Private Grocery Manager

Programming Exercises

Barbar Ahmad

Giuliano Spagnuolo

Julius Sange

Examination Regulations

Of the Department 2: Computer Science and Engineering, Computer Sciences and Engineering The University of Applied Sciences Frankfurt am Main – University of Applied Sciences. For the degree of Computer Science Of 10. July 2003

Group 112

Lecturer: Ralf-Oliver Mevius

Barbar Ahmad	bahmad@stud.fra-uas.de	1 22 93 27
Giuliano Spagnuolo	spagnuol@stud.frauas.de	1 25 39 40
Julius Sange	sange@stud.fra-uas.de	1 25 66 75

Evaluation Form

Note:

Place, Date

Signature/lecturer(s

Table of Contents

1 Introduction

This document is the result of the Programming Exercise Project as written in the Lecture Notes by Mr. Ralf-Oliver Mevius. The module is part of the curriculum for the Bachelor program in computer science at the University of Applied Sciences Frankfurt in the summer semester 2020.

1.1 Goal Definition

The main idea of this application is a service to manage groceries. You can scan products with their barcodes and add them to a database. With this data it is possible to create shopping lists and recipes. You get a fast overview of the Items in your household and know if there is everything at home to cook a certain meal. The goal is to have a central management application, where every member of a household can login and keep track of the current inventory and what is needed to buy. The potential users are students and in general people who live in a shared apartment.

1.2 Team Members

Barbar Ahmad	1229327	bahmad@stud.fra-uas.de
Giuliano Spagnuolo	1253940	spagnuol@stud.fra-uas.de
Julius Sange	1256675	sange@stud.fra-uas.de

1.3 Deliverable Requirements

The goal is to develop a Software which is connected to a relational Database and build with a client Server Architecture from the requirement analysis, the system design to the implementation of the software.

2 Analysis

2.1 Requirements Collection

In our first meeting we collected the following requirements

1. The User should be able to register into the Application
2. The User should be able to login into the Application
   (a) There should be a main Househould Account
   (b) A Household can have different members
3. The User should be able to Scan the barcode of the item
4. The User should be able to manually enter the barcode of the item
5. The User should be able to insert Informations about the Item
   (a) Name of the Product
   (b) The Barcode of the Product
      i. If it is not scannable
   (c) A Short description about the Product
   (d) Category of the Product
6. The user should be able to categorize the products
7. The user should be able to enter new recipes
8. The user should be able to link Products to recipes
9. The user should be able to manage Products
   (a) Deletion of Products
   (b) Update of Products
10. The user should be able to list not available products

2.2 Use case Diagram

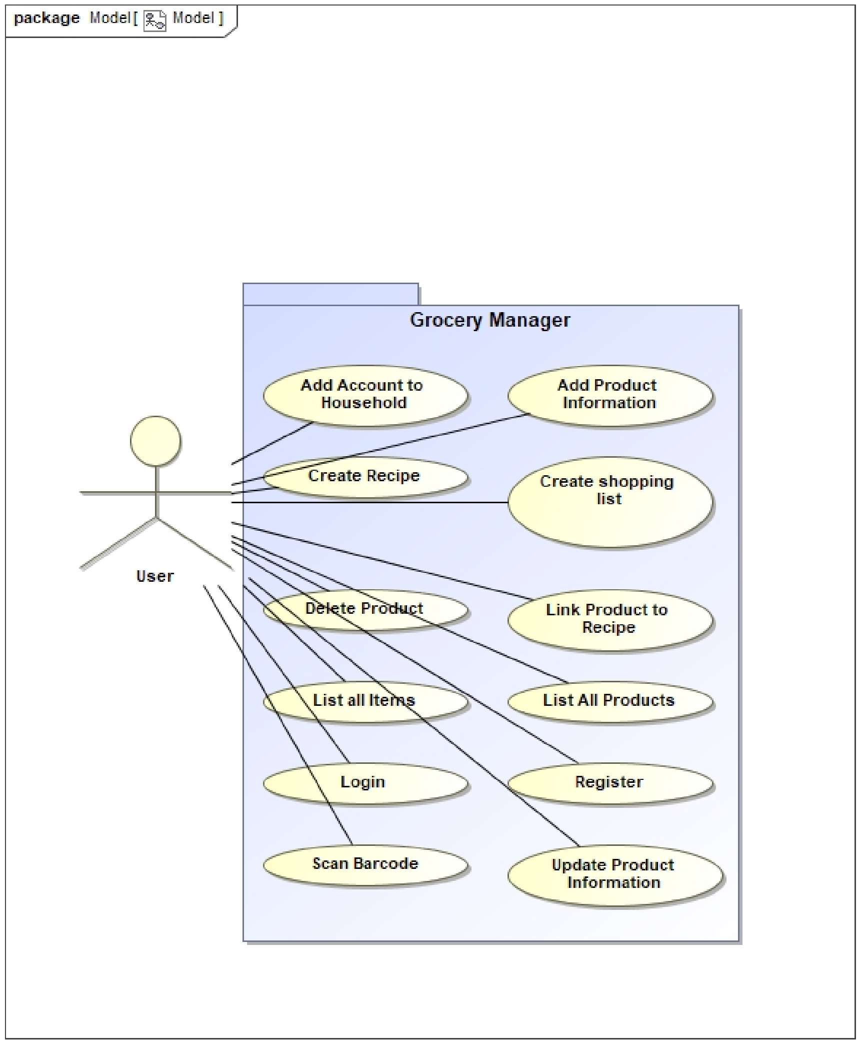


Figure 1: Use case diagram

2.3 Requirements Analysis (Use Cases)

In this section we created Use Cases for the most important functionality of the application

2.3.1 Customer Registration

Type:	Functional
Usecase Overview	The Customer can Register into the Web application. In order to do this, he has to enter his personal details. If the E-Mail address is already taken, an error message gets triggered.
Actors	User
Preconditions	• The User is not registered • The Email is not taken
Postconditions	• The User is registered
Basic Flow	1. User opens Webapplication 2. User navigates to Registration Page 3. User Enters Registration Data 4. User is registered
Estimation: 2	Implementation of Registration is a small operation, since many frameworks have pre build modules to handle it.

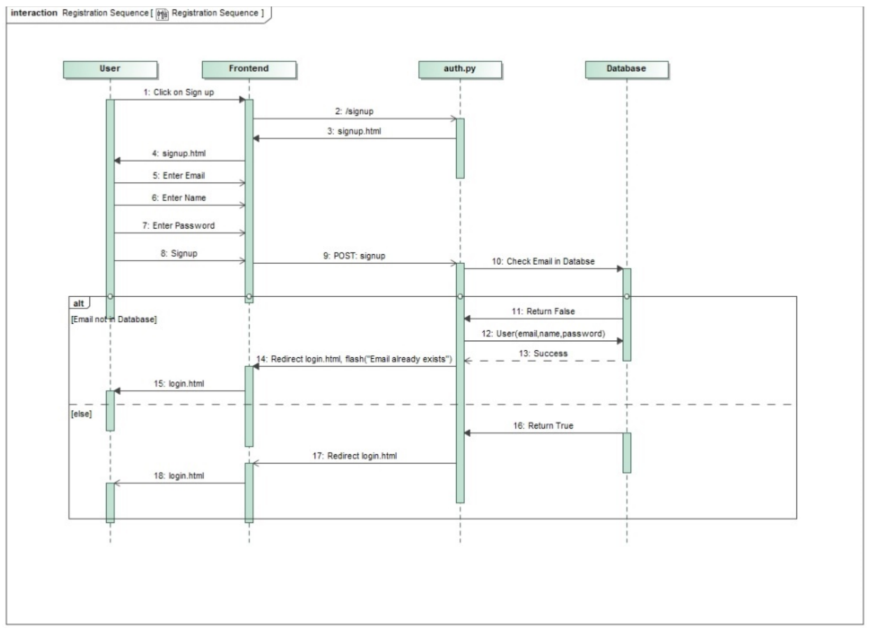


Figure 2 Sequence Diagram registration

2.3.2 Customer Login

Type:	Functional
Usecase Overview	The Customer can Login into the Web application. To do this, he has to enter his Email and his password. If the password is wrong or the email is not in the database, an error message gets triggered
Actors	User
Preconditions	• The User is registered
Postconditions	• The User is logged in
Basic Flow	1. User opens Web application 2. User navigates to Login Page 3. User Enters Login Data 4. User is logged in
Estimation: 2	Implementation of Login is a small operation, since many frameworks have prebuild modules to handle it.

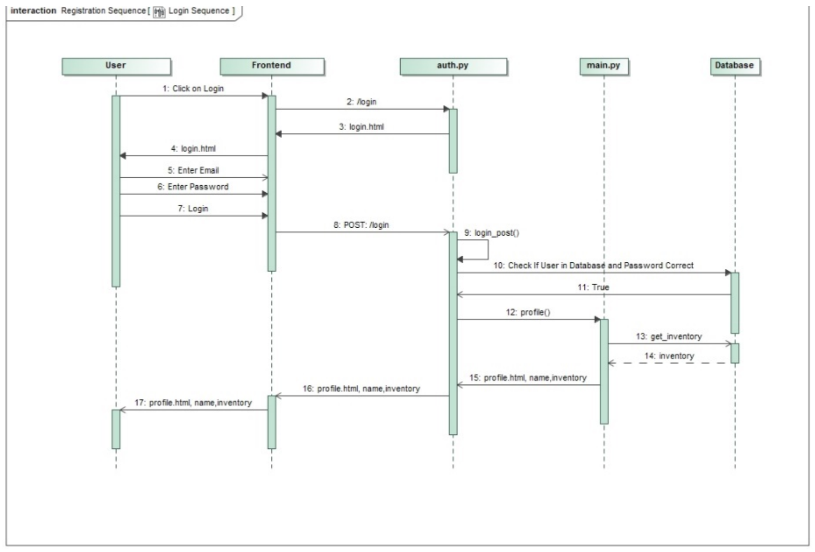


Figure 3: sequence diagram Login

2.3.3 Barcode Scanner

Type:	Functional
Usecase Overview	The User should be able to scan the barcode of an Item to add it to the database. In order to do this, he should navigate to the Scanner page, activate his camera and then scan the Item
Actors	User
Preconditions	• The User is logged in
Postconditions	• The User added the barcode
Basic Flow	1. The User navigates to the scanner page 2. The User allows access to the camera 3. The User scans the barcode 4. Barcode gets recognized
Alternative Flow	5. Barcode can`t get recognized a. User enters Barcode manually
Estimation: 6	Implementation of the Scanner is a medium task since there are Image recognition tools

	but you have to keep in mind the efficiency of the scanning

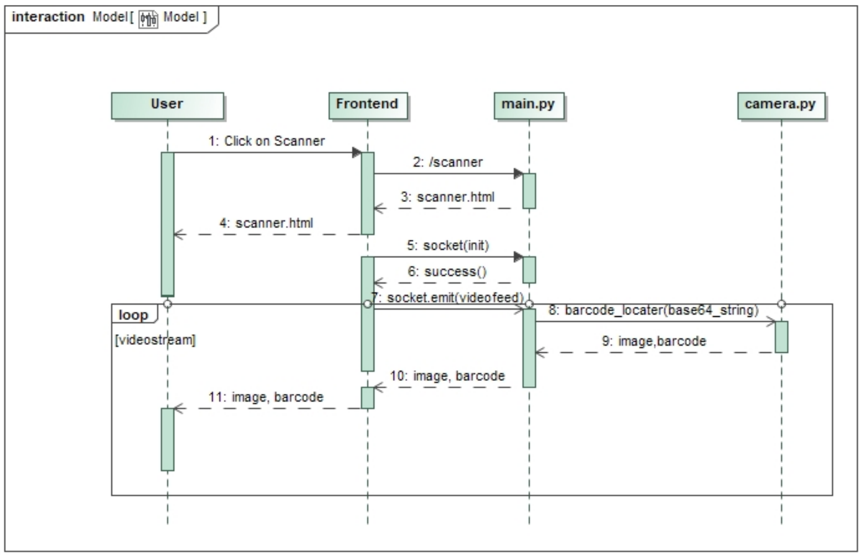


Figure 4 barcode scanner sequence

2.3.4 Insert Information about the Product

Type:	Functional
Usecase Overview	The User should be able to add the most crucial information about the product. 1. Name of the Prodcut 2. The Barcode of the Product (a) If it is not scannable 3. A Short description about the Product 4. Category of the Product
Actors	User
Preconditions	• The User is logged in
Postconditions	• The user Inserted the Product
Basic Flow	1. The User navigates to the new Item Page 2. The User Enters The Information about the new Item 3. Product gets saved to the database
Alternative Flow	4. Barcode already in Databse

	a. User just needs to add the new due date b. New Item gets saved into the Database
Estimation: 4	Insertion of the Product is a medium difficulty task, since you must keep track of the products to not add more than one

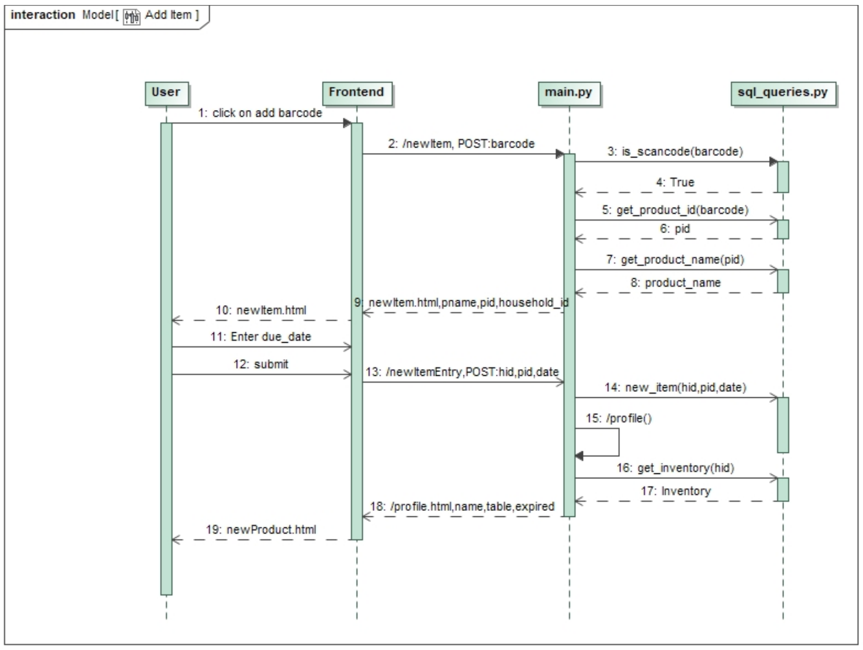


Figure 5: add item sequence diagram

2.3.5 Enter new Recipes

Type:	Functional
Usecase Overview	The user should be able to add new recipes to the database.
Actors	User
Preconditions	• The User is logged in
Postconditions	• New Recipe is added
Basic Flow	1. The User navigates to the profile page 2. The User click on Show all recipes 3. The User clicks on add Recipe 4. The user Enters Information about the recipe a. Recipe name b. Instructions c. Difficulty d. time 5. The User saves the recipe 6. The User adds new Items to the recipe

	7. Recipe Ingredients get saved to the database
Estimation: 6	Insertion of the Product is a medium to difficult task, since you must keep track of the products, and the recipes.

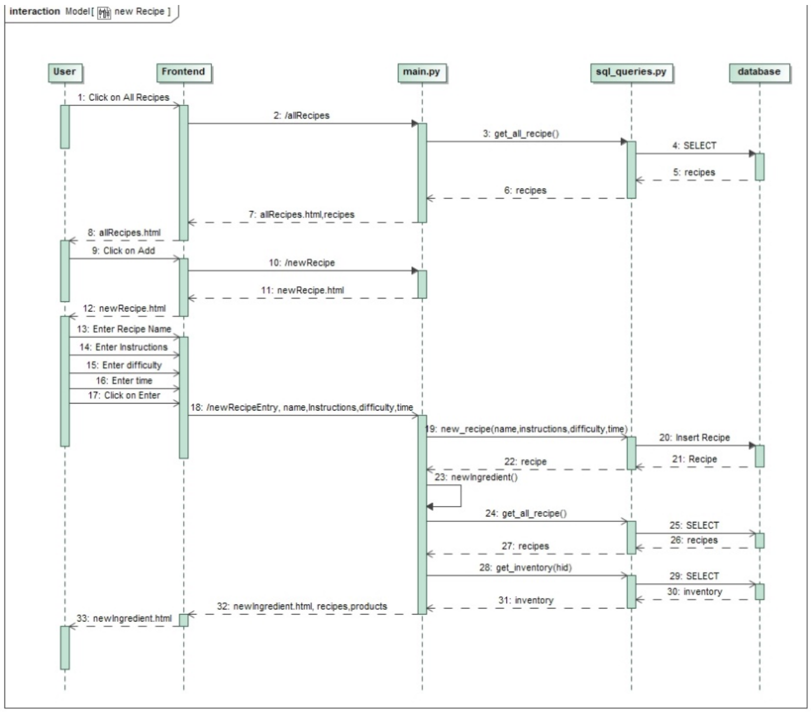


Figure 6 new recipe sequence diagram

3 Design

3.1 Application Architecture

The Application is built in an MVC pattern. Each Route is linked to a function within a controller. The application matches the entered URL to a route and if successful, it calls the controller action. The Model is used to retrieve all the necessary data from the database. After that, the result is passed to a view, which renders the page.

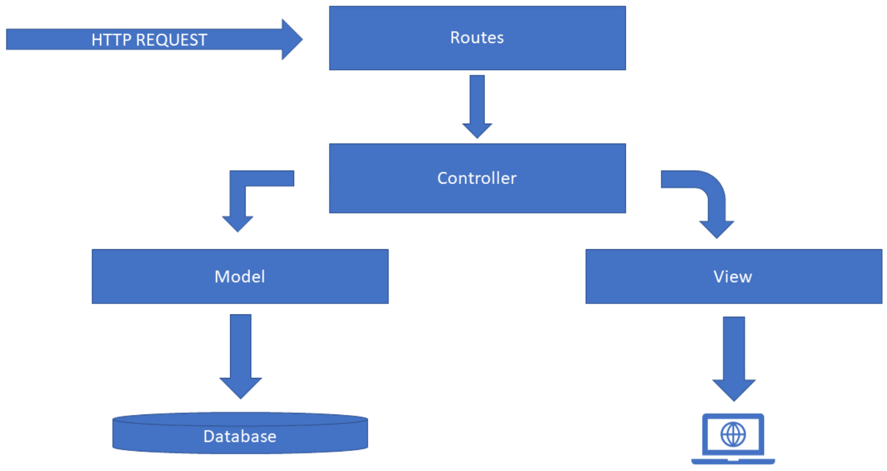


Figure 7 Application Architecture

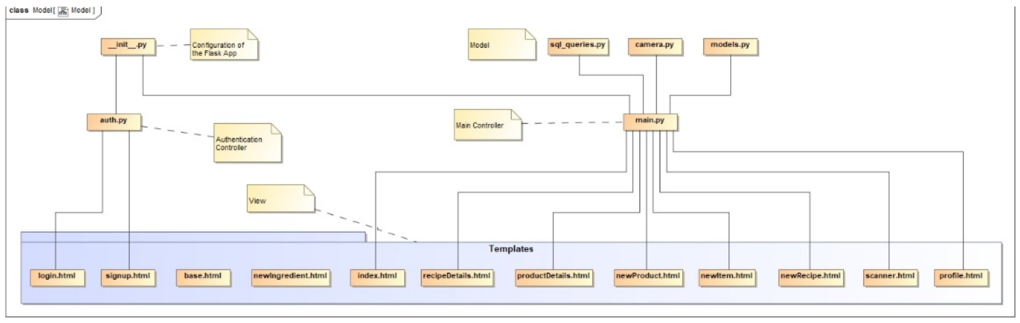
```
@main.route('/newProduct')
@login_required
def new_product(barcode=''):
    categories = db.get_all_product_categories()
    return render_template('newProduct.html', categories=categori
es, barcode=barcode)
```

3.1.1 Views

In the view, the data is accessed and gets rendered for the user to see as html content.

```
{% extends "base.html" %} {% block content %}


<div class="container index">
  <h1 class="title">
    Private Grocery Manager
  </h1>
  <h2 class="subtitle">
    Barbar Ahmad, Giuliano Spagnuolo, Julius Sange
  </h2>
<img class="transition-
swipe" src="{{url_for('static',filename='images/shopping.svg')}}"
></img>
{% if not current_user.is_authenticated %}
<a href="/signup" id="signup-btn" class="button is-block is-
info is-large ">Sign up Now</a>
{% endif %}
</div>
{% endblock %}
```



3.2 Database Architecture

Our database Schema consists of the following Entries:

- User:
    - User stores information about the user/Customer to identify him for a simple login procedure and link him to an household
- Household
    - It represents one instance of an inventory. It is separated from user to make it possible to have multiple customers access the same inventory
- Item:
    - Each line in Item represents an actual product in the managed household. It only stores a due date and is linked to an Entry in Product that contains the rest of the necessary information.
- Product
    - Product and Item is separated due to two reasons. One: even after removing/consuming an item the general Information about the type is not lost. Two: with the Separation we can track an individual due date for each item in the inventory.
- Recipe
    - Recipe provides the additional function of being able to store favourite meals and the necessary steps in a convenient way. Time is stored in Minutes. And the difficultly can be defined by the user him self in the data base it is a simple numeric value.

- RecipeIngredients
    - Recipe-ingredients provides a link between products and Recipes together with the needed amount. This enables us to cross check with the inventory and provide Information for shopping and planning.
- ScannCodes
    - This table provides a link between a scanned bar-code of an actual product with the corresponding database entry in product. This is not already in Product itself due to the fact that common products like for example a 1L carton of milk is produced by many different companies and thus have different bar-codes on them with out having to be necessarily to be differentiated in our database.

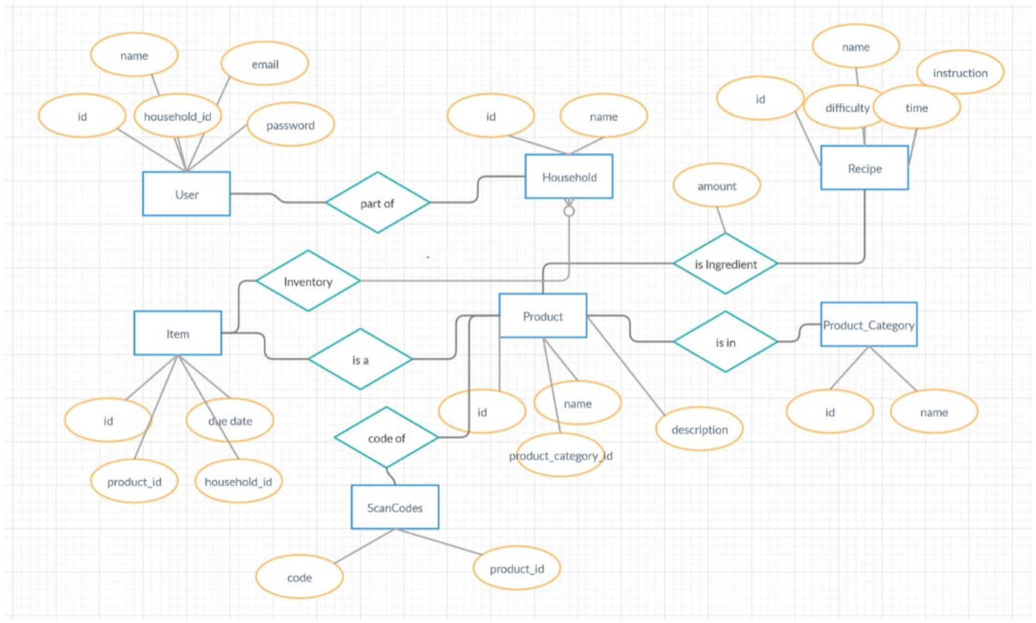


Figure 8: Relationship

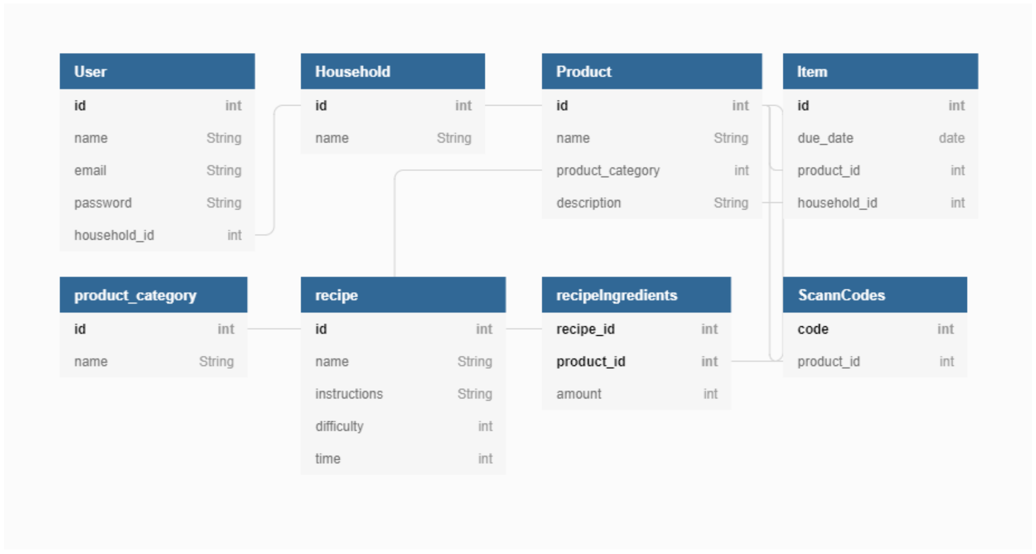


Figure 9: Database Schema

3.3 Technical Infrastructure

The following Diagram shows our technical Infrastructure

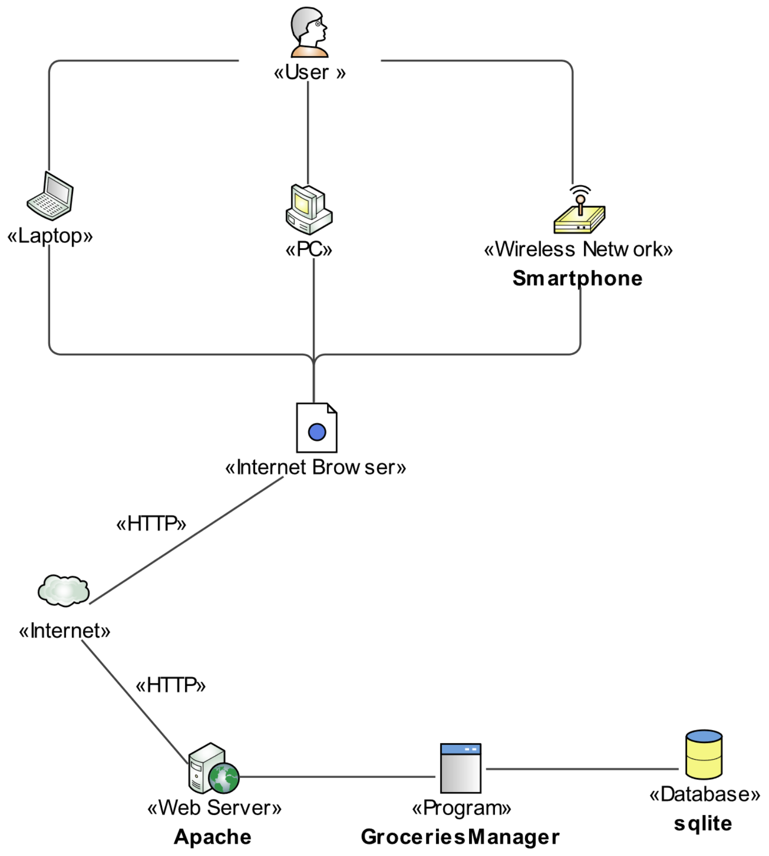


Figure 10 Technical Infrastructure

3.4 Mock-up (UI Prototype)

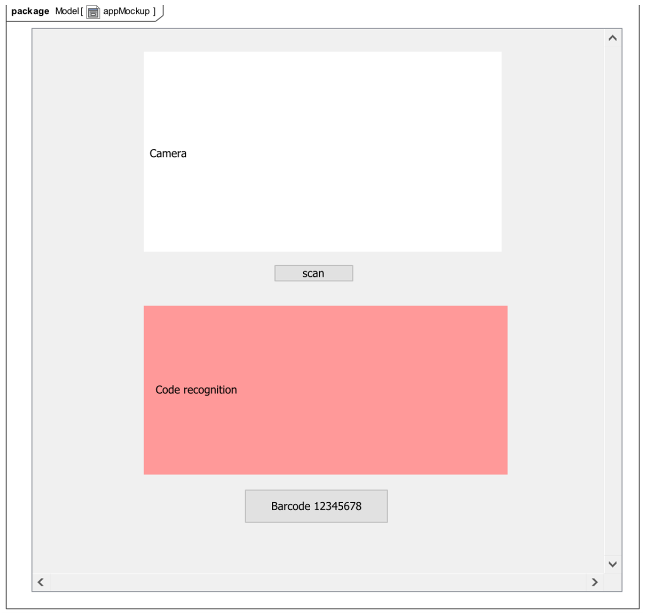
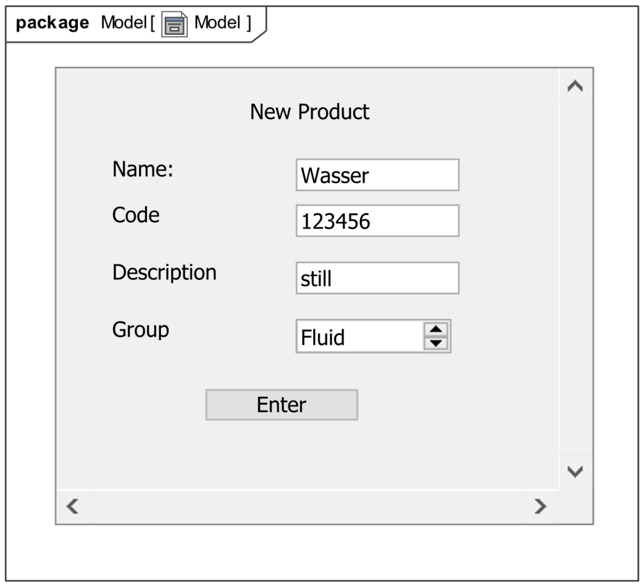


Figure 11 Barcode Scanner Mockup



Name:	Wasser
Code	123456
Description	still
Group	Fluid

Figure 12 Input Form Mockup

3.5 Technology Stack

3.5.1 Flask

The Software is written in Python. It uses the micro web framework “Flask”. It does not have any external dependencies and is well documented. It allows for rapid development of full stack applications and uses “Jinja” as its template engine which is easy to read and to write. The core of the framework is simple to develop in, but there are many extensions to integrate with. (Flask, 2020)

3.5.2 SQLite

We decided to use SQLite as our relational database because it is integrated in the python environment and easy to set up. It is small, fast, and full featured and is the most used database engine in the world. (SQLite, 2020)

3.5.3 Bulma

Bulma is a free, open source CSS framework which allows to rapidly style websites. (Bulma, 2020)

3.5.4 OpenCV

“Opencv is an open source computer vision and machine learning software library. It has more than 2500 optimized algorithms which can be used to detect and recognize faces, identify objects and more.” (OpenCV, 2020) It has Python support and an excellent documentation.

4 Programming

4.1 Software-Documentation

The following section documents the most important functions in our codebase.

4.1.1 ___init___.py

```
class AdminView(ModelView):
    """
    Overwrite View of the Admin Dashboard to show
    Primary Keys and foreign Keys
    """
def create_app():
    """Initialize The Flask application.
    Returns:
        [app]: returns the Flask application
    """
```

4.1.2 Auth.py

Controller to handle User Authentication in the Web application

```
@auth.route('/login')
def login():
    """Returns the template for the login Page.
    Returns:
        [render_template]: [login.html]
    """


@auth.route('/login', methods=['POST'])
def login_post():
    """
    Login controller. Checks if Email and Password are in the
    database
    and returns the profile view if successful.
    """
@auth.route('/signup')
def signup():
    """
    Controller for the signup Process.
    Returns the Signup Viev
    """
@auth.route('/signup', methods=['POST'])
def signup_post():
    """
    Registration Controller Function. Checks if
    Email Adress exists. If it doesn`t it adds
    a new user to the database.
    Returns view for the login

    """
@auth.route('/logout')
@login_required
def logout():
    """
    Logout Controller. Returns Index view
    """
```

4.1.3 Camera.py

Python module to handle the image recognition part of the scanner.

```
def barcode_locater(image_data):
    """
     Gets the Picture and tries to find a barcode.
    If yes then return the barcode and the manipulated picture
    else try again. Returns the Image and the barcode if found,
    else just returns the image
    """
```

4.1.4 Main.py

This is the main controller of the web application.

```
@sio.on('videostream')
def checkbarcode(data):
    """Listens to the socket and
    waits for a message with key = videostream.
    Checks if the barcode is in the database
    Args:
        data ([base64]): base64 String of an image
    """
@main.route('/profile')
@login_required
def profile():
    """
    Controller for the profile route.
    Checks the inventory of the current user.
    Checks if there are any expired items in the inventory
    and renders warning messages
    Returns profile view
    """
@main.route('/newProduct')
@login_required
def new_product(barcode=''):
    """
    Controller for the newProduct Route
    Gets all categories from the database and
    returns the view to add a new product

    Args:
        barcode (str, optional): [description]. Defaults to ''.
    """
@main.route('/newProductEntry', methods=['POST'])
@login_required
def new_product_entry():
    """
    Add a new product to the database.
    getparams:
        name
        group
        description
        barcode

    """
@main.route('/newItem', methods=['POST'])
@login_required
def new_item():
    """
    Controller to add a new Item to the database.
    Checks if barcode is in POST parameters. If
    the barcode is already in the database, it
    return the newItem view. Else it returns the
    newProduct view. If there are no POST parameters
    it returns the new product view.
    """

@main.route('/newItemEntry', methods=['POST'])
@login_required
def new_item_entry():
    """
    Add a new Item to the database.
    Returns Profile view
    """

@main.route('/deleteItem/<item_id>')
@login_required
def delete_item(item_id):
    """
    Delete an Item from the database.
    returns the allProducts view with the inventory
    in the context

@main.route('/allRecipes')
def all_recipes():
    """
    Gets all recipes from the model and returns the
    allRecipes view with the recipes in the context.
    """

@main.route('/recipeDetails/<recipe_id>', methods=['POST', 'GET']
)
@login_required
def recipe_details(recipe_id):
    """
    Gets the details to an recipe (due date)
    Returns the recipeDetails view with the
    details and the inventory in the context

    Args:
        recipe_id ([int]): [Id of the recipe]
    """


@main.route('/newRecipe')
@login_required
def new_recipe():
    """
    Returns the newRecipe View
    """

@main.route('/newIngredientEntry', methods=['POST'])
@login_required
def new_ingridient_entry():
    """
    Add a new ingredient entry to the database
    Returns the new ingredient view.
    postparams:
        recipe_id
        product_id
        amount
    """

@main.route('/scanner')
@login_required
def scanner():
    """
    Returns the scanner view
    """
@main.route('/scan', methods=['POST'])
@login_required
def scan():
    """
    returns the recognized barcode and the
    manipulated image in the scannerView.
    """






@main.route('/allProducts')
```

```
@login_required
def allProducts():
    """
    returns the allProducts view. Shows
    all products currently available in the inventory
    """

@main.route('/productDetails/<product_id>')
@login_required
def product_details(product_id):
    """
    gets the details (due_date) from an product and
    returs the productDetails view.

    Args:
        product_id ([int]): ID of the product
    """

@main.route("/settings")
@login_required
def settings():
    """
    returns the settings view

    """
@main.route("/settings/newCategory", methods=['POST', 'GET'])
def newCategory():
    """
    Add a new category to the database.
    returns the newCategory view

    """
```

4.2 User Manual

To run the software on localhost, Python 3.8 is needed.

1) pip install -r requirements.txt
2) activate virtual environment
   a. Windows: src/venv/Scripts/activate
   b. Linux: src/venv/bin/activate
3) Start the Flask development Server
   a. Windows: run.bat
   b. Linux: run.sh

5 Declaration of Authorship / Eidesstattliche Versicherung

„Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit mit dem Titel „Private Grocery Manager“, selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter der Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form an irgendeiner Stelle als Prüfungsleistung vorgelegt worden. Die Anmeldung beim Prüfungsamt der FH-Frankfurt für die Prüfung ist erfolgt.

Frankfurt am Main, 25.06.2020

[illegible]

Barbar Ahmad

Spagnuolo

Giuliano Spagnuolo

[illegible]

Julius Sange

6 Sources

[1] Bulma, 2020. *Bulma.* [Online] Available at: https://bulma.io/documentation/overview/start/

[2] Flask, 2020. *Flask.* [Online] Available at: https://flask.palletsprojects.com/en/1.1.x/foreword/

[3] OpenCV, 2020. *OpenCV.* [Online] Available at: https://opencv.org/about/ [Zugriff am 25 06 2020].

[4] SQLite, 2020. *SQLite.* [Online] Available at: https://www.sqlite.org/index.html