November 10, 2025

FoodPal Backlog

We are building a novel, distributed *cooking organizer* app that runs in a client/server setting.

1 Stakeholders

User: A user of the *client* application that uses FoodPal to organize recipes and prepare for cooking.
Admin: An operator of a FoodPal *server* application that allows others to host their recipes on a central system.

Note The application does not provide any specific functionality for admins. We assume that servers can be operated by individuals on their PCs/laptops. We might distinguish *Users* and *Admins* in some user stories for clarity.

2 Terminology

Recipe: A recipe for cooking. Has a name, a list of ingredients, and instructions for preparation.
Ingredient: An atomic item to be used in the recipe, e.g., salt or potatoes.
Server: An instance of the FoodPal backend. It is running on a network device that is reachable from other devices.
Client: The FoodPal frontend that is started by a *user*. Connects to a *server* to maintain all things around *recipes*.

3 Non-Functional Requirements

- The application is built with Spring/JavaFX.
- Client/Server communication is performed via HTTP and JSON through REST or websockets.
- Client/Server have shared data structures and use Jackson for automated object-mapping to/from JSON.
- It must be possible to connect more than one client simultaneously to the server.
- The application does not require the implementation of a user account system.
- The server is client-agnostic and does not store any *client-specific* information.
- The client uses a local file to persist its configuration and client state, e.g., server URL, favorite recipes, ...
- The location of the configuration file can be defined per (optional) command-line argument `-cfg path`.

> Info
> Students seem to struggle with the idea of a client-agnostic server. We want to see a design that centralizes shared information and moves all user-specific state to a local configuration file. This means that two FoodPal instances state cannot have any shared state, but it allows full focus on feature implementation.

> Info
> Do not write/parse the structure of the local config file yourself.
>
> - Create a `Config` class and use the Jackson mapper to convert the object to/from a JSON string.
> - You can use the `FileUtils` of the →Apache Commons IO library to read/write string to a `File`.

4 Epics and User Stories

4.1 Basic Requirements

Manage recipes in a distributed application.

Note: All(!) basic requirements must be met to pass the course.

As *user*, I want ...

- To store my recipes on a server, so I can share recipes with other users.
- To name recipes, so I can find them again.
- To add/change/delete ingredients for a recipe, so I remember what to use.
  At this level of the implementation, it is okay to store ingredients as basic strings.
- To add/change/delete individual preparation steps to a recipe, so I remember how to make it.
- To browse all existing recipes on the server, so I can get inspiration when cooking.
- To add/change/delete any recipe I see, so I freely manage the complete list of recipes on the server.
- To open a recipe and see its details, so I can understand its ingredients and preparation.
- To directly send all changes of recipe details to the server, so other users can see the updates.
- To clone an existing recipe and give it my own name, so I can give it my own twist without changing the original.
- To have refresh button (if there is no automated update), so I can pull updates from the server.
- To download a printable version of a recipe, so I can take it to the kitchen.
  You can pick a format of your choice, e.g., plain text or markdown is sufficient.

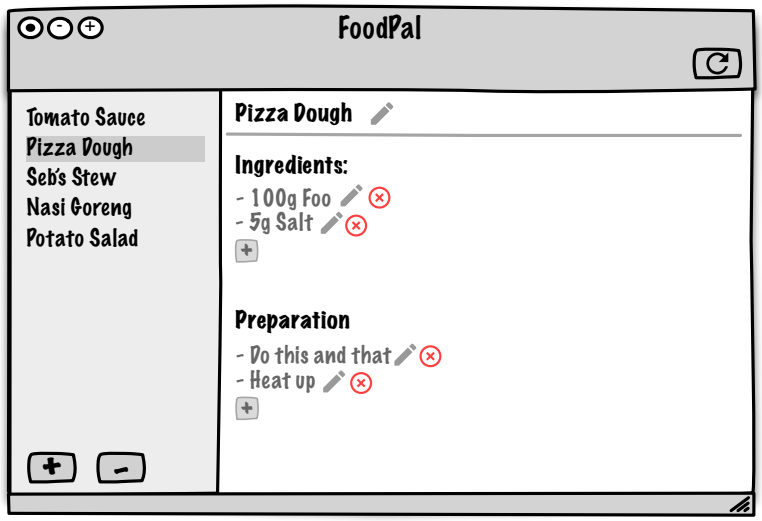

Figure 1: Basic Overview

4.2 Automated Change Synchronization

Modernize the application with auto-updates and make manual refreshes a thing of the past.

As *user*, I want ...

- To see that *some changes* (see later) are auto-propagated across all clients, so I do not have to manually refresh.
- To see a change of a title to be propagated, so no manual refresh is required.
- To see the addition/deletion of recipes to be propagated, so no manual refresh is required.
- To see any change in recipe content to be propagated when I view the same recipe, so I avoid manual refreshes.

Non-Functional Requirements

- The client uses *web sockets* to subscribe for changes.
- The client *does not poll* for changes, changes are pushed from the server.

> Info
> Do not send around *everything* with each change. Instead, model your web sockets in a way that, for example, allows subscribing for changes to the list of titles or for changes in one specific recipe.

Additional Information

- Synchronizing and handling conflicts in simultaneous *edits* is *really* difficult.
- Your application should not crash when multiple users are simultaneously editing, but making this scenario collision-safe and performant is out of scope for the CSE project. Our strong recommendation: do not even try to do more than a basic implementation, we also will not consider any optimization in the assessment.

4.3 Nutritional Value

Support dietary choices with information about the nutritional value of recipes.

As *user*, I want ...

- To define nutritional values for *ingredients*, so I can better plan my meals.
- To separate *ingredient* and *recipe* definitions, so I do not have to configure redundant information all the time.
- To add/change/delete *ingredients*, so I can make it easier to create recipes.
- To see an overview over all ingredients, ordered by name, so it is easy to find one.
- To set *fat*, *protein*, and *carbs* (per 100g) for ingredients, so I can define their nutritional value.
- To see an estimate for the kcal/100g for each ingredient, so I can cross check my values.
- To see in how many recipes an ingredient has been used, so I know which ones better not to delete.
- To change ingredient names and see the change propagated to recipes, so I can fix typos.
- To select ingredients for my recipe from a drop-down list, so I can conveniently use whats there.
- To receive a warning, should I try to remove a used ingredient, so I do not accidentally delete a needed one.
- If I decide to delete a used ingredient, I want to to delete the ingredient from all referring recipes.
- To define *formal* ingredient amounts (e.g., 100g) in my *recipes*, so I can be precise when it matters.
- To use a variety of units, like grams, kilograms, liters, or →table spoon, so I can use the most sensible unit.
- To use *informal* ingredient amounts (e.g., a pinch) in my *recipes*, so some things are left to personal taste.
- To be able to create new ingredients right from the recipe window, so I do not loose focus when a required ingredient is missing.
- To define resulting total number of servings for the recipe, so I can indicate how much food will be created.
- To scale any recipe by an arbitrary factor, so I can accommodate recipes to my situation.
  Scaling is "client-only" and should not change the recipe on the server. Informal amounts do not need to be scaled.
- To see normalized units in scaled recipes, so I do not see 1000g, but 1kg in the recipe.
- To see an estimation of the *kcal/100g* for a recipe, so I can make healthy dietary choices.
  Informal amounts should be ignored in the calculation.
- To see scaled nutritional values and serving sizes, so I can plan better.

Non-Functional Requirements

- The system automatically infers the serving size for recipes.
- The system automatically infers the kcal/100g.

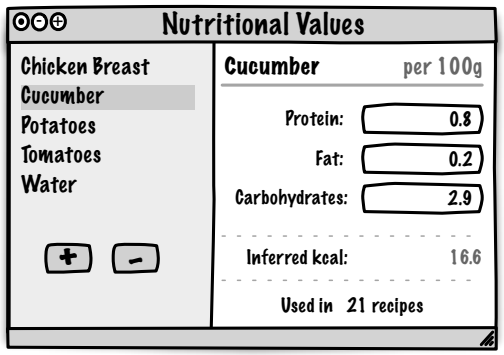

Figure 2: Nutritional Value Feature

4.4 Searching for Recipes

Make it possible to organize recipes and search for details to find specific recipes.

As *user*, I want ...

- To *star* my favorite recipes, so I can mark recipes that I really liked.
- To unstar favorites, so I can also reduce my list of favorites.
- To have a special overview over my favorite recipes, so I can browse them easily.
- To have my favorites highlighted even when looking at all recipes, so I have more visual guidance.
- To keep my favorites local, so nobody else knows about them.
  For example, you can store references to recipes in the local client config.
- Favorites to be references and not clones of recipes, so all changes to recipes get propagated.
- My favorites to keep working even when a recipe has been renamed on the server, so I do not loose track.
- To remove a favorite when I remove the corresponding recipe, so I keep my favorites consistent.
- To receive a warning when a favorite got deleted *by someone else*, so I can mourn the loss of the recipe.
- To use a full text search to find recipes, so I can search by ingredients, name, or special preparation instructions.
- To search for multiple things at once, so I can express more complex searches.
  It is enough to support a basic query syntax, in which the search string "t1 t2 t3" is considered as "t1" AND "t2" AND "t3".
- That a search filters the view of available recipes, so I am not bothered by a new search window.
- To search should filter both my favorites and the overall recipes, so it is easy to find the right recipe.
- To cancel a search by pressing escape, so I can quickly get back to a complete list of all recipes.

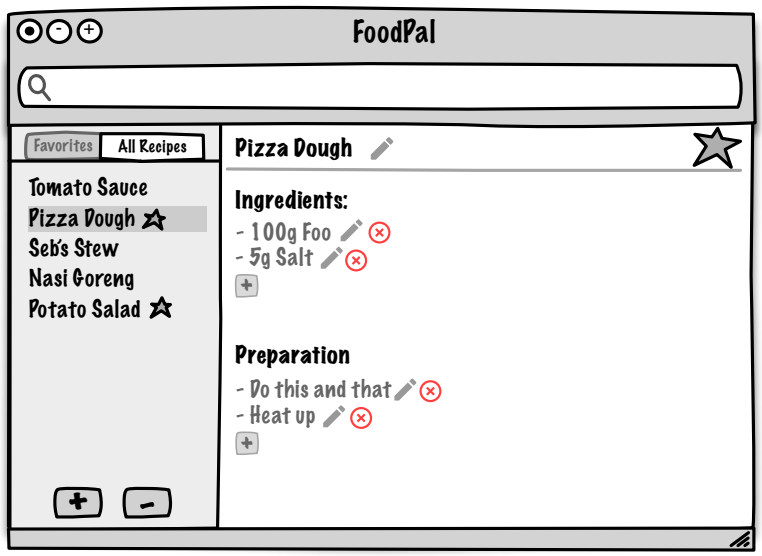

Figure 3: Search Feature

4.5 Shopping List

Plan your next meals and compile a shopping list to make it easy to get the right groceries.

As *user*, I want ...

- To compile a *shopping list*, so I can plan my next trip to the super market.
  Do not store the shopping list on the server.
- To add/remove ingredients directly on the shopping list, so I can add unrelated things, like a box of cereal.
- The shopping list to only contain a flat, ordered list of ingredients, so it is easy to go through while shopping.
- To add the ingredients of a recipe to my shopping list, so I can buy them on my next shopping trip.
- To see an editable *overview* over all ingredients before they are added, so I can adjust the list if needed.
- To adjust amount for all ingredients in the *overview*, so I can fine-tune recipes to my taste.
  For example, I want to use 250g of something instead of 200g
- To add/remove arbitrary ingredients in the *overview*, so I can fine-tune recipes to my taste.
- To confirm the *add overview* and add all its items to my *shopping list*, so I know what to buy.
- If I add the same ingredient through adding multiple recipes, I want to see it appear multiple times in the shopping list, with the name of the source recipes included
  Simple string propagation is sufficient, it is not necessary to reflect later changes to recipe names in the shopping list
- To reset the shopping list, so I can start over should I change my mind.
- To download a printable version of the shopping list, so I can take it to the super market.
  You can pick a format of your choice, e.g., plain text or markdown is sufficient.

> Info
> Make sure that scaled recipes (see "Nutritional Value" feature) are properly copied over to the shopping list.

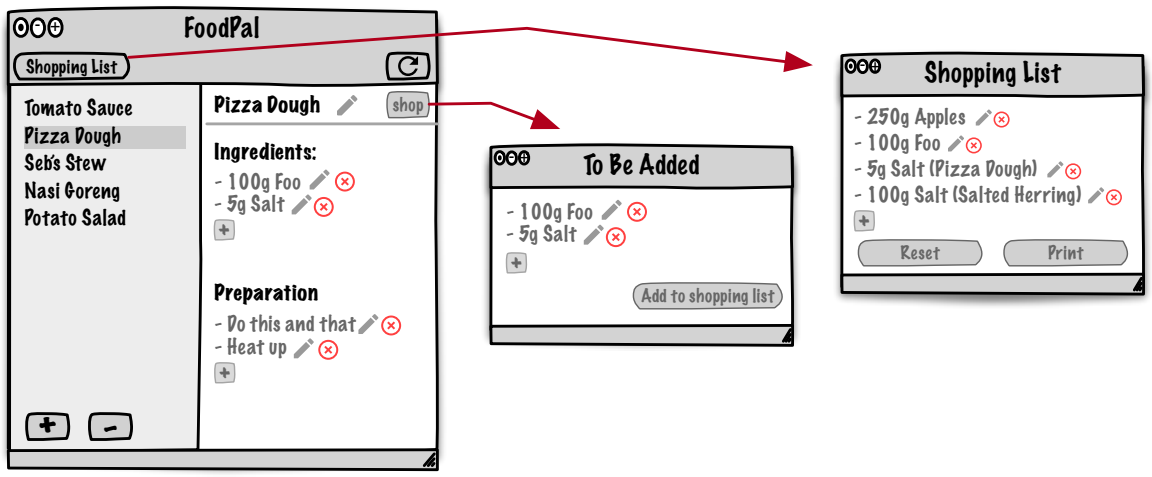

Figure 4: Shopping List

4.6 Live Language Switch

Add support for switching the language of your client during runtime.

As *user*, I want ...

- To see a language indicator, so I know which language is currently configured in my client on a first glance
- To see a flag icon as the language indicator, so I do not have to read additional text.
- To see all available languages through clicking the indicator, so I can find my preferred one easily.
- To persist my last language choice through a restart, so I do not have to pick the language again.
- To have all application labels and buttons translated to the selected language, so the app is international.
  Recipe contents, i.e., title, ingredients, preparation instructions, do not have to be translated.
- To set a specific language for a recipe, so I can indicate the language of ingredients and instructions.
- To have a language filter for recipes, so I only see recipes I understand.
- To select multiple languages at once, so I can find recipes in all languages I speak.
- To persist my language filter through a restart, so I do not have to configure it again.

Non-Functional Requirements

- App label translation should be implemented via →JavaFX Internationalization. This involves creating localized property files in the resources folder and calling →FXMLoader.setResources.
- The application must fully support at least English and Dutch. A third language must be added as a proof of concept, but can be made up.
- Please note that languages that do not flow left-to-right are much harder to integrate. While we encourage you to think about this use case, it is enough to limit your localization support to left-to-right languages.
- The language selection and filter is persisted in the config file.

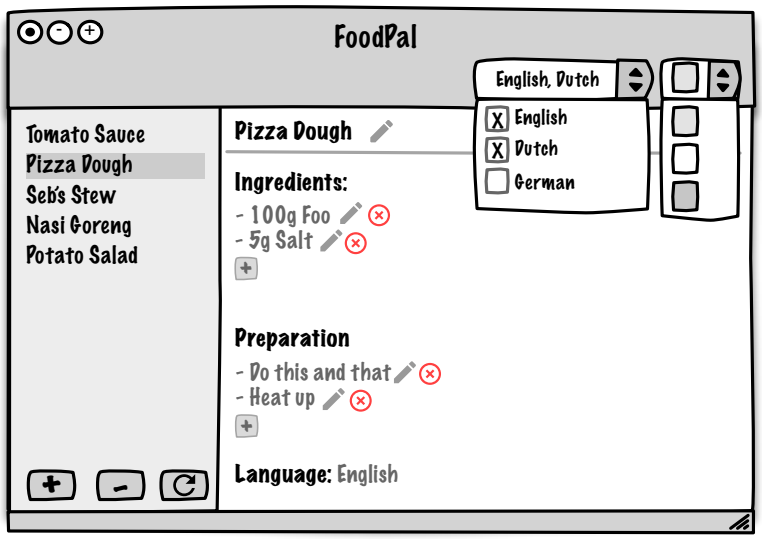

Figure 5: Language Switch