Strawberry

the collaborative shopping list

Livio Bieri // June 2015

Contents

1. Introduction

1.1. What's the point?

Strawberry is a collaborative shopping list. Strawberry aims to simplify doing groceries in households with more than one person by offering a simple shared shopping lists.

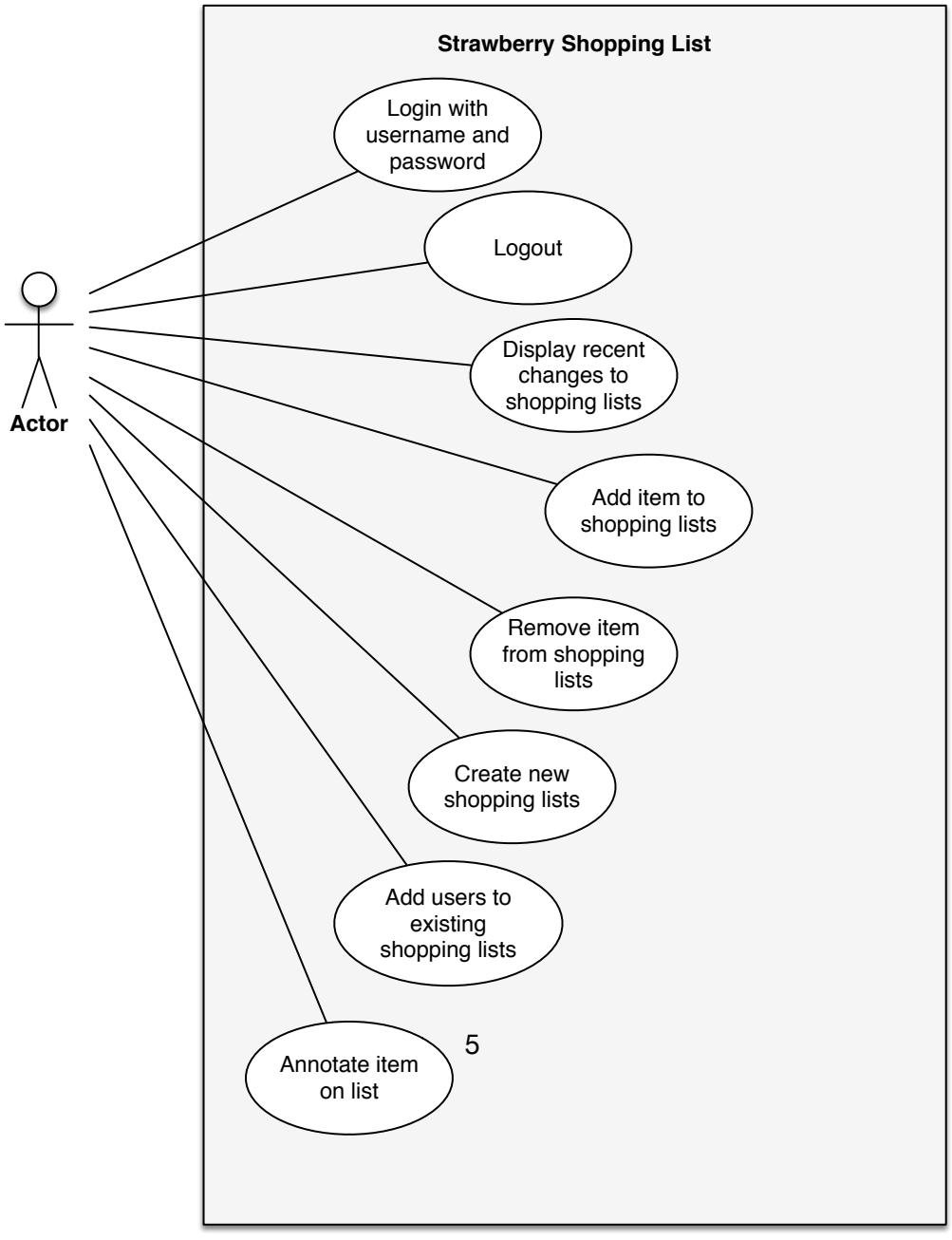
1.2. What are the features?

Strawberry has the following features:

- **Shared lists** with as many users as you wish. You can simply invite everyone with a Strawberry account to take part in your list. Each user can many have as many seperate lists with different people as he wants. For example: *"me and girlfriend", "me and family" or "me and my flatmates".* Of course it is also possible to have a private list just for yourself.
- **Simple item-based lists** simply add items and remove them by checking them off the list. You can add a short note to each item seperately. For example if you want to have a certain brand of chocolate.
- **Twitter Style Typeahead** with common shopping items. Such as "Apples", "Bananas" or "Strawberries" in item input field.
- **Sign in via Facebook.**

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 (see `/loggedin` response)
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 logged in session.
Actors	User
Preconditions	Logged in (see `/loggedin` response)
Basic Flow	User clicks logout button.
Alt. Flow	User enters URL `/logout`
Postconditions	*User is logged out (has session token revoked)*
Notes	-

2.1.3. Usecase: Display recent changes to shopping lists

Use Case	Display recent changes to shopping lists
Description	Shows a history of recent changes.
Actors	User
Preconditions	Logged in (see `/loggedin` response)
Basic Flow	User pressed the `bell-icon` in the top right.
Alt. Flow	None.
Postconditions	List is shown.
Notes	-

2.1.4. Usecase: Add user to existing shopping list

Use Case	Add user to existing shopping list
Description	User can invite others to existing shopping lists.
Actors	User
Preconditions	Logged in (see `/loggedin` response), list exists.
Basic Flow	User searches / enters usernames and presses add.
Alt. Flow	None.
Postconditions	User has been added to the list as collaborator.
Notes	-

2.1.5. Usecase: Remove item to shopping lists

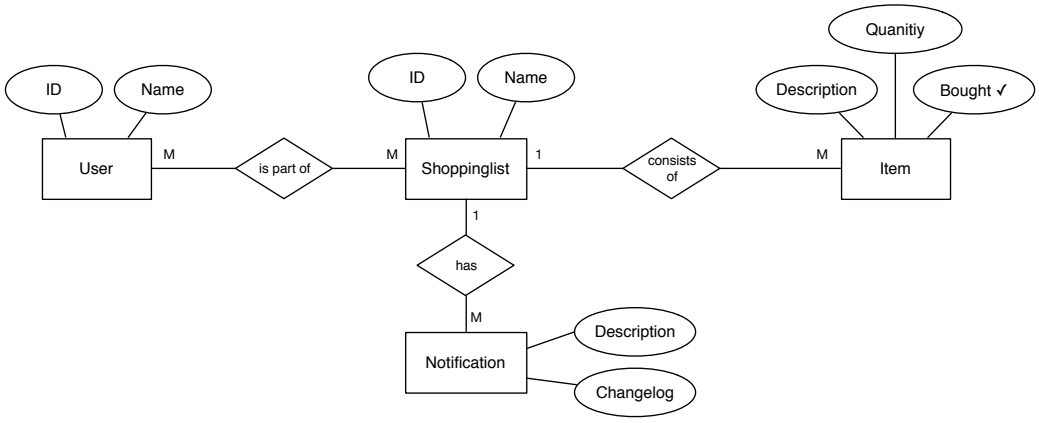
see *Usecase: Add item to shopping lists*. Same as this but instead of adding item will be removed.

2.1.6. Usecase: Create new shopping lists

Use Case	Create new shopping lists
Description	Creates a new empty shopping list.
Actors	User
Preconditions	Logged in (see `/loggedin` response)
Basic Flow	User enters the name of the new list and clicks add.
Alt. Flow	None.
Postconditions	A new empty shopping list has been created.
Notes	-

3. Conceptual Data Model

3.1. Overview



3.1.1. Entity: User

A user of the system *(as in member of (many | one) shopping lists)*.

3.1.2. Entity: Shoppinglist

Set of items for one or several users. The list has **Notifications** about the latest changes so all members can see the latests changes to the list.

3.1.3. Entity: Item

An item is defined as following: A `description` such as *Strawberry*, a `quantity` for example *some*, *one* or *many* and a `bought` checkmark which indicates an item that has been ticked of the the list *(thus no longer is required to be bought).*

3.2. Operations

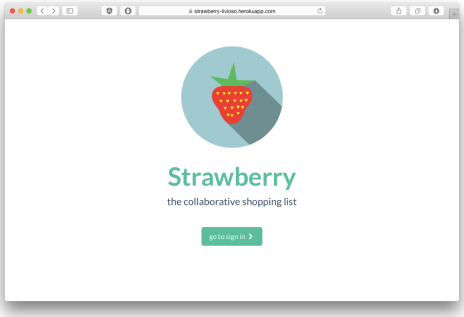
Actor	Operation	Arguments	Result
User	Add item to list	`item`	`success` or `fail`
User	Tick item on list	`item`	Changes `bought` to `true`
User	Add user to list	`user`	`success` or `fail`
User	Remove user from list	`user`	`success` or `fail`
User	Login / Logout	`un/pw`	`Session ID`
Server	Notify change	`changes`	-

Notes: `fail` could be returned when you try to access a list via id you are not a member of.

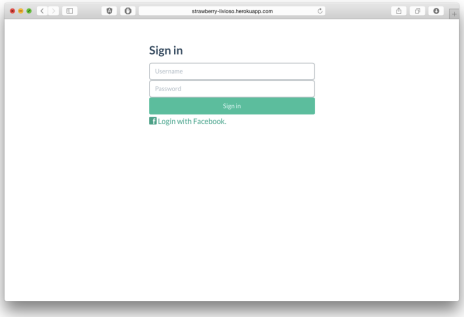
4. Mockups

4.1. Landing Page / Sign In Page / Main Page

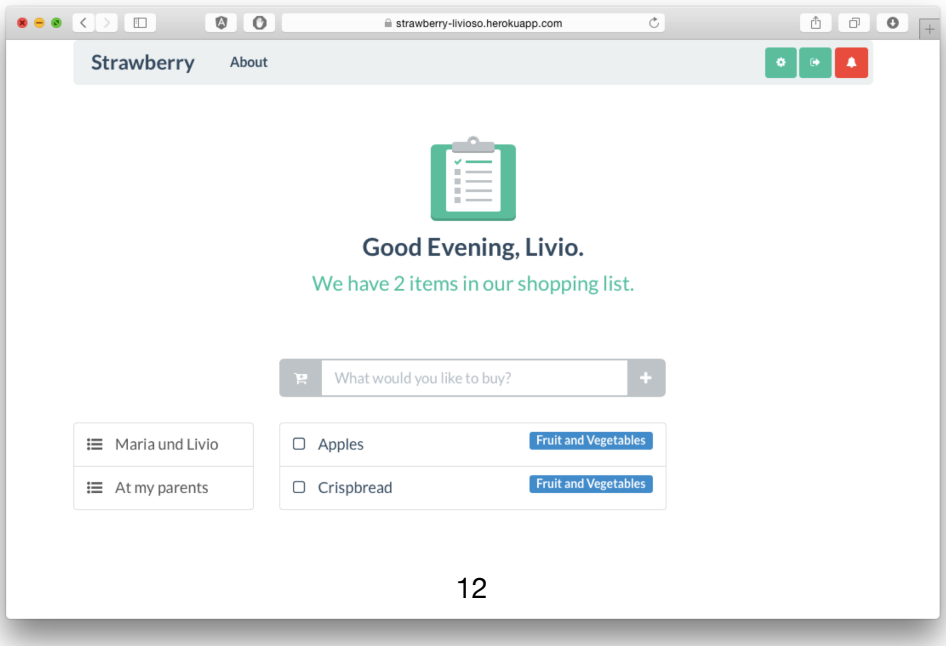
Landing Page



Sign In Page

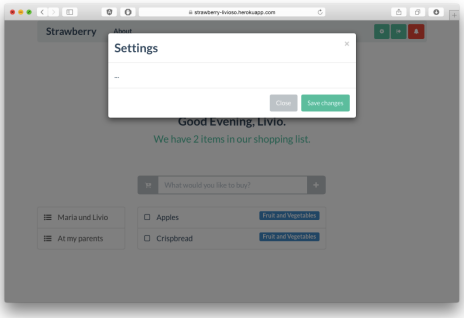


Main Page

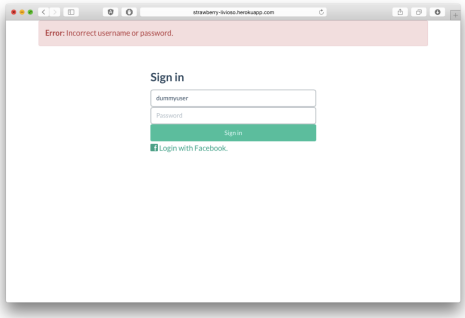


4.2. Settings Page / Sign In Page Error / Main Page Typeahead

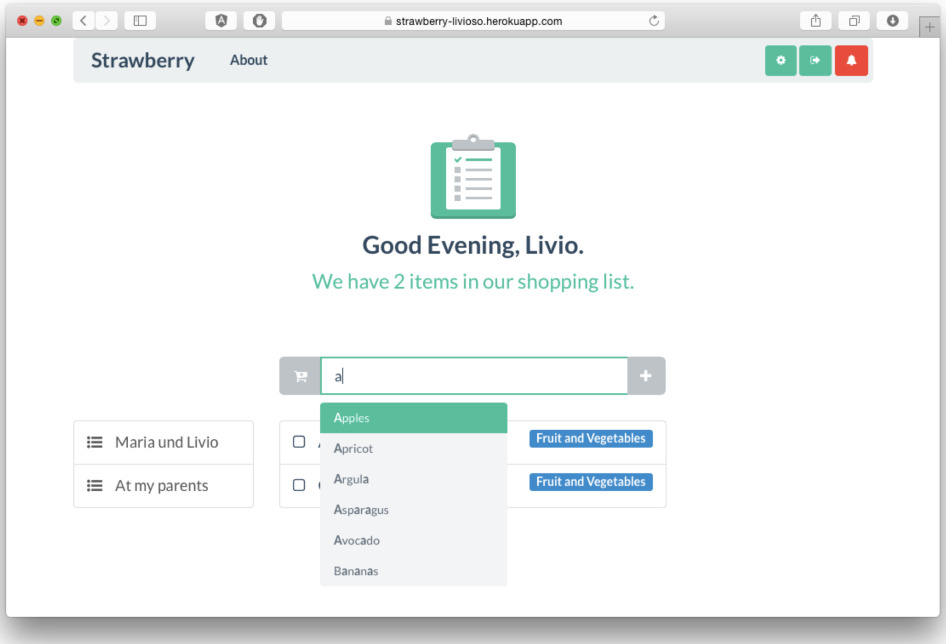
Settings Page



Sign In Page - Error



Main Page - Type Ahead

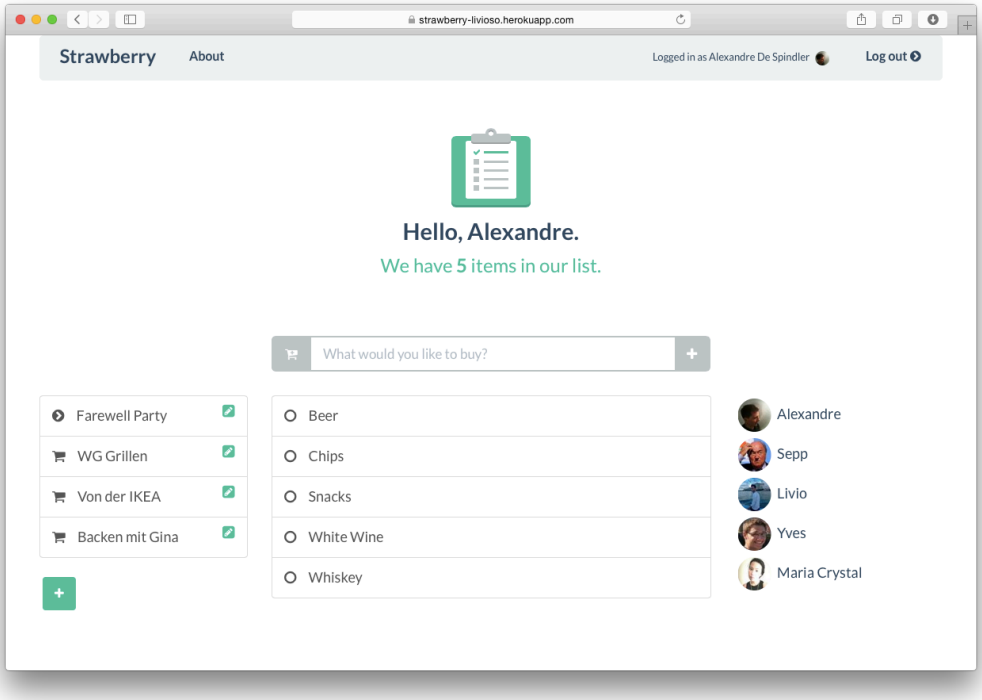


5. Implementation

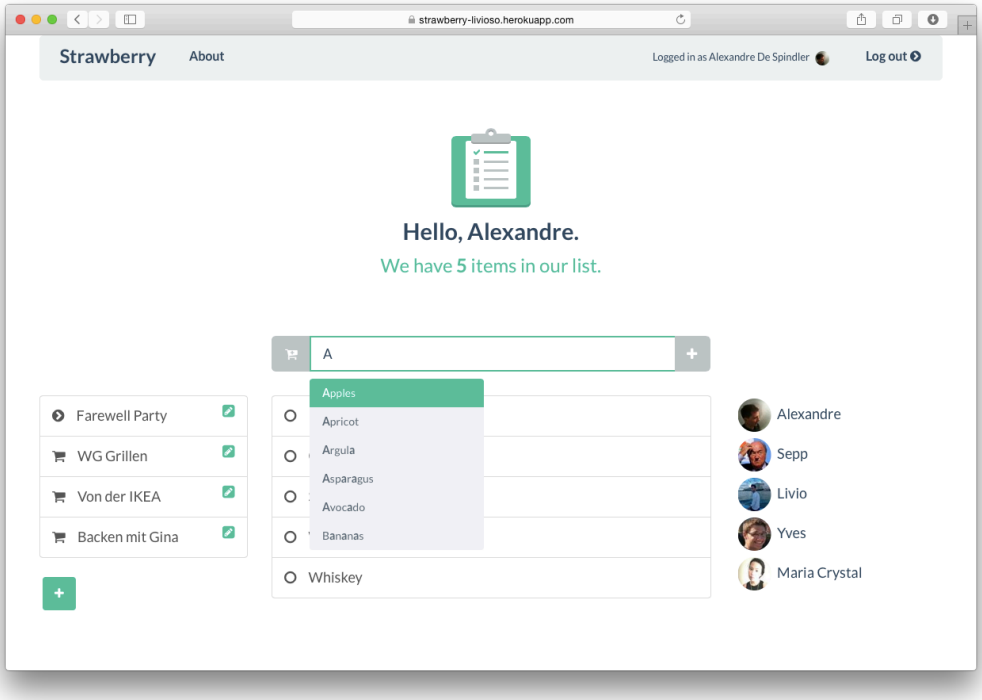
5.1. Final vs Mockup

- Login and Landing Page are very similar to original Mockups.
- Settings is now separate per list (the green edit button).
- Instead of just names we have profile pictures.

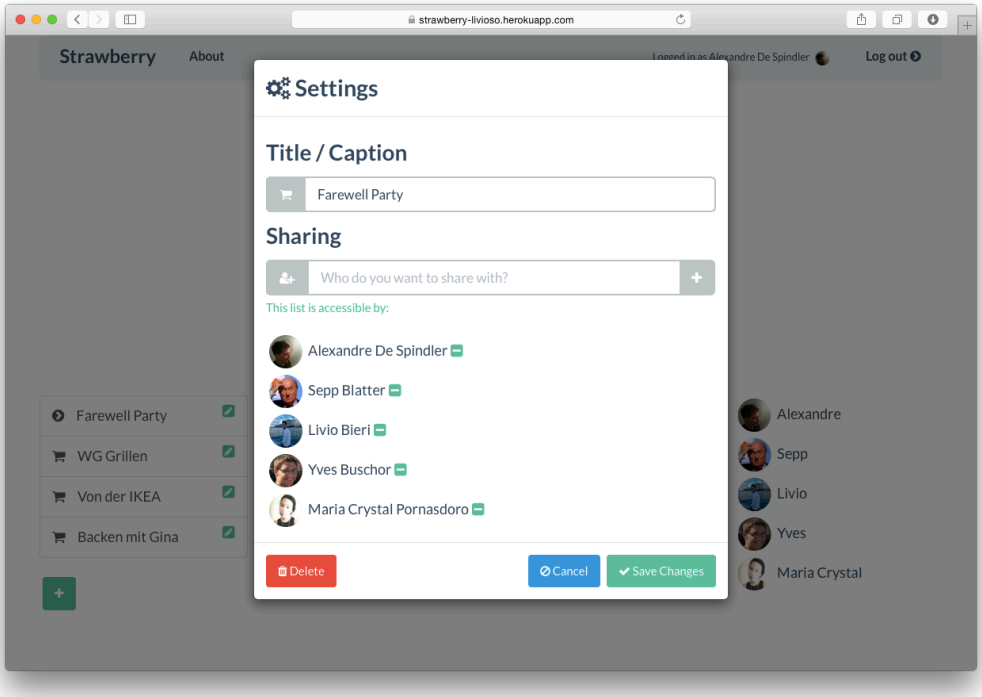
5.1.1. Main Page



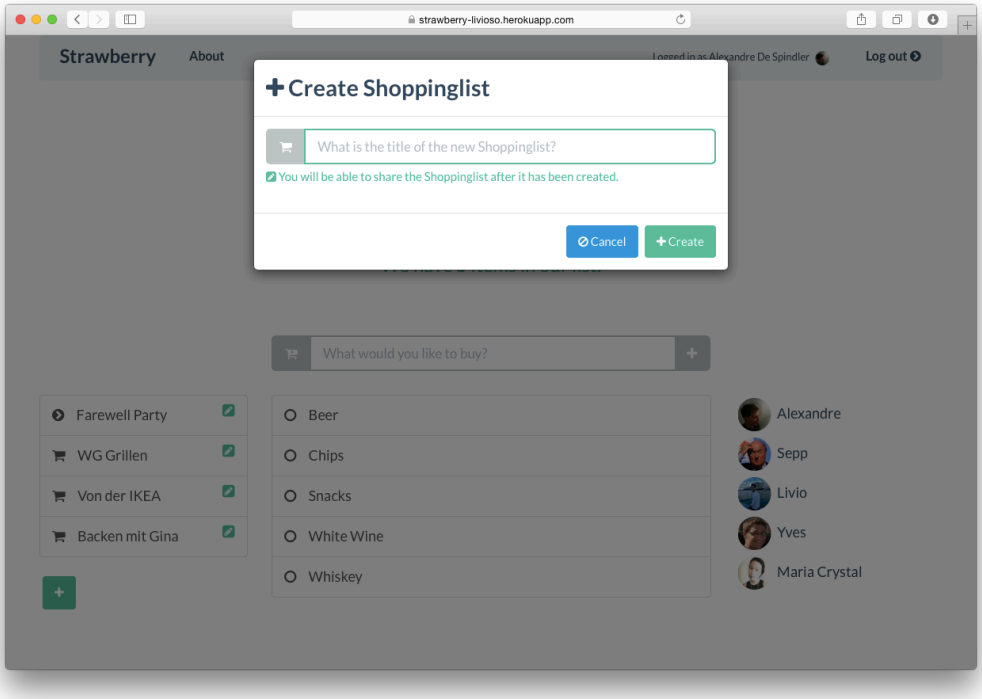
5.1.2. Main Page - Typeahead



5.1.3. Settings



5.1.4. Create



5.2. API

- **Incomplete list of possible API calls.**
- **For more information please check out routes.js.**

5.2.1. PUT `/api/list`

- Can be executed by authenticated user.
- Parameter with MIME-Type `application/json`.

Example Parameter:

```
{
    "name": "My new cool list"
}
```

5.2.2. GET `/api/list`

- Can be executed by authenticated user.
- Returns all the users lists.

5.2.3. POST `/api/list/:idlist`

- Can be executed by authenticated user.
- Update name of existing list.
- and/or members of existing list.
- and/or items of existing list (with `:listid`).
- Parameter with MIME-Type `application/json`.

Example Parameter:

```
{
    "name": "My new cool list name",
    "members": ["0193472304", "0564667234"],
    "items": [{"_id": "09293932", "checked": true}]
}
```

5.2.4. GET `/api/list/:idlist`

- Can be executed by authenticated user.
- Returns all items of given list.

5.2.5. DELETE `/api/list/:idlist`

- Can be executed by authenticated user.
- Deletes the list permanently.

5.2.6. GET `/api/user`

- Can be executed by authenticated user.
- Gets information about the logged in user.

5.2.7. POST `/login`

- Simple login with username + password.
- Or Facebook (see `/auth/facebook`).

5.2.8. POST `/logout`

- Can be executed by authenticated user.
- Logs the currently signed in user out.

5.2.9. GET `/auth/facebook`

- Sign in via Facebook.
- Redirects to Facebook and tries to authenticate user.
- Facebook redirects back to `/auth/facebook/callback` with success / failure message.

5.2.10. GET `api/category`

- Can be executed by authenticated user.
- Returns an array of products that are used for autocompletion / typeahead.

5.3. Technologies

5.3.1. Server Side

- Node.js Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications.
- Express.js Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It's responsible for the API routing e.g. `/api/list/id`.
- Passport.js Passport is authentication middleware for Node.js. Extremely flexible and modular, Passport can be unobtrusively dropped in to any Express-based web application. A comprehensive set of strategies support authentication using a username and password, Facebook, Twitter, and more. Strawberry uses the username and password as well as the Facebook Strategy.
- MongoDB MongoDB is a document-oriented database. Classified as a NoSQL database, MongoDB eschews the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas, making the integration of data in certain types of applications easier and faster.

5.3.2. Client Side

- AngularJS AngularJS is a web application framework maintained by Google and by a community of individual developers and corporations to address many of the challenges encountered in developing single-page applications. It aims to simplify both the development and the testing of such applications by providing a framework for client-side model–view–controller (MVC) and model-view-viewmodel (MVVM) architectures, along with components commonly used in rich Internet applications.
- Bootstrap Bootstrap is a sleek, intuitive, and powerful mobile first front-end framework for faster and easier web development.

- Flat UI is a Bootstrap Framework design and theme for flat UIs.
- Typeahead.js a flexible JavaScript library that provides a strong foundation for building robust typeaheads / auto completion. Used for the product auto-completion with a preset of over 200 commonly used products.

5.3.3. Development Tools

- Bower A package manager for the web. Responsible for installation of the front end frameworks (such as Bootstrap, Flat UI Kit, etc.) *See **bower.json** for a complete list of used client side packages.*
- npm node package manager. Installs, publishes and manages node programs. *See **package.json** for a complete list of used packages for node.*
- Gulp Gulp is a build automation system. Automates tasks such as building and deploying etc. *See **gulpfile.js** for more.*
- Jasmine Jasmine is a Behavior Driven Development testing framework for JavaScript. *For more check out the Strawberry **unit tests**.*
- TravisCI Continuous Integration Service. If all tests pass, commits to GitHub will be automatically deployed on Heroku.
- Jshint Linter for JavaScript. Helps to detect errors and potential problems in code. *See **.jshintrc** for the detailed configuration of Jshint. Commits that violate Jshint will be rejected automatically. (see git pre commit hook)*
- JSCS JSCS is a code style linter for programmatically enforcing your style guide. *See **.jscsrc** for the detailed configuration of JSCS. Commits that violate this style guide will be rejected automatically (see git pre commit hook)*

5.3.4. Authentication

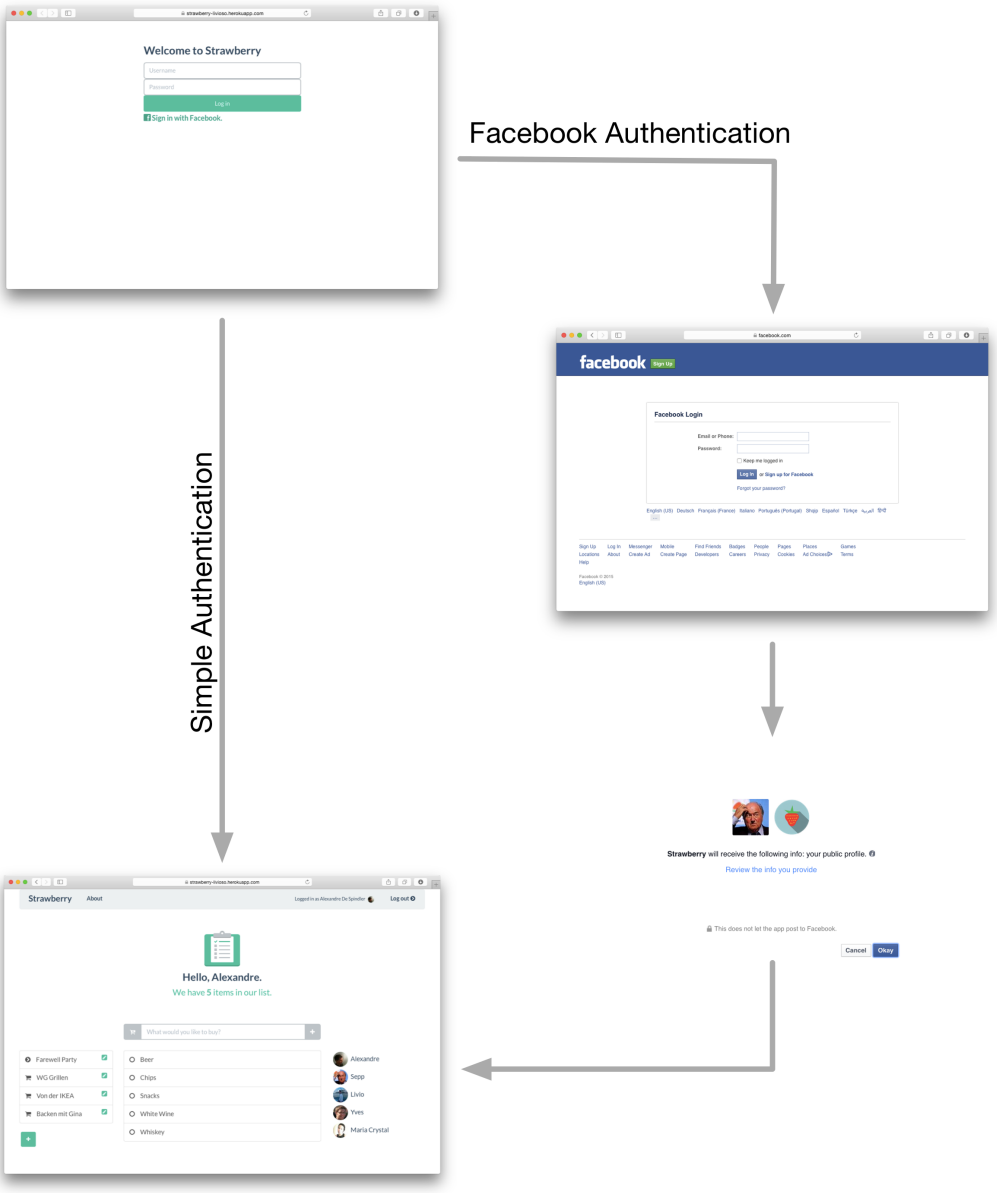
Sign in with Facebook

While there is a simple login for some (dedicated) user, the main authentication method is via Facebook using Passport.js Facebook authentication strategy.

Strawberry only uses the most basic information provided by Facebook:

- Given and family name
- Facebook Profile ID (to get a profile picture via Graph API).

5. Implementation



5.3.5. Important Files & Folders

In order the understand the project it is highly advisable to browse its source code on Github. The following incomplete list of files is a good starting point though:

- **server.js:** General server setup and configuration.
- **app/routes.js:** Handles API requests / routes.
- **app/models/:** Contains the Models.
- **public/html/views/:** Contains the Views.
- **public/js/controllers/:** Contains the Controllers.
- **public/js/services/** Contains Services used by Controllers.
- **tests/:** Contains the Unit Tests.
- **config/:** Contains database and authentication configuration.

5.3.6. Installation

- **Please see the project Readme on Github for a detailed installation guide.**
- Installation: `npm install && bower install && gulp build`
- Start Server: `npm start`

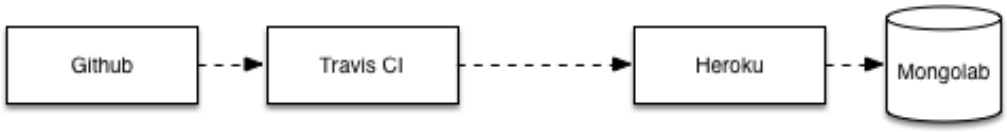
5.4. Next Steps

- Replace Flat UI with Angular Material (improve responsiveness of application).
- Refactor shoppingListController.js. This controller has way too many responsibilities. Violates SRP.

A. Appendix

A.1. Development Workflow

The development environment and workflow is straightforward:



By default there is a git pre-commit hook setup which will run the unit tests locally and ensure the code is according the guidelines (jscs, jshint). Once committed locally changes will be pushed to Github and Travis (Continues Integration) will compile the project and run the unit tests once more. As soon this is done and there are no errors the code will be deployed automatically on Heroku.

References