Mozzarella - webeC Project

Marius Küng (marius.kueng@students.fhnw.ch)

14.06.2015

Contents

1 Introduction

1.1 Motivation

Mozzarella is a collaborative shopping-/what's in your fridge list app. Mozzarella aims to help the user keep track of what groceries he has in his fridge, which of these he should consume and what he needs to buy soon. Because of shared lists it's able to keep track of an entire household.

The idea originates from the problem that a family has several frigdes, freezers and kitchen cabinets with loads of stuff in it. It's super easy to loose track of what's already at home and should be consumed next. Otherwise you throw away way to much groceries.

With this you can easily save money and help the environment by not throwing away food.

1.2 Features (v1)

In the first version the following features will be implemented:

- **Shared lists** with unlimited users. Invite everyone you want. For example a you can manage lists for ***at home***, ***weekend bbq***, or the ***birthday party***.
- **Item entries** in every lists. Each item has a title, a piece counter ***(how many apples do I have?)***, and an overview of all pieces ***(1 & 3 pound pack of ground meat)***
- **Due-date notifications** which inform you easily which items you should consume next.

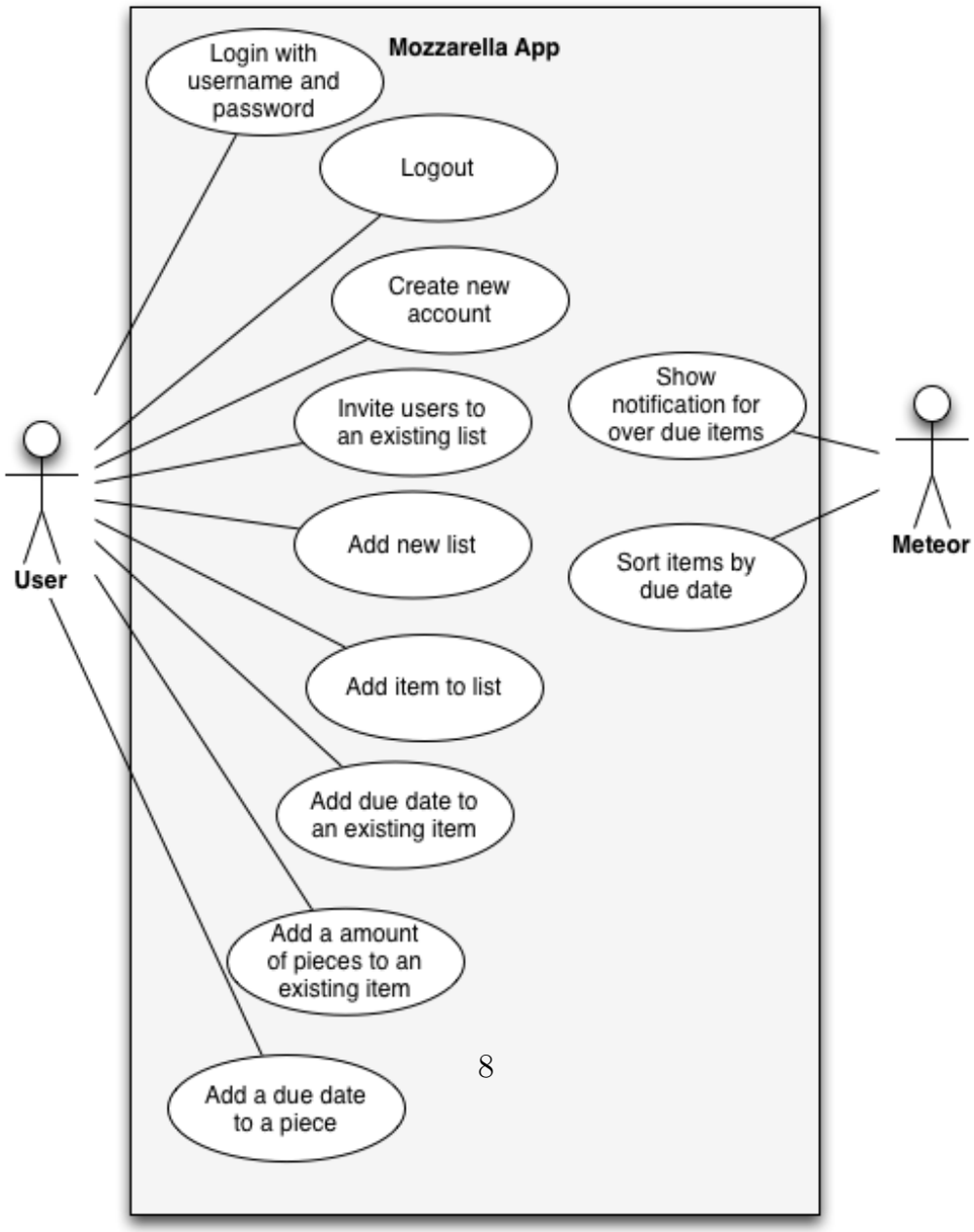
1.3 v2

Future improvements:

- **Item notifications** If a user makes a CRUD-action on items in a shared list all other users are getting notified about the recent changes.
- **UX improvements** There is still a lot of stuff to fix.

2 Requirements

2.1 Usecase Overview



2.1.1 Usecase: Login with username and password

Use Case	Login with username and password
Description	Allows user to login with his username + password
Actors	User
Preconditions	Not logged in
Basic Flow	User fills in username + password, clicks login
Alt. Flow	None
Postconditions	*User is logged in (has session token)*
Notes	-

2.1.2 Usecase: Logout

Use Case	Logout
Description	Logout user from running current session.
Actors	User
Preconditions	Logged in
Basic Flow	User clicks logout button.
Alt. Flow	-
Postconditions	*User is logged out (session token removed)*
Notes	-

2.1.3 Usecase: Create new account

Use Case	Create new account
Description	A user creates a new account.
Actors	User
Preconditions	User doesn't have an account yet.
Basic Flow	User clicks "create new account" button.
Alt. Flow	User tries to login but no existing account was found.

Use Case	Create new account
Postconditions	UC: Login
Notes	-

2.1.4 Usecase: Add new list

Use Case	Add new list
Description	A user creates a new list.
Actors	User
Preconditions	UC: Login
Basic Flow	User clicks “add new list” button.
Alt. Flow	-
Postconditions	List is now added and visible to the user
Notes	-

2.1.5 Usecase: Invite user to existing list

Use Case	Invite user to existing list
Description	User shares a list with other users.
Actors	User
Preconditions	The invited user exists (username).
Basic Flow	User clicks the “share list” button, enters username.
Alt. Flow	-
Postconditions	List is now shared.
Notes	-

2.1.6 Usecase: Add item to list

Use Case	Add item to list
Description	A user adds a new item to an existing list.
Actors	User
Preconditions	UC: Login, UC: Add new list
Basic Flow	User fills in the “new item” input field an submits it.
Alt. Flow	User adds a **due date** and a **piece amount** to the new item.
Postconditions	Item is now added to the existing list and visible to the user.
Notes	-

2.1.7 Usecase: Add due date to an existing item

Use Case	Add due date to an existing item
Description	A user adds a due date to an existing item.
Actors	User
Preconditions	UC: Login, UC: Add new item
Basic Flow	User clicks the “add due date (clock)” button.
Alt. Flow	User double clicks the current due date and changes it.
Postconditions	-
Notes	Due date can be null.

2.1.8 Usecase: Add amount of pieces to an existing item

Use Case	Add amount of pieces to an existing item
Description	A user adds an amount of pieces to an existing item.
Actors	User
Preconditions	UC: Login, UC: Add new item
Basic Flow	User clicks on the arrows on the amount element.
Alt. Flow	If no amount is given, 1 is the default amount.
Postconditions	-

Use Case	Add amount of pieces to an existing item
Notes	A piece is automatically added with a reference to its item.

2.1.9 Usecase: Add due date to a piece

Use Case	Add due date to a piece
Description	A user adds a due date to a piece of an item.
Actors	User
Preconditions	UC: Login, UC: Add new item
Basic Flow	User opens item view and changes due date.
Alt. Flow	-
Postconditions	-
Notes	-

2.1.10 Usecase: Show notifications for over due items

Use Case	Show notifications for over due items
Description	The app shows the user which items are over due.
Actors	Meteor (Backend)
Preconditions	UC: Add new item
Basic Flow	Meteor adds all over due items to a generated list.
Alt. Flow	Meteor shows the amount in over due list title.
Postconditions	-
Notes	-

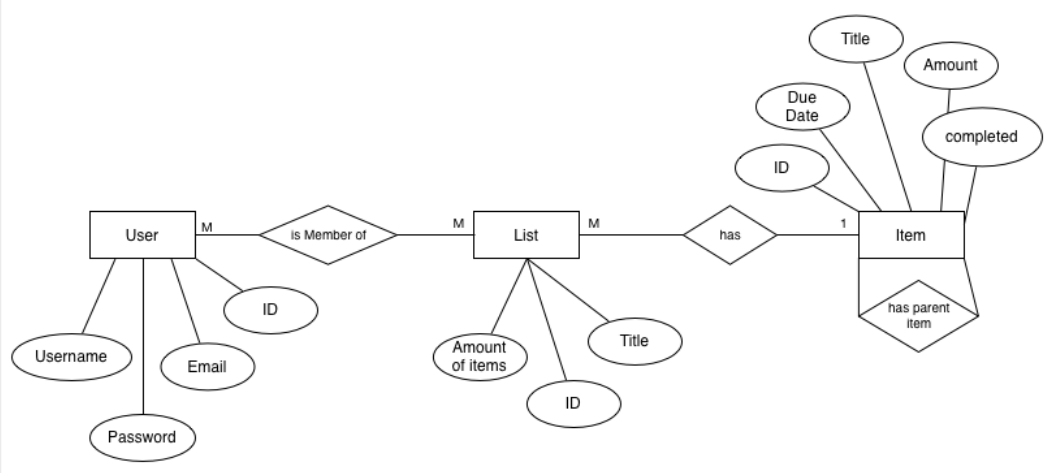
2.1.11 Usecase: Sort items by due date

Use Case	Sort items by due date
Description	The app sorts all items in each list by its due date.

Use Case	Sort items by due date
Actors	Meteor (Backend)
Preconditions	UC: Add new item
Basic Flow	Old items are on top, new items below.
Alt. Flow	-
Postconditions	-
Notes	-

3 Conceptual Data Model

3.1 Overview



3.1.1 Entity: User

A *user* of the app. He can be a member of one or more *lists*. He can also share a *list* with other *users*.

3.1.2 Entity: List

A *list* with a given name and one or more *users* that stores and sorts several *items*.

3.1.3 Entity: Item

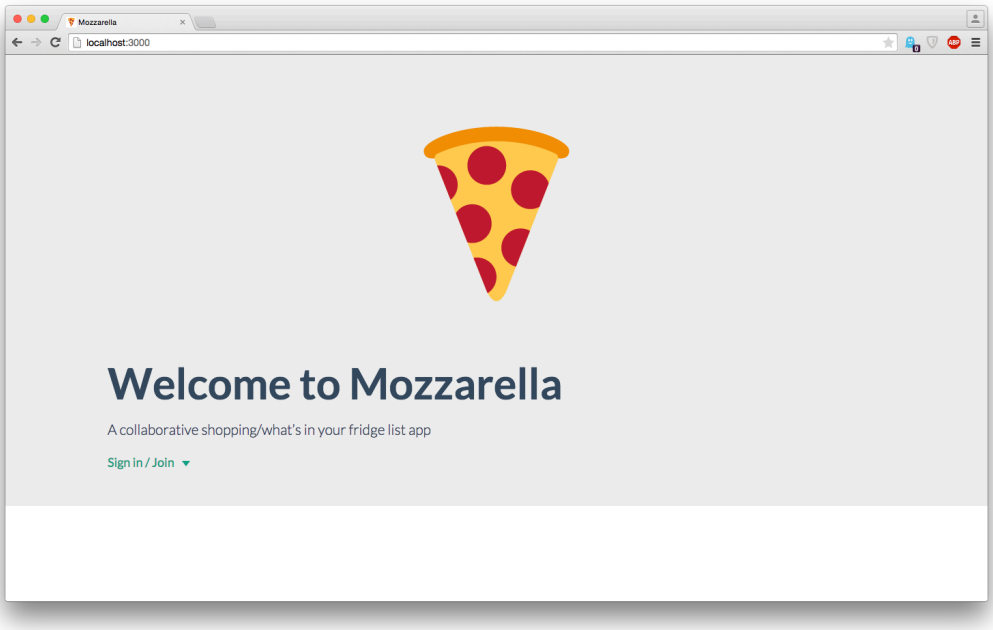
An *item* has the following attributes: *title*, *amount*, *due date* and *completed*. *Completed* items are automatically hidden in a *list* and visible on request.

3.2 Operations

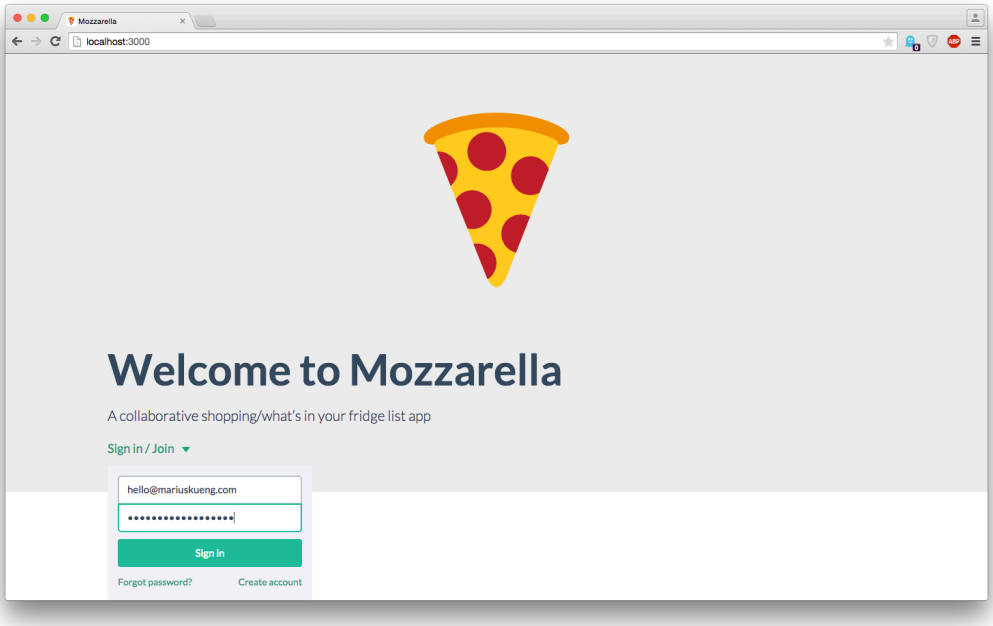
Actor	Operation	Arguments	Result
User	Add item to list	`item`	`success` or `fail`
User	Remove item from list	`item`	`success` or `fail`
User	Change item title	`item`	Changes old `title` to new `title`
User	Tick item on list	`item`	Changes `completed` to `true`
User	Add amount to list	`item`	`success` or `fail`
User	Add due date to list	`item`	`success` or `fail`
User	Attach item to item	`item`	`success` or `fail`
User	Add user to list	`user`	`success` or `fail`
User	Remove user from list	`user`	`success` or `fail`
User	Login / Logout	`un/pw`	`Session ID`

4 Mockups

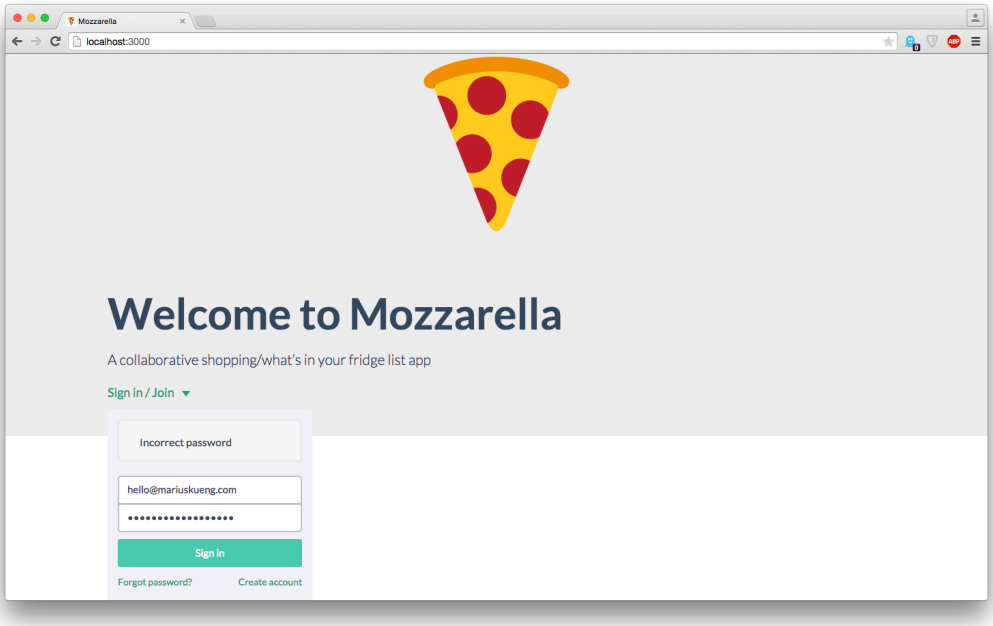
4.1 Landing page



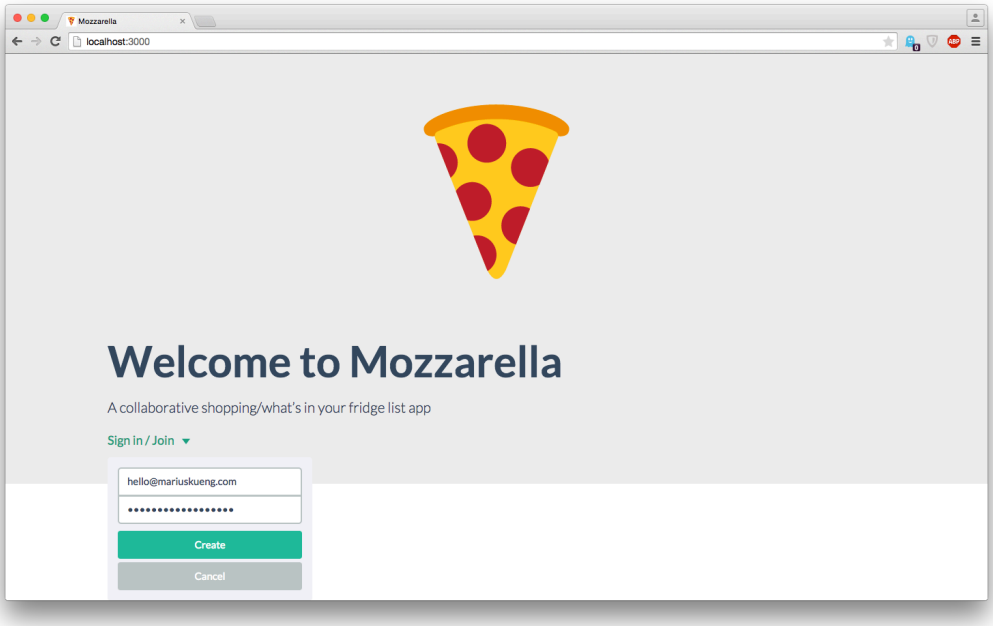
4.2 Login page



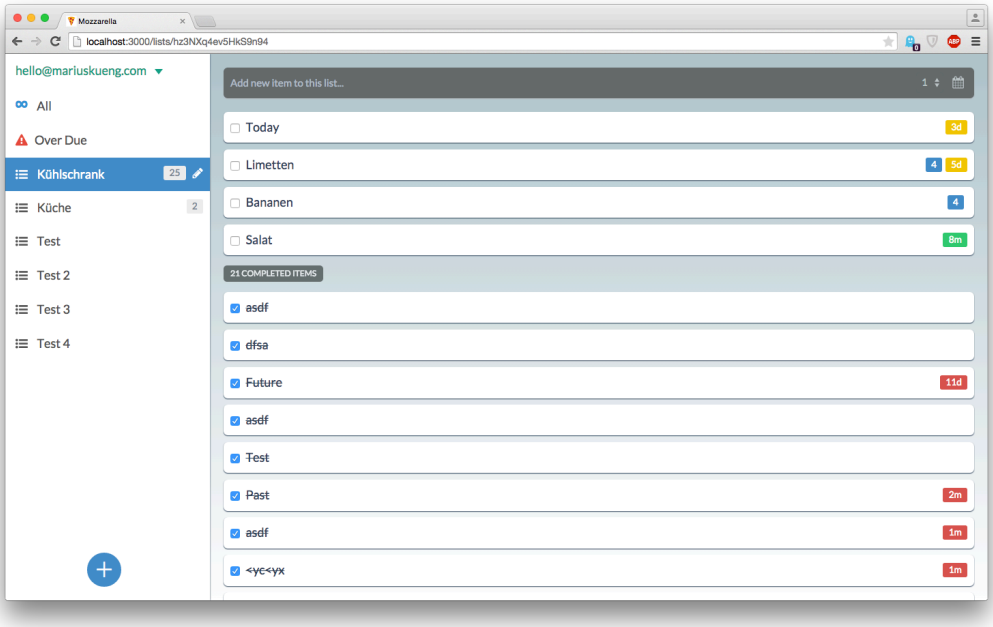
4.3 Login page



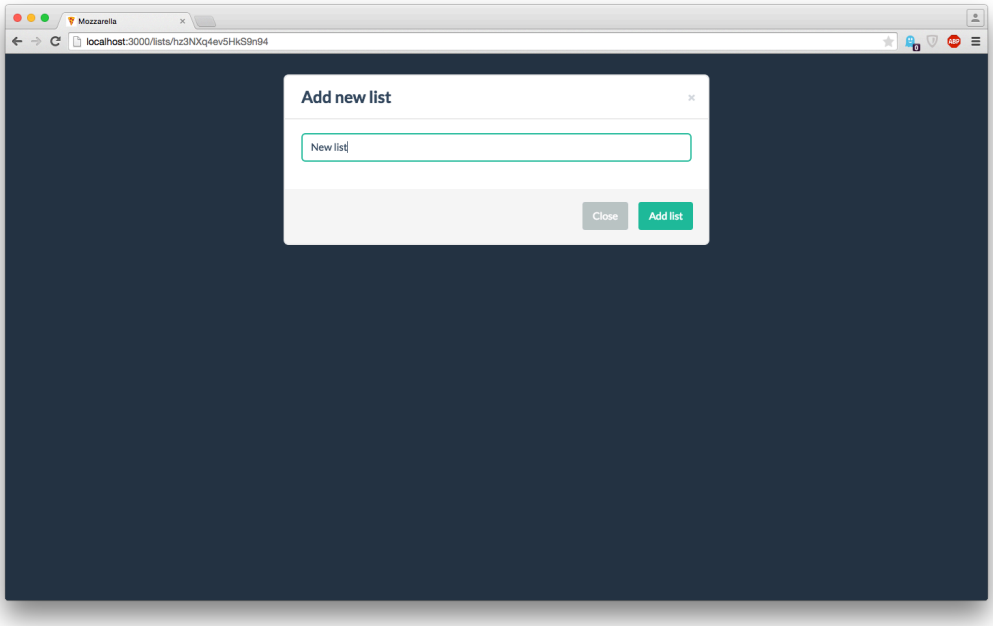
4.4 Signup page



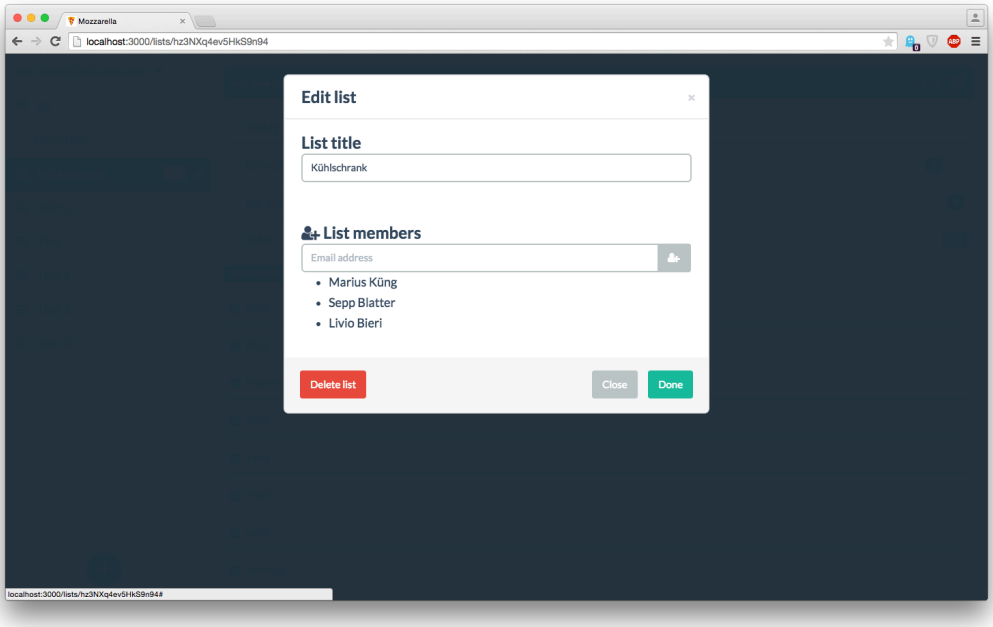
4.5 List view page



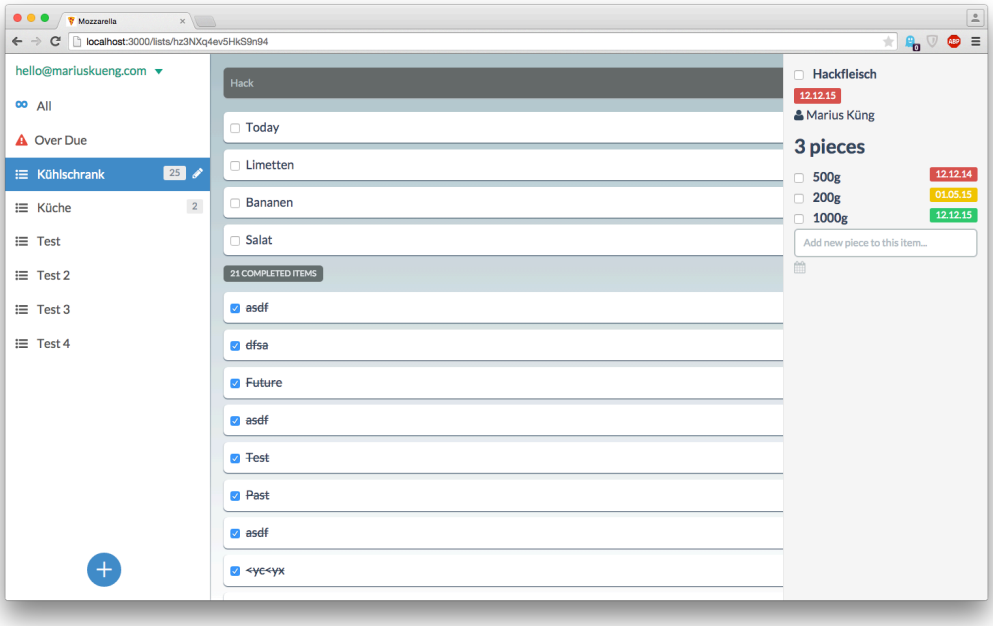
4.6 List create page



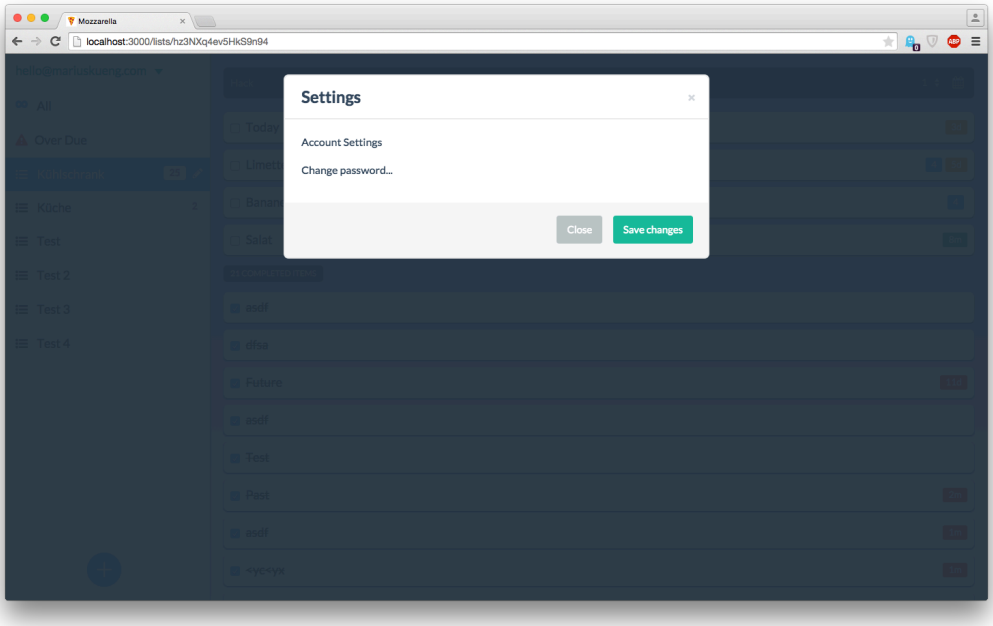
4.7 List edit page



4.8 Item edit page



4.9 Settings page



5 Implementation

5.1 Final vs Mockup

- Edit view was heavily improved
- Delete item button added
- Added typeahead.js
- Notification plugin added.

5.2 Technologies

Due to the Meteor Socket technology there is no classic API needed. Everything is getting piped through a socket and updated automatically. All data collections (Lists, Items, etc.) are reactive.

5.2.1 Backend

This project is built with Meteor. Meteor is a complete open source platform for building web and mobile apps in pure JavaScript.

5.2.2 Database

Meteor only offers MongoDB for now. A document-oriented database in JSON format. Relations can be implemented with keys and cross-table matching queries. Mongo is fast and very easy to implement. It is

recommended to use a schema setup to handle migrations and specific datatypes. This was not used in this project though.

5.2.3 Frontend

- Meteor: Meteor is a full-stack framework and works on the back- as well as on the frontend.
- Bootstrap: As a ui framework with responsive components.
- Flat-UI: A flat design library on top of Bootstrap
- Typeahead.js A autocompletion library for form inputs. Was used to add food.
- S-Alert
- Fontawesome
- Jasny
- Moment.js

5.3 Development Tools

- Meteor: Comes with a built-in build tool
- Jasmine: Testing framework
- TravisCI: Continous integration service
- GitHub: Distributed source control service with Git
- Heroku: App deployment service

5.4 Development Workflow

- Meteor builds app
- Push to heroku

- TravisCI starts build, testing
- Heroku starts deployment as soon as Travis is finished & successful.

5.5 Project structure

- client: All data shared on the client (templates, helpers, libs,styles)
- lib: Data & methods shared on both client and server. Most database actions (i.e. Insert) is handled through such a method
- public: Data such as images, icons
- server: Sensitive data only accessable by the server.
- tests: Application tests
- packages: Version Meteor packages for package manager

5.6 Installation

5.6.1 Install Meteor if you haven't already

```
$ curl https://install.meteor.com/ | sh
```

5.6.2 Get it

```
$ git clone git@github.com:mariuskueng/mozzarella.git
```

5.6.3 Run it

```
$ cd mozzarella
$ meteor
```

5.7 Next steps

- Replace Bootstrap with Angular Material. Boostrap is simply not designed to built a modern web-app.
- Refactor code
- Implement a schema
- Add a monitoring system for the app to observe events and db queries

5.8 Final thoughts

Building this project was very intersting and a lot of fun. I wanted to built something bigger than a tutorial app with Meteor since over a year ago. I think that Meteor has a future but a least in a year. It really makes developing and prototyping fast but sometimes also a bit messy. Furthermore it can be confusing because so much is happening at the same time.