Foodfeed

DAT076 - Group 3

Group members:
Marcus Stigelid, 0705-294298, stigelid@student.chalmers.se
Marika Hansson, 0707-596377, hmarika@student.chalmers.se
Max Witt 0708-247070, wmax@student.chalmers.se
Sebastian Ljunggren, 0738-439290, sebljun@student.chalmers.se

Overview

FoodFeed is a web application for recipe sharing. One can create a user and log in to contribute with their own recipes, as well as view existing recipes without logging in.

Registered user: may create, edit and delete their own recipes.
Guests: may view existing recipes but cannot add, edit or delete them.

Table for usernames in Github repo

Kohina - Marika Hansson
kirayatail - Max Witt
marcusstigelid - Marcus Stigelid
sebastianljunggren - Sebastian Ljunggren

Use cases

- Register a new user
- Be able to log in
- Be able to log out
- Add a recipe when logged in
- Add ingredients to a recipe the user owns
- Remove ingredients from recipes
- Edit a recipe the user owns
- Show all recipes that has been added to the application
- Show a specific users recipes
- Search for a specific recipe
- Delete a recipe
- Be able to change password
- Autocomplete on ingredients

Future directions

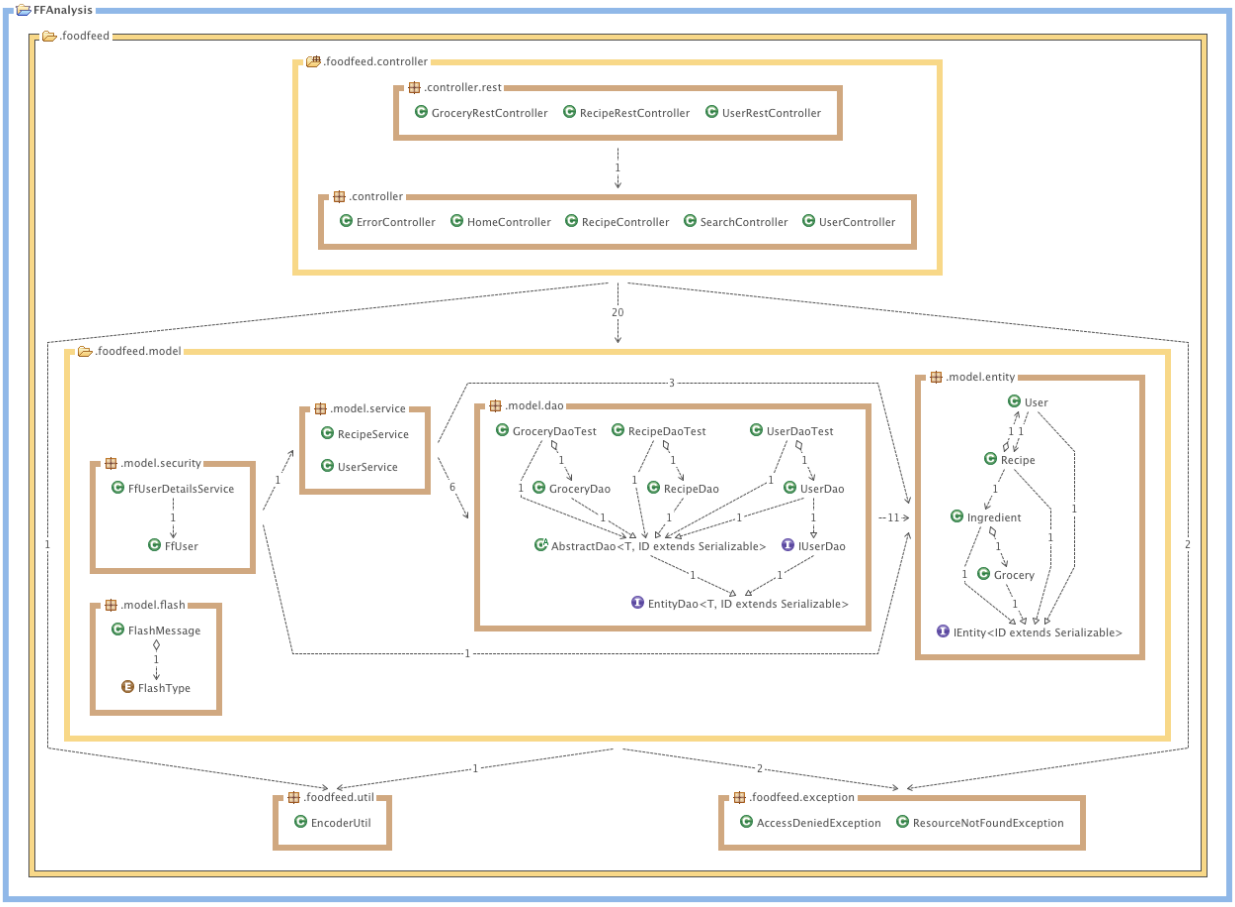
- Categorize recipes
- Show/search recipes by category
- Add images to recipes
- Implement mobile client

Technical Design

Tiers and responsibilities

- Persistence layer- actually saving the data
  - JPA with Hibernate and embedded Derby database
- Persistence model - define data relations (entities and DAO etc)
  - edu.chl.dat076.foodfeed.model.entity.*
  - edu.chl.dat076.foodfeed.model.dao.*
- Service layer - reusable business logic and control of transactions
  - edu.chl.dat076.foodfeed.model.service.*
- Controller - serve the view with data from the persistence tier, some web specific business logic
  - edu.chl.dat076.foodfeed.controller.*
- View - views for displaying data in browser
  - WEB-INF (.jspx pages)

UML (larger version: doc/structanalysis.png)



Important classes and annotations

- edu.chl.dat076.foodfeed.model.dao.AbstractDao
    - Implements general CRUD for entities
- edu.chl.dat076.foodfeed.controller.*
    - Manages communication between daos and views

From dependencies

- DispatcherServlet and @RequestMapping to have clean URLs and to avoid writing much code for identifying desired actions.
- @Transactional annotations and LocalContainerEntityManagerFactoryBean to automate creation of transactions and make it possible to write short and concise DAOs.
- Several classes from Spring Security that simplifies session creation and authentication.
- Apache Tiles to avoid writing more JSPX than you have to.

Used technologies

- Spring Framework 3.1 for annotations and autoinjection.
- JavaServer Pages for all views.
- CSS3 for some web elements.
- HTML5 for some web elements.
- GlassFish 3.x for the application server.
- Hibernate 4.x for ORM.
- Derby embedded database for the application database.
- Apache Tiles for reusable layouts. (see WEB-INF/tiles/tiles-config.xml).
- JQuery for adding and removing ingredients for a recipe and autocompletion.
- REST with JSON thanks to Jackson
- JavaScript for the styling of form when adding or editing a recipe.
- Aloha editor for editing HTML for recipe instructions.

Strong parts of the application

- Clean code thanks to the use of Spring Framework
- Complex model relation design
- Real life application
- Password encryption
- Snappy interface thanks to JS but all functionality has non-JS fallback.
- Easy deployment with Maven and Glassfish. (Just hit Run, no database configuration since it’s embedded)
- Clear division of responsibility between layers and use of autoinjection makes project prepared for testing (so far we only test the DAOs).
- Prepared for mobile client with REST API. All that is needed is authentication.

Weak parts of the application

- A bit hard to implement further if the developer is not used to Spring Framework
- Application still quite basic, not much bells and whistles
- Hard to manually check data in the database when running tests with embedded database