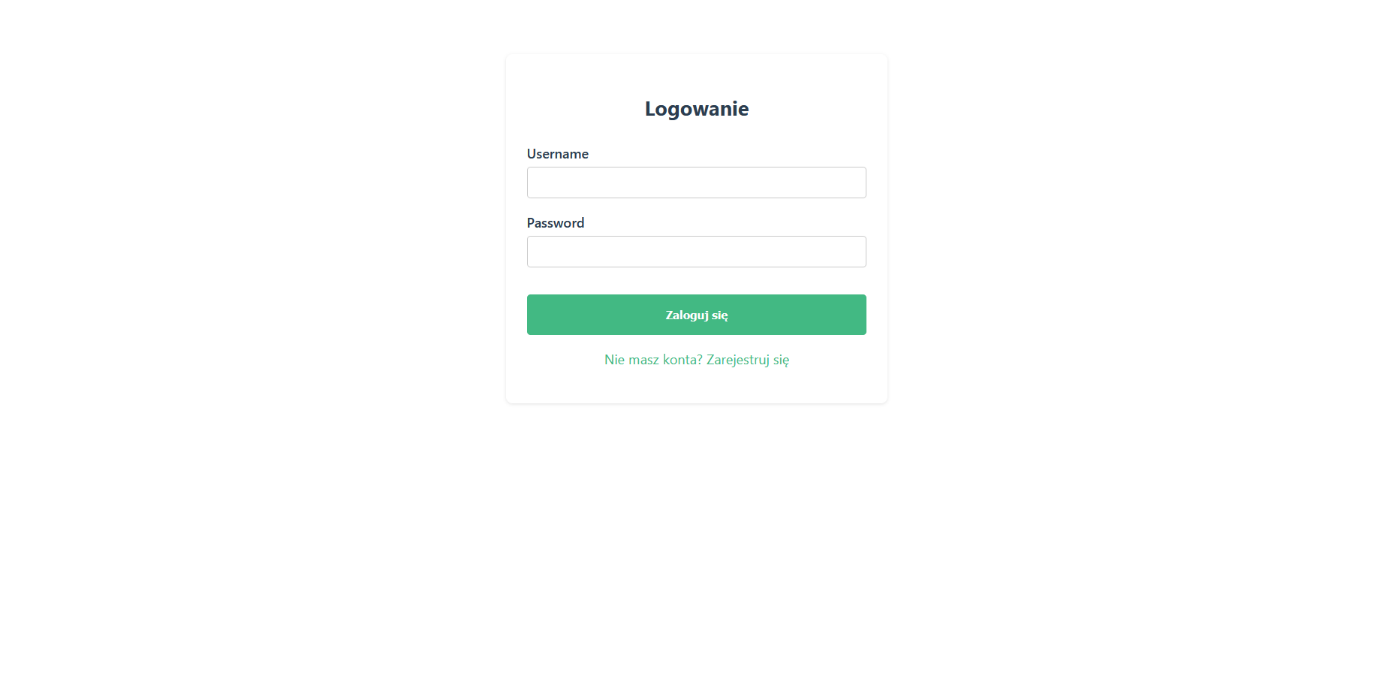


Figure 3- Login screen where existing users can enter their credentials to access their personal meal planner account or choose to register.

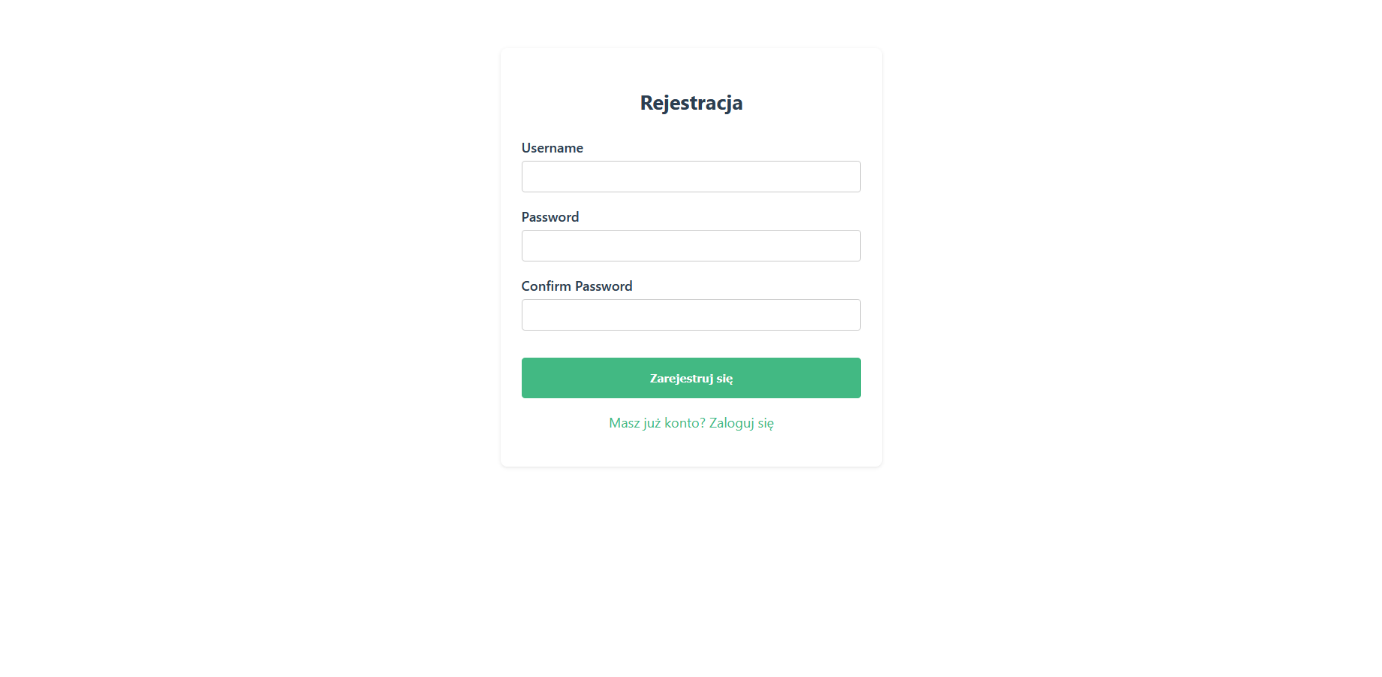


Figure 4- Registration form allowing new users to create an account by providing a username and password.



Figure 5- View presenting a list of available meals stored in the application database, displaying their names, descriptions and needed ingredients.

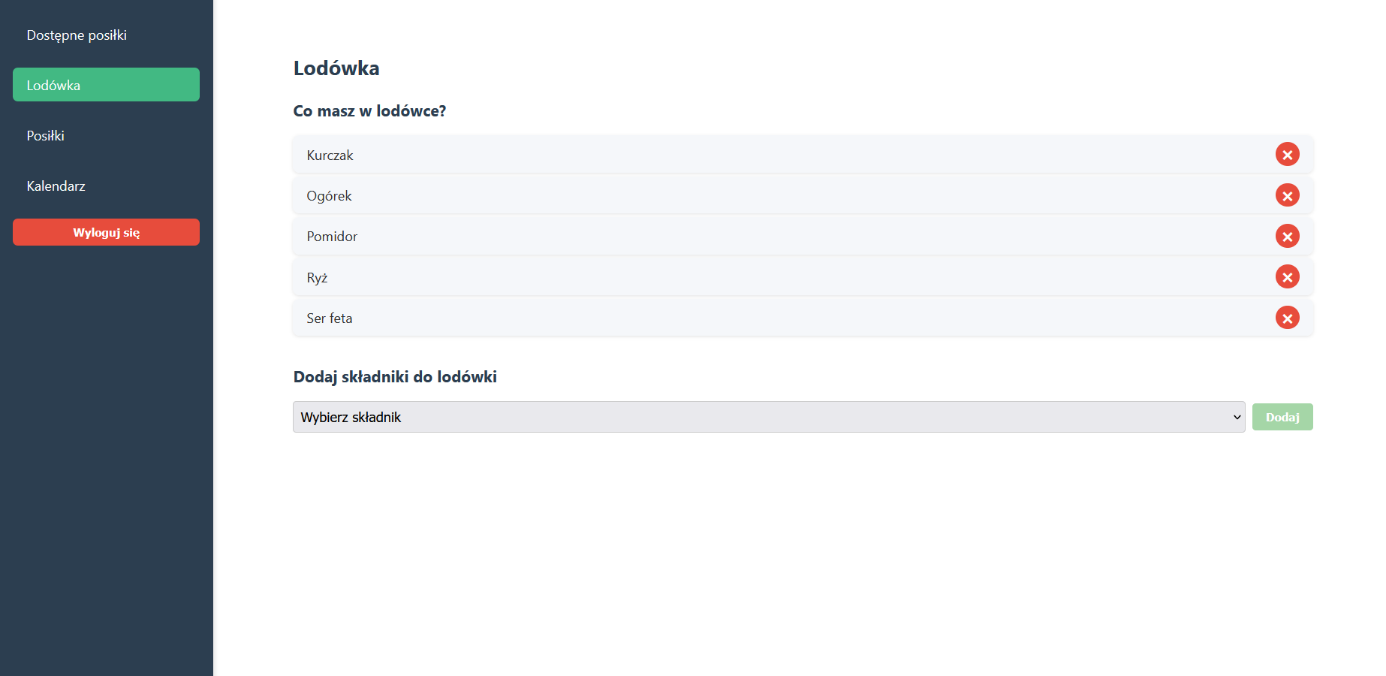


Figure 6- Fridge view showing the user’s personal list of available ingredients, with options to add or remove items.

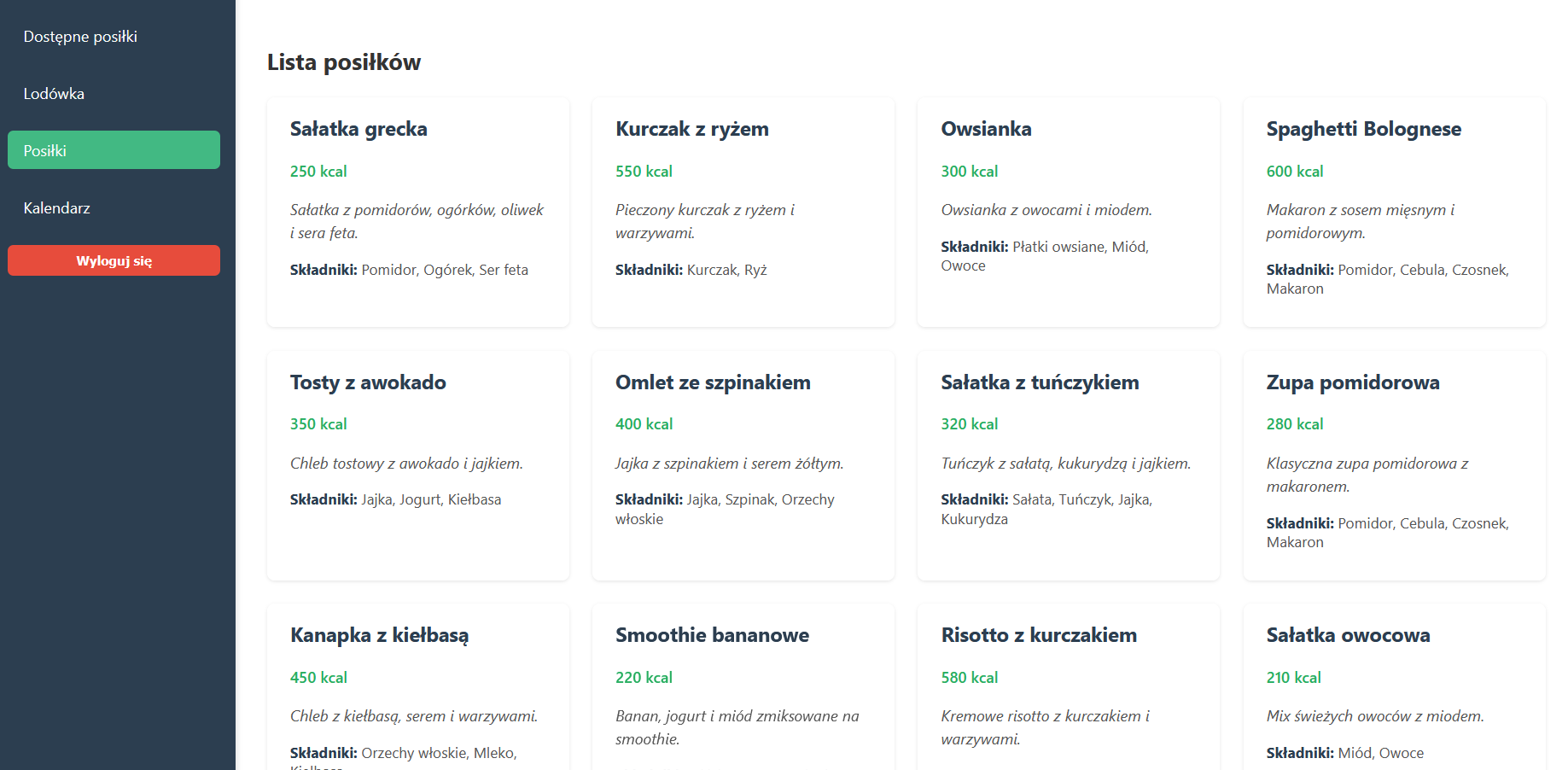


Figure 7- List of meals existing in a database

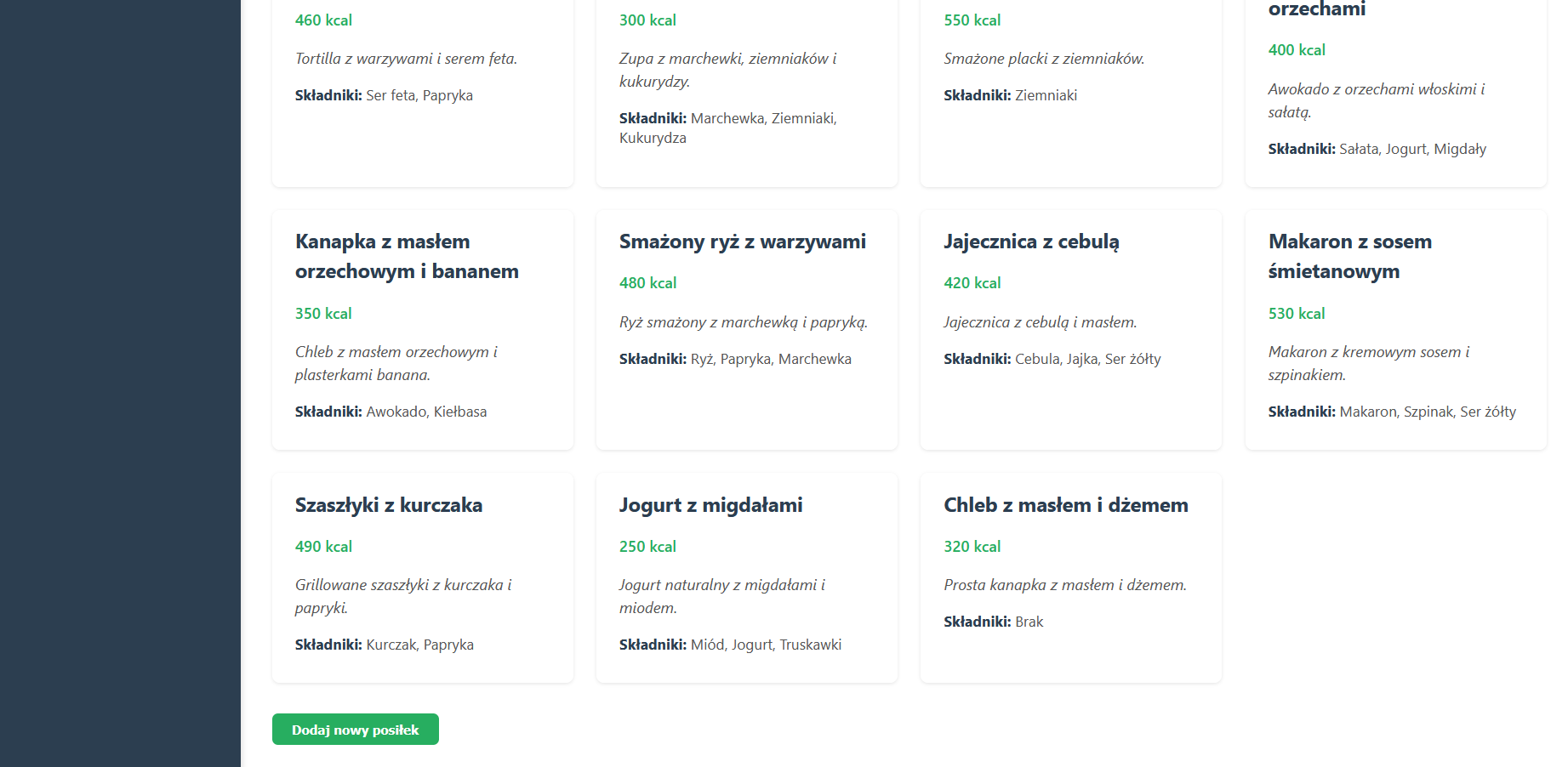


Figure 8- with option to add new meal.

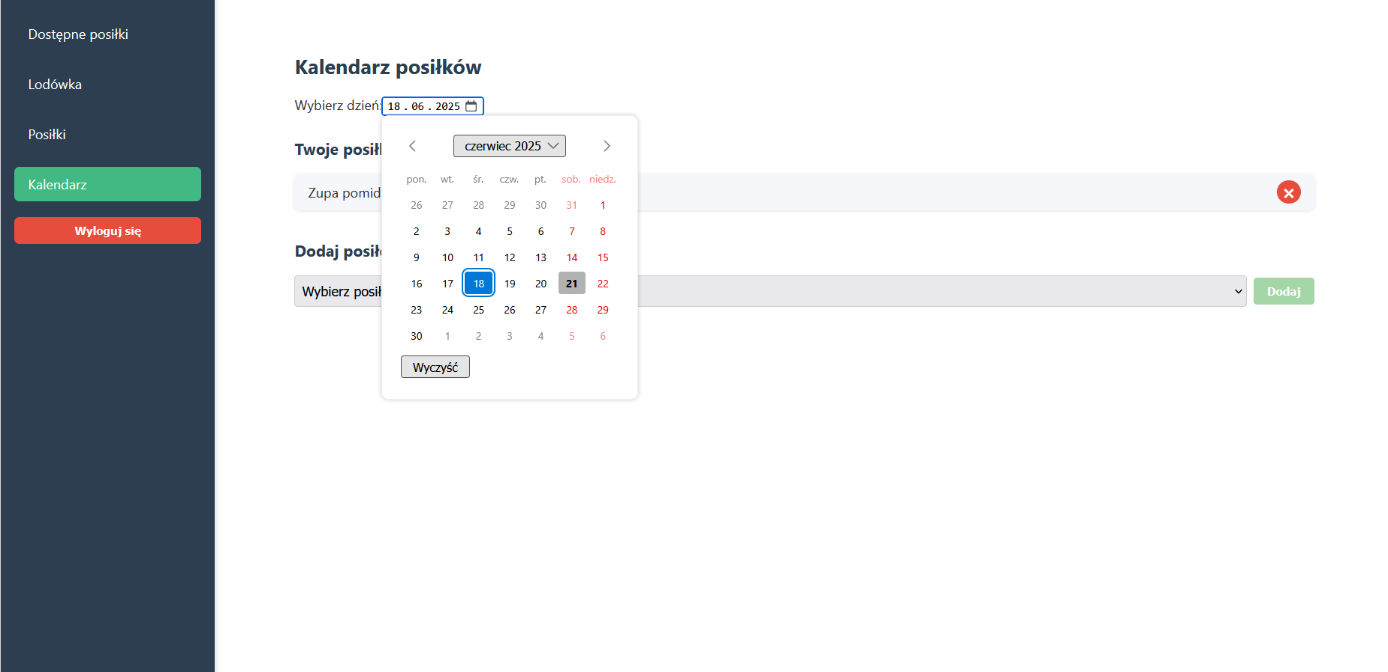


Figure 9- Calendar view with a date picker, allowing users to select a specific day and view planned meals for that date.

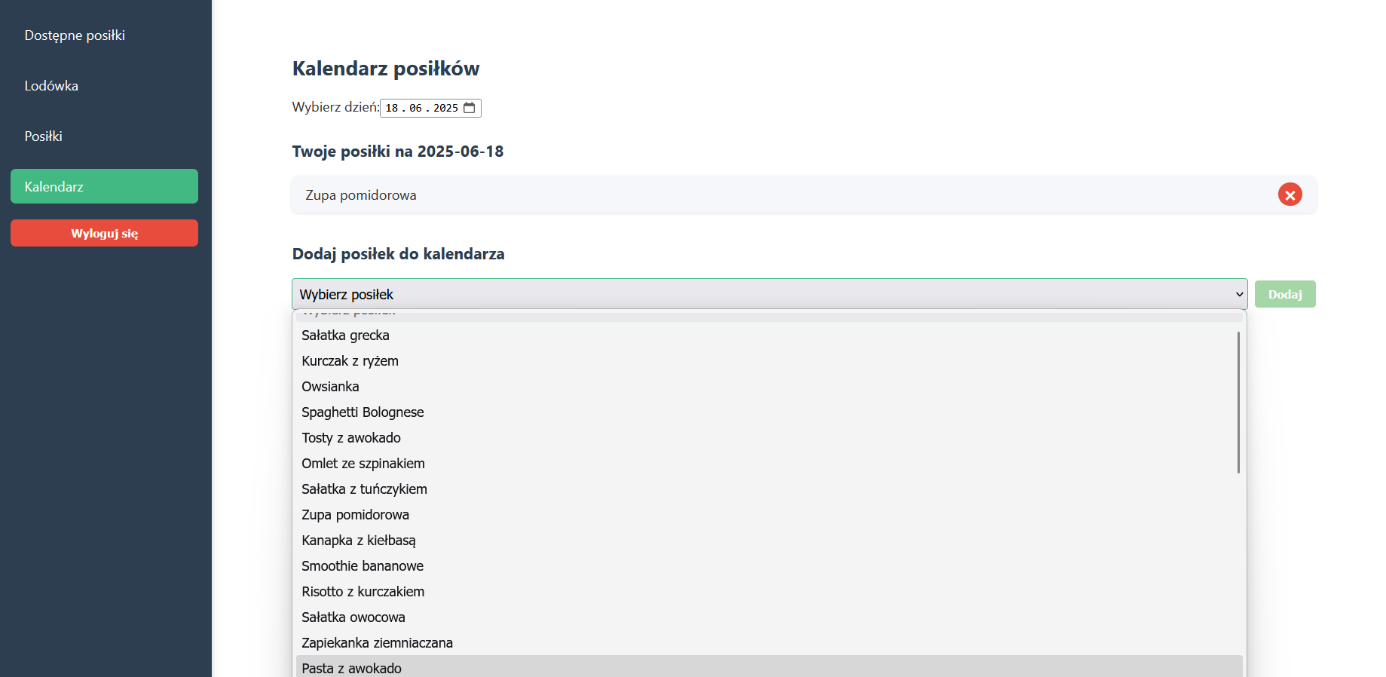


Figure 10- Calendar view with a dropdown menu for choosing a meal from the available list and adding it to the selected date.

The PostgreSQL database schema was fully implemented, including six related tables:  
users, meals, ingredients, meal\_ingredients, fridge\_ingredients, and calendar\_meals.  
Data relations were verified and maintained through proper foreign key constraints.

Additionally, all frontend views were styled for a clean and intuitive UI using **CSS variables and scoped styling in Vue**.

**5.2 Code Example**

Here is an example of code created during **Sprint 5**.  
The Vue component Calendar.vue allows users to interact with the backend API — fetching calendar meals, adding new ones, and removing selected entries by their unique id (refactored from mealId + date combination for more reliable operations).

Listing 1: Vue.js method for removing a meal by entry id.

async function removeFromCalendar(entryId) {

const res = await authFetch(`${import.meta.env.VITE\_API\_URL}/api/calendar/entry/${entryId}`, {

method: 'DELETE',

});

if (res.ok) {

await fetchCalendarMeals();

} else {

alert('Błąd podczas usuwania posiłku');

}

}

On the backend side, Express.js routes were extended to handle the new deletion logic based on entry id, ensuring accurate removal of specific records from the calendar\_meals table.

Listing 2: Express.js route for deleting a meal calendar entry.

router.delete('/delete/:id', authenticateToken, async (req, res) => {

const { id } = req.params;

try {

await pool.query(`DELETE FROM calendar\_meals WHERE id = $1`, [id]);

res.json({ message: 'Meal removed from calendar' });

} catch (err) {

console.error(err);

res.status(500).json({ message: 'Error removing meal' });

}

});

The system was also tested for database integrity, ensuring cascading deletions for connected data such as when removing a meal or ingredient.

This completes the planned scope for Sprint 5 and delivers a functional, dockerized meal planning platform with a calendar interface.

**6 Deployment and Maintenance**

The application will be deployed after each Sprint, without a fixed final release date.  
Our deployment and maintenance strategy includes:

* **Continuous Deployment** after every completed Sprint via Docker containers for both backend and frontend services.
* **Weekly backups** of the PostgreSQL database to avoid data loss.
* **Daily cleanup and optimization scripts** to maintain database integrity and remove unused or orphaned records.
* **Integrating user feedback** — a designated person from the team will regularly check user comments, answer questions, and log reported issues in Jira.
* **Social media and content marketing plans** including collaboration with food bloggers, local chefs, and influencers to promote the app.
* **Adding seasonal recipe packs and feature updates** periodically to keep the application fresh and engaging.
* **Automated deployment pipeline** set up using Docker Compose, allowing for easy scaling and environment resets when needed.

**7 Results**

**Observations**

I initially prepared too many tasks for Sprint 1.  
The **Sprint 1 Review** helped me adjust the workload more realistically for the following sprints.  
Now I have a clear sense of how many tasks can be handled within each micro-sprint by a three-person team.

**Achievements**

* We have learned the basics of **Docker, Vue.js, and PostgreSQL** integration.
* Successfully implemented **authentication and authorization** with JWT.
* Developed a **backend API** with endpoints for managing users, meals, fridge ingredients, and calendar scheduling.
* Created a functional **frontend in Vue 3** with views for login, registration, meal management, ingredient management, and calendar planning.
* Learned how to use **Scrum methodology** to manage the workflow effectively.
* Completed integration testing and validated data operations between frontend, backend, and database.
* Set up a **Docker-based deployment environment** for both local development and staging servers.

**Development Potential**

* The next step will be to enhance the frontend with **real-time notifications and meal recommendations**.
* Integrating **nutrition facts and caloric data analysis** for each planned meal.
* Adding a **social feature for sharing meal plans with friends or family members**.
* Implementing **user achievements and badges** for cooking streaks or healthy eating milestones.
* Potential for a **mobile app version** using technologies like Capacitor or React Native.
* Expanding the system to support **multi-language interfaces and regional ingredient databases**.