Introduction

Making coffee is a part of most people's daily life. Some need it to get going in the morning, others meet in the afternoon and enjoy it in company. People use it on any occasion as a reason to take a break. Just like there are different occasions to drink it, there are different methods to acquire it; the coffee shop next door, a vending machine or the coffee machine in the kitchen. In an office it often occurs that colleagues share a machine. But who is responsible for buying new beans, the cleaning or payment? These are tasks that no one likes to take over voluntarily. So what exactly could a fair distribution of those activities look like, and how are they executed most easily? This question calls for an IT supported solution that could be used to track who drinks how many cups and when, in order to balance their accounts appropriately. Furthermore, statistical data can be collected that ensures a fair distribution of the cleaning tasks. Those aspects motivated us, Cornelius Pohl, Daniel Birnstiel, and Theresa Zobel to develop a solution for coffee machine administration in the office in the context of the tBPM seminar, during the summer term 2015, at the BPT chair of professor Weske. The goal was to create a software that could work largely autonomously and simplify any employee's work day.

Organisational Process

General

Likewise many other software projects, we used for coffe-in-the-cloud the scrum technique. That means we organised the project with Jira, which is an issue tracking software developed by Altassian. Our main communication tool for the team itself was Slack and Facebook messanger. We used Slack as it provides the possibility to integrate git notifications and many more. We had weekly standup meetings and biweekly sprints.

Functional Requirements - Userstories

All the user stories where assessed and discussed beforehand. The effort and the core features were determined. In the following the userstories are explained in detail according to their sprints.

Sprint 1

Add Account

The admin can add an account to the system including first name, last name, email and a picture. The password will be created automatically and will be

sent to the usa via an email. Additionally he can activate his account thourgh this email.

Edit Account

When the user is logged in, he is able to do some changes. This includes the email adresse, the password and the profile picture.

Remove Account

The admin is able to remove an account if requested.

Login

The user can authenticate through username and password.

Picture Login

The user is displayed a set of picture on a sreen. Via clicking or touching on a picture, the user can authenticate. Next to the picture will be the name of the user.

Coffee Tracking

When logged in, the user can add one or more coffees. He can undo this.

Cleaning Schedule

The user can see the cleaning schedule for the upcomming weak.

Sprint 2

Email Reminder - Cleaning

When a user is assigned for cleaning he will get an email reminder. This reminder can be turned off in his personal settings.

Email Reminder - Account Balance

When the user's balance goes below a certain amount he is notified via email. This reminder can be turned off in his personal settings.

Email Confirmation - Coffee Tracking

When tracking coffee(s) the user gets a notification via email if he choses to. Using this email he can also undo the tracking within a certain time frame.

Statistics

From all data gathered over the statistics will be generated. The user can opt out of having his data displayed to others.

Checklist: Cleaning

A checklist provided on the tablet and mobile application helps the user to keep track of each step involved in the cleaning process.

Tutorials: Photo

To support the processes of cleaning and coffee making a photo tutorial is provided.

Sprint 3

Cleaning Schedule: Intelligent Assignment

The responsible user for cleaning is chosen by the system. It will take into account if somebody has marked himself as absent for a certain amount of time.

Guest Account

For tracking coffees a guest account exists. It allows guests of the chair to also track their coffees without having an own account. * The account should be accessible by the admin (login with admin credentials) * It should behave like a normal user account, e.g. it includes statistics * This account can be enabled or disabled by system administrators * A summary email should be send to the admin every X days (configurable)

User Rankings

Rankings show some kind of leaderboard according to the amount of coffee consumed by each user in a certain time frame. Users can opt out of having their data displayed.

Display total amount of money available

As an administrator I want to see how much money is available for buying coffee. This should be shown in the admin backend. When new coffee is bought the admin needs to reduce the total amount.

Add cleaning checklist to tablet mode

The user wants to see the cleaning checklist although he is not logged in.

Distinguish between weekly and bi-weekly cleaning in calender

As a user I want to see which kind of cleaning I should perform. Use different colors.

Show a message on the main page if selected for cleaning

As a user I want to see whether I have to clean. This should be visible on the tablet as well as on the user page.

Settings page for users

As a user I want to be able to configure my account on a settings page. * email configuration * profile pic * password

Extend cleaning checklist

Add restart Button and Finish Button to the coffee checklist. Cross out finished tasks

Include calender entry invitation in cleaning email

As a user I want that a calendar invitation is included in the email that informs me about my cleaning schedule.

Blame feature

I want to have a feature in tablet mode to blame someone who did not clean the porta filter. An "reminder" email is sent to the last coffee drinker (add something like "it might be possible that someone else did not clean but did not register the coffee").

XP-Techniques

In order to work efficiently and fullfill our users requirements as good as possible, we used some of the practices of extreme programming. In our opinion this was really successfull.

small releases

We intregrated code early and often. As a result we had less bugs and always a running version of our software. Additonally we got the feedback quicker and it also incresed our self confidence.

pair programming

In the course of the weeks we realized, that we did much better work when developing together. Therefore we often implementen user stories in pairs where one was the driver (types code) and the partner tried to be completly engaged and understand everything.

collective code ownership

As we often worked together on user stories, everybody was allowed to change the code of the others. This was reasonable because it avoids expert knowlegde.

AngularJs

In the Coffee in the Cloud-Application we use angularjs for our frontend and user interface. AnuglarJs is a javascript framework that allows the creation of

MVC (Model-View-Controller) based web applications. Because of this the code will stay properly divided into three core parts that are described below.

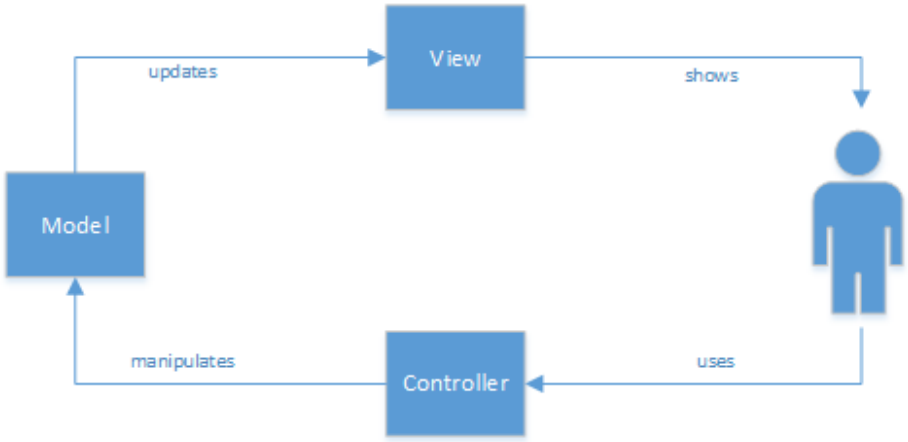


Figure 1: mvc architecture

In order to manage dependencies we use the node package manager (short npm) and bower. These allow installing of dependencies in a bulk. Because of this we do not have to ship them and can keep the code base clear.

The application loads all dependencies asynchronously through require.js whenever they are needed. Because of this we have short loading times and can make sure that everything is loaded in the right order. Require.js also provides dependency injection so we do not have to worry about initializing controllers and services ourselves.

As we do not want the application to reload every time the user performs an action we use a routing system. This allows us to have multiple views within one application that are loaded dynamically.

Models

Models contain all kind of data we would like to use, for example users, tally list or calendar entries. In our application these are created and managed server side. The server also handles persisting the data into our database. Because of this the models are replaced by a service that fetches data from our server.

Controllers

Controllers are necessary to create useful data from the models we have stored. For every page that will ultimately be displayed in our application a controller

fetches the data and prepares it for the user. This also includes validation of user generated information.

The controllers will be both implemented server and client side as they will manage the connection between these to parts of our application.

Views

Views are responsible for displaying all data to the user. In the views we create a user interface that will be displayed on the client's browser. AngularJs extends the HTMl syntax and functionality to allow the creation of basic templates in these views.

Each view is tied to a controller that determines its behavior.

Directives

AngularJs offers a way to extend basic HTML. These so called directives introduce new tags or attributes.

Project structure

The client project contains of several type of resources that are separated in different folders.

/index.html contains the main application and entry point.

/js/ contains controllers, directives, services and libraries installed by bower.

/less/ contains less stylesheets that will be compiled to css on deployment.

/public/ contains static images and stylesheets.

/views/ contains views as HTML files.

All other files in the client directory are resource files needed for bower and node to include all needed dependencies.

Finding a backend framework

As our application needed a web service to run permanently we first had to decide on a framework. We narrowed down our options to three different technologies some of us had already worked with.

Flask

Flask is a lightweight framework for creating web services written in Python. It offers basic routing and has has close to no overhead. While it is very easy to use it only provides very basic functionality and thus we would have to implement many core features like session management or database-models ourselves. Also flask has no strict project structure which could lead to an unstructured code base if not maintained properly. Because we didn't want to take this risk we decided against the flask framework.

node.js

Node.js is a powerful framework for creating web services written in Javascript. As we were using node.js' package manager for managing our front end dependencies it seemed natural to use node.js for our backend as well. The downside to node.js was that none of us had already built a more complex platform using it, so we would have to slowly learn all necessary features. We also had no knowledge of available APIs and functionality which means we would have to familiarize ourselves with these before actually starting to develop our app.

Django

Django is a web framework also written in Python. It has a more complex structure but includes a lot of additional functionality we would not have to implement ourselves. The core features we rely on are session management, object-relationship-management for creating our database and url-routing for the API endpoints.

We had some experience in working with django from previous projects and it integrated perfectly with our requirements.

In addition to django we are using the `rest_framework` library to ease the creation of a restful API. This framework provides features to automatically create endpoints from models, data serialisation and validation.

Django - a short introduction

A django project consists of several modules, which are thematically separated. Each module contains models, views and other classes needed.

Django follows the MVC architecture, separating data, logic and display. Because of this it is possible to develop logic and interface separately from each other and maintain a clean project structure.

Models

A model is a class that describes the data structure of objects. Instances of these models are automatically stored into and loaded from our database. Also changes are automatically tracked and can be reverted at any time.

Models are defined in each module within a `models.py` file. This file then contains one class for each model. These in general are a subclass of the type `django.db.models.Model` and define all fields as python data types. In addition to simple data types like numbers and text django also provides complex data types like dates or files which are automatically validated. It supports all features that are important for a relational database.

Apart from fields a model can also specify special behavior, for example for automatic validation when editing field values or relationships between models.

Permissions

A permission allows access restriction based on users and groups. The built in permissions can be extended to allow custom control. These can then be assigned to specific users or user groups.

Permissions can be set automatically by the system or through the administration interface.

Views

A view is responsible for converting model data into viewable information. In our case each view either loads models from the database and converts then into a json response or processes a request and changes a model's state. We separated our angularjs front end from the backend so our views will only represent the needed API endpoints and will not create viewable HTML.

As we are using the `rest_framework` extension we do not have to handle json conversion and object validation ourselves. Each view can so define different methods for viewing or manipulating data.

For example a `list` method allows automatic rendering of an object list. `post` and `get` respectively handle POST and GET requests via HTTP.

The `rest_framework` also provides some debugging utility. For example when accessing an endpoint directly it displays additional information like available methods and allows drafting new requests. Apart from testing purposes this can also be used as a developer documentation.

In order to grant an application access to a newly created view it has to be registered in the routing system. This can be found in the `urls.py` file and lists all API endpoints available.

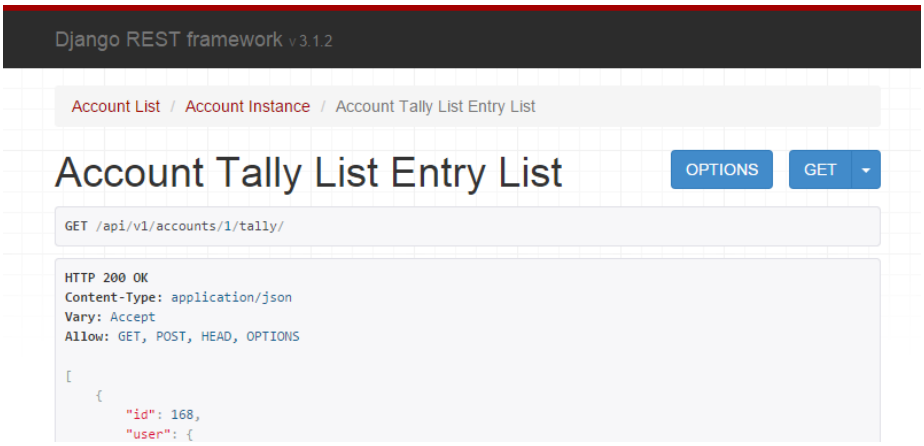


Figure 2: API endpoint when opened in a web browser

Serializers

A serializer is responsible for converting a model into a json representation. In most cases it defines a collection of fields that are publicly accessible. For special cases like account creation it is possible to add custom validation behavior, which in this example validates the password and then updates session information.

Migrations

Migrations are a powerful tool to alter models without manually changing the database structure. They can be created automatically by django after a model has been changed. It also allows for multiple developers to change the models and then merge all changes. For example when a new field is added to a model the author can define a default value to be set when an existing database is upgraded. In some cases this is necessary to not break any existing validation constraints.

A new migration can be created through the server console via the `manage.py makemigrations` command. Afterwards all pending migrations can be applied by using `manage.py migrate`. This step is not necessary when creating a new database as django automatically applies the needed changes.

Management/Commands

Each module can define special commands to be used from the management console (`manage.py`). These in general are used for administration purposes and

can only be accessed from the server side. In our application this is used for handling automatic schedule assignment.

Commands can be used to run a special task periodically by connecting them with a scheduling tool like crontab.

Templates

A template contains formatted text that can be dynamically filled with information. Usually these are used for creating dynamic HTML responses, in our application we only use then for the email functionality.

Static file deployment

In our application we use django to deploy all static files including our angularjs app. Because of this there is no need for a secondary web server like apache. Any request that does not match a defined endpoint will be resolved within a given directory which is specified inside the server configuration.

Settings

To configure the server django provides a configuration file which can be found at `/server/server/settings.py` file. The most important settings are

INSTALLED_APPS: A list of installed modules and frameworks in this application. When a new module is created it has to be added to this list manually.

DATABASES: A list of available databases that can be accessed by the application. By default an sqlite3 connection is configured here.

LANGUAGE_CODE, TIME_ZONE: This setting provides localisation. By default it is set to an English locale and the Central European Time zone.

STATICFILES_DIRS: A list of paths where django will search for static files. For our application it is important to include the client directory because otherwise it cannot be deployed.

MEDIA_ROOT: A directory where django will store uploaded files which should be given as an absolute path. This directory has to be writable because otherwise any uploads will fail.

Custom settings

Apart from default settings we use the configuration file to add our own options.

AUTH_USER_MODEL: Name of the desired user model. In our application we use an own model which includes additional fields.

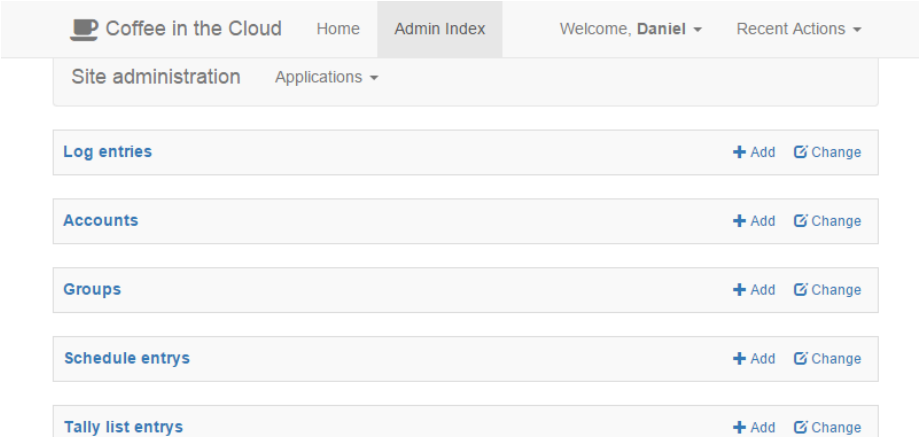
COFFEE_PRICE: The price of a single coffee in euros.

MAIL_SERVER, MAIL_SENDER: Email configuration for notifications, set to the HPI server by default.

Administration interface

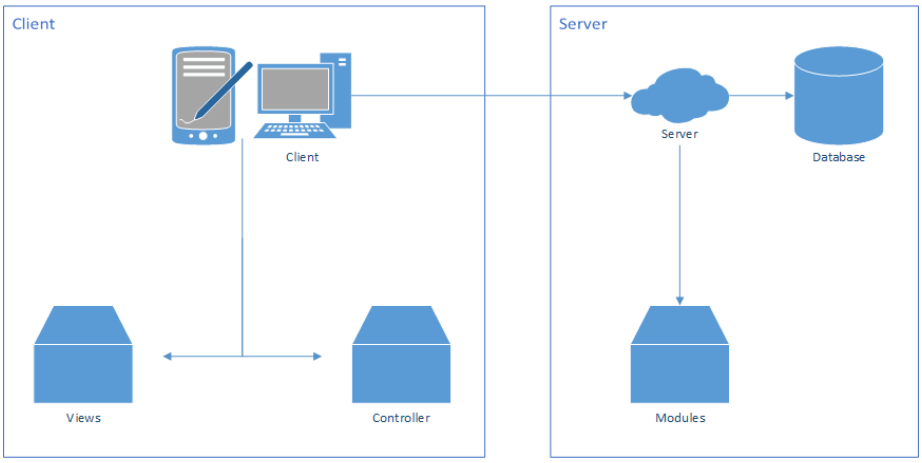
Django provides an administration interface which allows all users tagged as `staff` to manipulate models directly. This can be used to correct errors or create objects manually. Every action taken in the administration interface will be logged in case some unwanted changes are made.

A common usage would be to change a user's password, in case they forgot it.



Architectural Overview

The Coffee-in-the-Cloud application consists of a separated client and server part. Both communicate through a common interface and will be described more in detail within the next pages.



Frontend Architecture

The frontend for our Coffee-in-the-Cloud application is written in Javascript using among others the angularJS and requiereJS framework and bootstrap.

In the following, we will explain our services and afterwards the controllers together with their views

Services

As some functionality was needed on multiple occasions we grouped them into services.

coffeeCloud service

The `coffeeCloud` service is responsible for communicating with our backend. Each function sends a request with the desired information to the server. All requests use the deferred API to allow chaining actions together and enable parallel processing.

The requests are divided into three categories:

user mainly provides authentication functionality and allows querying user objects.

tally is responsible for adding or removing entries to a user's tally list.

schedule provides functions to query the cleaning schedule.

statistics fetches relevant data for generating statistics and preprocesses it.

settings allows fetching and storing a user's settings.

balance allows a user to manage the global balance.

blame encapsulates the blame feature.

status service

The `status` service is responsible for displaying notifications to the user. These are based on bootstrap alerts and are dynamically generated. The service supports information, success, warning and error messages.

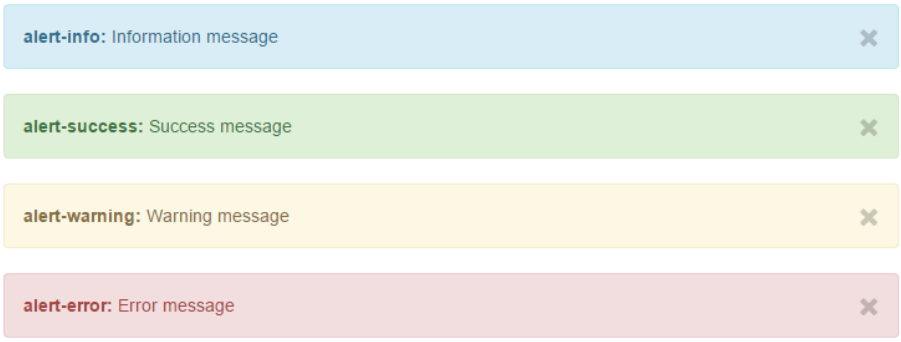


Figure 3: list of available statuses

Controllers and Views

Main Page - Home

The **Main Page** is the first page seen when visiting the coffee-in-the-cloud page. It consists of the `welcomeController.js`and the view `welcome.html`.

welcomeController.js

The main task of the controller is to give an alert-message saying "Thank you for doing the cleaning!". This happens after someone did the cleaning and pressed the *Finished Cleaning* button. In case, something an erorr occured during the function call, another allert will tell you so.

welcome.html

First of all the welcome view has to states depending on whether you are logged in or not.

Logged out

When first visiting the page without being logged in, you get greeted and you have the possibility to log in via a "sing in" button. Futhermore one is able to click on the Tutorials and Cleaning-Checklist navbar tab or click on the icons for tablet mode and admin interface in the footer.

Logged In

When logged in, different kind of banners can me seen. The one in blue, which is shown always, states how many coffees you have on the list and your current balance. If this balance is below 2.00 Euro, another red banner appears and kindly asks you to add more money to your account. By clicking on the x, this banner disappears.

In case you are selected for cleaning, a third banner in yellow can be seen beneath. There you can either close the banner by clicking on the x or tell the system, that you sucessfully finished the cleaning by clicking the check. This can alternatively be done when pressing the "finished cleaning"-Button.

Moreover, you will get complimented if you are the number 1 coffee drinker. This is again shown by a blue banner stating "You have drunk most of the coffee so far. Good job!". It can be closed aswell. The data used for estimating the winner, are loaded beforehand in the loginController. (see below).

Of course, when logged in, one has access to all the other navbar tabs like Statistics or Tally List too.

Login

For Logging in via the Login Button, on the one hand the `loginController.js` and on the other hand the `login-popup.html` view is needed.

loginController.js

The loginController has a lot of functionality as it loads all the necessary data beforehand. For reason of having value code and not any unnecessary code repetitions we outsourced function, which are called more often by diffrent views, in the loginController.

As mentioned in the paragraph before, the ranking is calculatet here. So everytime someone logs in, the controller compares the tally list entries of all the users and estimate the winner(s).

One of the global function in the loginController is *updateUser()*. This function is called in several occasions. For example, when the user logs in. It is responsible for updating the currently logged in user's data and loads the user object and his tally list from the server. Additionally it is checked whether or not the user is selected for cleaning.

The second relatively huge global method is *updateTally()*, called when you add a coffee to your tallylist.

When the user wants to logout and presses the "sign out" button, the controller accomplish the logout process by calling the *logout()* function.

login-popup.html

The Login-Popup appears when the "sing in" button is clicked on. The user can now enter his login credentials composed of an email address and a password. By

clicking “Login” the *login()* function is called in the loginController. Otherwise the user can abort the login process by simply pressing the “cancel” button. Once logged in, a short alert message welcomes the user and the user has now full access to the web-application.

Tally List

In order to see the current coffee consumption the user visits the **Tally List** tab. Responsible here is the `tallylistController.js` and the corresponding `tallylist.html` view.

tallylistController.js

The **tallylistController** has two rather obvious methods. On the one hand, when a coffee is added to the tallylist the *addCoffee(amount)* function is called. The amount depends on whether the user decides to drink a single or a double coffee. In the following the two global methods *updateTally()* and *updateUser()* from the LoginController are called.

On the other hand, the user has the possibility to revise the adding within two hours. Therefore the *removeCoffee(id)* is called with the matching coffee ID which causes the entry to be removed from the users tallylist.

tallylist.html

Like on the *Main Page* the user is shown a banner with his current balance and the amount of coffees on the list. Additonally if the balance is low, a warning appears.

The user has now the possibilty to add a single or a double coffee to his tallylist by pressing the corresponding button. This will trigger the *addCoffee(amount)* function in the **tallylistContoller**. Of course, after adding the coffee, the banners are updated with the new amount of coffees and balance.

As mentioned beforehand the user is able to revise the adding of the coffee by clicking on the dustbin symbol. This is only possible within the next two hours of adding.

Cleaning Schedule

As the cleaning of the coffee machine has to be scheduled, we build a calendar composed of the `scheduleController.js` and the `schedule.html`.

scheduleController.js & schedule.html

Basically the controller sends a request to the backend in order to receive the necessary data for the calendar. How the algorithm works, will be explained later on. The data important for us is the person and the type of cleaning assigned. To distinguish the diffrent kinds of cleaning we assigned them different colors.

Additionally if the cleaning is finished, it will be shown as crossed out und in gray. Of course one can select between daily, weekly and monthly view in the calendar.

Statistics

In order to compare his own coffee consumption with the consumption of others, we thought about some usefull diagramms to show different comparisons. In the end we came up with four kinds:

1. the overall coffee consumption
2. the users coffee consumption
3. the amount of single and double coffee consumed
4. the users coffee consumption compared to the overall ones

The JavaScript framework *chart.js* supplied us with some interesting chart types for example the "Radar-Chart" or "Doughnut-Chart".

statisticsController.js & **statistics.html**

The procedure is rather simple. The statisticsController sends a request to the backend in order to get the necessary data. Then the data will be process according to the requierements of the diagramm type. Ensuing the diagramm is filled with the matching data.

Tutorials

The turorials page is one of the pages, which can be seen, even if the user is not logged in. It's just a simple photo tutorial, where the user gets a step by step instruction on how to use the coffee machine correctly.

tutorialsController.js & **tutorials.html**

The whole process is rather simple. At the begining the user sees two pictures and can deceide whether to drink a single or a double coffee. After some more instructions, the user has to choose between adding milk, water or nothing to his coffee. Depending on the decision the matching photos are shown.

In the *tutorialsController.js* the important method is *next(cb)*. It is called by clicking on the picture directly or the arrows next to the pictures. This function then determines as the name says, the following pictures.

Cleaning Checklist

Likewise the photo tutorial shows the user how to make perfect coffee, the cleaning checklist should support the cleaning of the machine.

cleaningController.js & **cleaning-checklist.html**

In this controller all the steps are included. Now depending on weekly or biweekly cleaning the right steps are shown to the user. The user choose the type of cleaning by clicking the right button. All the steps are listed and can be crossed out when finished. It is also possible so reset the list and start from the begining.

After the cleaning is done, the user can mark is cleaning duty as done by simply pressing the *Finished Cleaning* button, which is located at the bottom of the page or at the main page.

Picture Login - Tablet Mode

One of the core features requiered was the tablet mode. Here the user doesn't need to log in with his login credentials bur rather click on his picture to add a coffee to his account.

pictureloginController.js & **picture-login.html**

As the user does not login with his credentials, he has limited access to the website. That means, he is only able to see the tutorials and the cleaning checklist. However if someone is selected for cleaning, a banner will tell you so.

The procedure of adding a coffee in tablet mode is nearly the same as adding a coffee when logged in. So again after selecting the coffee size *updateTally()* is called.

Another feature espacially implemented for the tablet mode is the blame button. If someone clicks on this button, the person last adding a coffe will get a blame email and a promt to tidy up the kitchen.

Settings

The Settings page has two states, deping on whether the user is an admin or not.

settings.html

basic user

The user is able to change is avatar for the picture login aswell as his password. Additionally the user can decice, if he wants the data about his coffee consumption to be evaluated in the statistics and ranking. As receiving emails can be really annoying, we decided on an option to disable all notifications.

admin user

The admin has more power over the global balance and is able to add money to users balance.

settingsController.js

The controller contains two important methods. *update()* is called, when the user wants to change his password. As usually, the user has to enter his new password twice and the function then verifies the password.

Moreover as mentioned before, the balance of a user can be changed by the admin. If so, *update_balance()* is executed and shows an alert-message if succeeded.

Backend Architecture

The backend for our Coffee-in-the-Cloud application is written in Python using the django web service framework.

In the current application release the following modules are used.

database layout

Each model is represented by a data table in our relational database.

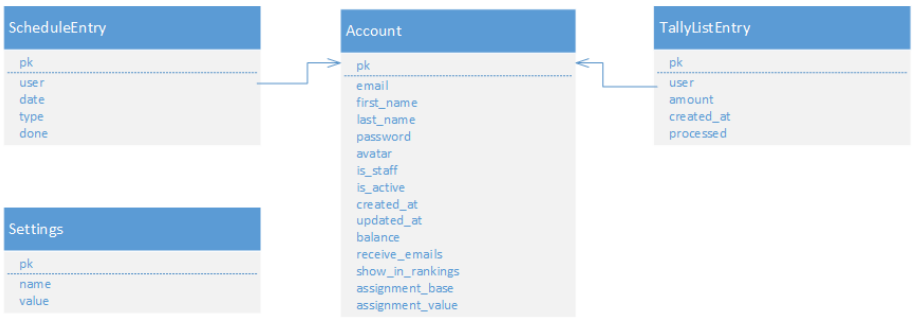


Figure 4: class diagram

authentication

The `authentication` module is responsible for user management and extends the built in system django uses. While django provides a simple authentication and security system we decided to extend it in order to allow more customization.

API Endpoints

`GET /api/v1/accounts/` Query a list of all registered accounts.

`GET /api/v1/accounts/{ID}/` Query an account specified by {ID}.

`POST /api/v1/auth/login/` Try to login a user with email and password.

`POST/api/v1/auth/logout/` Logout the current user.

`GET /api/v1/auth/status/` Query the login status of the current user.

`GET+POST /api/v1/auth/settings/` Query or update the current user's settings.

Models

Account This model extends the basic user model and extends to add neccessary fields like email address, name, profile picture or user balance. Also different user settings are stored there. In order to replace the built in user model with our own we added an `AccountManager` class which handles account creation.

Settings This model allows application wide settings. Currently it is used to store the global available balance.

Permissions

IsAccountOwner This permission checks if the currently logged in user is the owner of the given account. This is needed in order to prevent users from editing another user's account.

IsBalanceAdministrator This permission checks if the currently logged in user is allowed to manage a user's balance and view/change the global balance.

Views

AccountViewSet This view allows querying all accounts with all details.

LoginView This view handles and validates user logins.

LogoutView This view handles user logouts.

StatusView This view allows querying the current login status. If the user is logged in he will receive all account information as well as available permissions.

SettingsView This view allows the user to view and change their settings. This includes changing the profile picture.

schedule

The `schedule` module is responsible for the cleaning schedule. It provides models for schedule entries and does the automatic cleaning assignment.

API Endpoints

`GET /api/v1/schedule/` Query the current cleaning schedule.

`GET /api/v1/schedule/{ID}/` Query a specific schedule entry.

`POST /api/v1/schedule/done/` Mark the current assignment as done.

Models

ScheduleEntry This model represents one entry in the cleaning schedule and contains user, type and date information. It overrides the saving behavior to automatically send an email.

Views

ScheduleEntryViewSet This view allows querying all schedule entries.

ScheduleDoneView This view allows marking the currently assigned cleaning as done.

Commands

assignusers This command assigns users for cleaing. It can be called by using `manage.py assignusers <numberOfWeeks>`. The algorithm used is described more in detail in the `Module Description` chapter.

statistics

The `statistics` module is responsible for aggregating information about the coffee consumption. It defines no own models or permissions.

API Endpoints

`/api/v1/statistics/` Query statistics for all tally list entries.

`/api/v1/statistics/own/` Query statistics for the current user's tally list entries.

`/api/v1/statistics/type/` Query statistics by coffee type (single/double).

Views

StatisticsView This view groups all coffees by months and returns them to the application.

StatisticsOwnView This view groups the current user's coffees by months and returns them to the application.

StatisticsCoffeeTypeView This view groups all coffees by the amount (single or double) and returns them to the application.

tallylist

The `tallylist` module is responsible for tracking coffees. It provides the basic tracking functionality as well as additional features.

API Endpoints

`GET /api/v1/tally/` Query the current user's tally list.

`GET /api/v1/tally-all/` Query all tally list entries.

`GET /api/v1/tally/{ID}/` Query a specific tally list entry.

`GET /api/v1/accounts/{ID}/tally/` Query a specific user's tally list.

`GET+POST /api/v1/manage/balance/` Query and modify the global balance.

`GET+POST /api/v1/blame/` Blame the last user to have tracked a coffee.

Models

TallyListEntry This model represents one entry on the tally list. It contains functionality to automatically notify the user when the coffee was tracked. Apart from that the user's balance gets updated aswell.

Permissions

IsTallyUser This permission checks if a tally list entry belongs to the current user.

IsRecentTally This permission check if a tally list entry is recent and thus can be removed.

Views

TallyListEntryViewSet This view allows fetching all tally list entries for the current user.

TallyListAllEntryViewSet This view allows fetching all tally list entries.

AccountTallyListEntryViewSet This view allows fetching or adding tally list entries for/to a specific user.

BlameView This view allows blaming the last user that tracked a coffee.

GlobalBalanceView This view allows viewing and updating the global and user specific balance. Accessing this view requires the `IsBalanceAdministrator` permission.

server

The `server` module acts as a configuration module for django.

mail This module contains functionality for sending notification emails.

settings This module contains basic django configuration, see the official documentation for details.

urls This module contains the endpoint configuration. New views have to be registered here in order to make them accessable. Also the deployment of static files and the front end is configured here.

wsgi This module contains startup information for deploying the server using wsgi. # Additional functionality

Not all of our features are publicly accessible and thus cannot be associated with one module. These will be described in the following paragraphs.

Cleaning assignment

Our application includes an algorithm to automatically assign users for cleaning. For example an assignment for 4 weeks can be done through the server console by using the command `manage.py assignusers 4`. Then the algorithm will assign users depending on their coffee consumption within the last four weeks.

When searching for an algorithm that met our needs we came across several ideas. Our first approach was to either randomly or evenly assign users for cleaning. This would create a distribution where everybody would clean the same amount of times. Even though this would be fair in a scenario where all users drink the same amount coffee we had some concerns because for example a person who drinks one coffee a month would have to clean as often as somebody who drinks a coffee every day.

A solution to this is called **priority elevation** which is an algorithm mostly used in operating system scheduling.

The basic principle is that every user has a value associated that will be used for selecting. Each time the user is not assigned for cleaning it will increase by a fixed amount. Because of this somebody who has not cleaned for a long time has a higher chance of being assigned. After a user was assigned his assignment value is reset to a base value. This decreases the chance of a user being assigned multiple times in a row.

In order to add more fairness regarding the coffee consumption we decided to increase each users' value based on the coffee consumption since the last assignment was done. This allows us to assign users who drink a lot of coffee more often while still having everybody assigned eventually.

The following flowchart illustrates the algorithm:

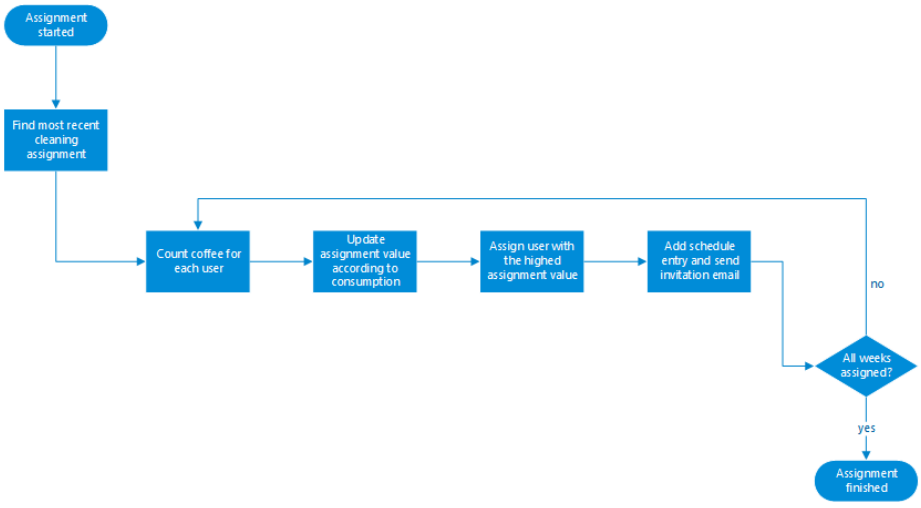


Figure 5: priority elevation algorithm

Coffee pricing

In order to change the price of a single coffee a server administrator has to change the project configuration which is found in the `settings.py` and change the `COFFEE_PRICE` setting to the desired value. The default value is `0.25` euros.

```
COFFEE_PRICE = 0.25
```

The price of a double coffee is automatically calculated.

Admin interface

Our application provides an administration interface to manage all models by hand. It can only be accessed by *staff* users. These have to be assigned by a server administrator. A staff member can only manage those models he was given the permissions for.

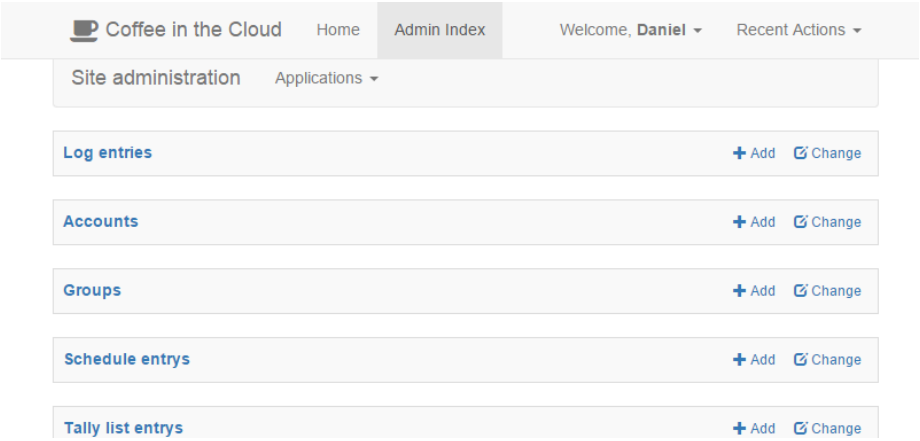


Figure 6: default administration view

This interface should be used with caution as we provided a front end solution for most necessary functionality.

Log Entries

Displays a list of administrative events. This can be useful to track unwanted access and modifications.

Accounts

Displays a list of all available accounts. An administrator can edit these accounts or create a new one. The account management can also be used to reset a user's password and to assign permissions or groups.

For every model django provides three different permissions that have to be assigned:

add The user can add a new object.

change The user can modify an existing object.

delete The user can remove an existing object from the database.

As long as a user has access to one of these permissions he will see the model in his administration interface.

Apart from individually assigning permissions to users it is possible to assign them to groups. These groups then apply the permissions to the selected users.

Groups

Displays a list of all available groups. It is possible for an administrator to add new groups or group members.

Schedule Entries

Displays a list of all schedule entries for the cleaning schedule. These can be manually edited in case a wrong or unwanted assignment occurred.

Tally List Entries

Displays a list of all tally list entries and their respective users. These can be added in case a coffee was wrongfully added and the default deleting time expired. # Application Setup and Deployment

In order to deploy our application you will need the following components installed on your system:

- Python 2.7 including PIP
- Apache including `mod_wsgi`
- Node Package Manager (npm)
- git

Clone the repository

At first you have to clone the repository from github.

```
git clone https://github.com/Birne94/Coffee-in-the-Cloud.git
```

Install client dependencies

Install bower and grunt.

```
npm -g install bower
npm -g install grunt
npm -g install grunt-cli
```

Install dependencies and compile less (run inside client directory).

```
npm install
bower install
grunt less
```

Install server dependencies

Install dependencies and create the database (run inside server directory).

```
python install-dependencies.py
python manage.py migrate
```

Copy the file `setup/settings2.py` to `server/server/` and adjust `MEDIA_ROOT` (absolute path).

Copy the file `setup/django.wsgi` to `server/apache/` and adjust absolute path names.

Configuring apache

Add the contents of the file `setup/apache.conf` to your apache configuration or include it.

Adjust the port, virtual host and absolute path names for the application and your local python installation.

Adjusting permissions

Make the `server` directory, `server/db.sqlite3`, `server/static/upload/` writable to everyone (`chmod 777`).

Restart apache

After restarting apache the application should be accessible. # User manual

Tablet

When accessing the application through the tablet in the kitchen you will see a list of users.

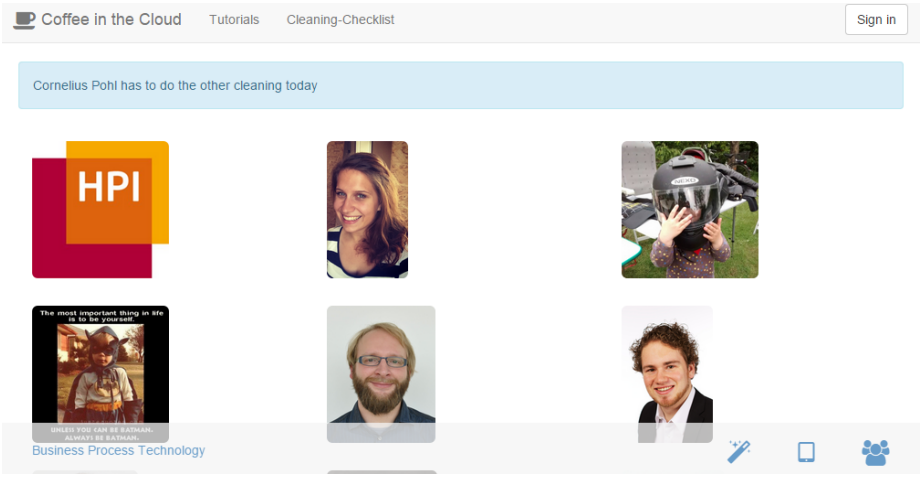


Figure 7: tablet screen

There you track a coffee without authenticating yourself by tapping on your profile picture. The list is sorted by the amount of coffee a person consumed so you might have to scroll down a little bit.

In the following dialog you can select either a single or a double coffee by tapping on the small (single) or large (double) cup.

In case the kitchen was dirty you can use the *Was the kitchen dirty?* button to message the person that tracked the most recent coffee.

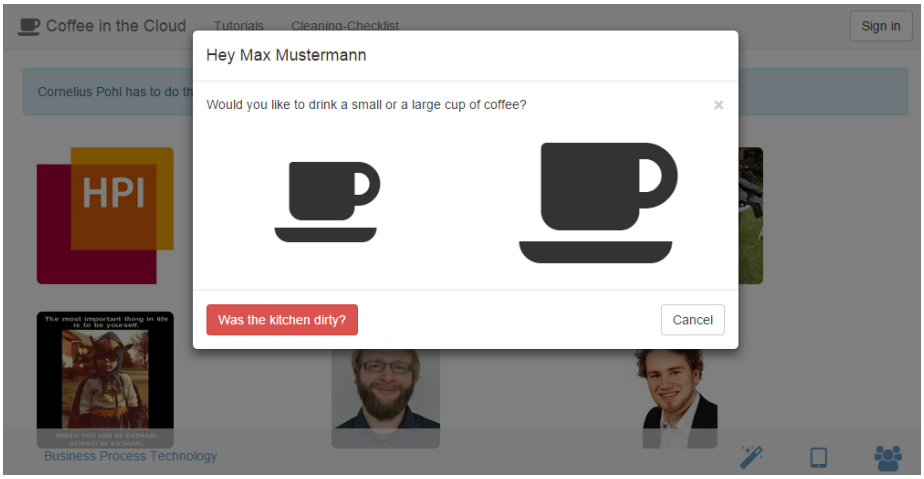


Figure 8: coffee tracking

Afterwards you should see a notification that your coffee has been tracked

successfully.

Tutorials

In case you are not familiar with the process of coffee making you can select the *Tutorials* menu. There you will find a picture tutorial with each step involved in making coffee.

You can switch to the next step by tapping on the current picture. If there is a decision to be made, like single/double coffee, you have to tap on the corresponding picture.

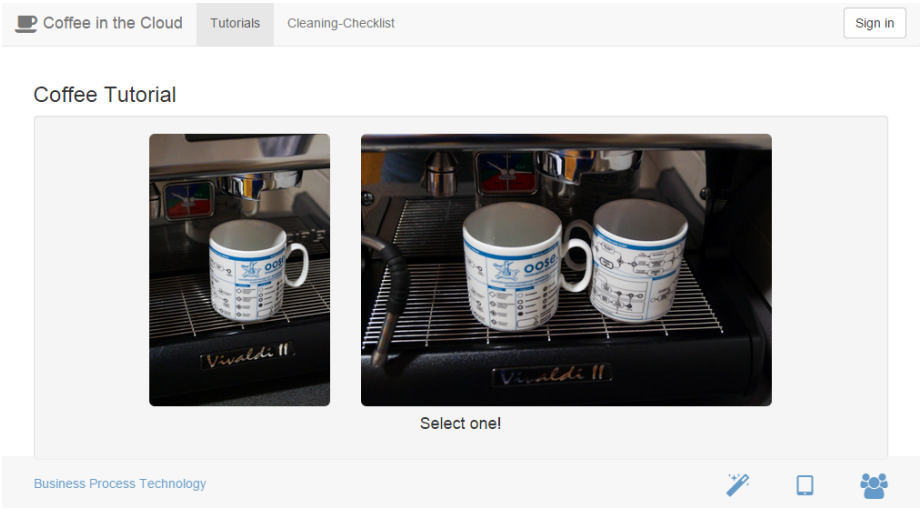


Figure 9: picture tutorial

Cleaning Checklist

If you are assigned for cleaning you can use the cleaning checklist to keep track of the necessary steps. By tapping on a step you can mark them as done. On the bottom you will find a button for resetting the checklist and marking the cleaning as done.

Login

When accessing the application through your browser you will see a welcoming screen like this.

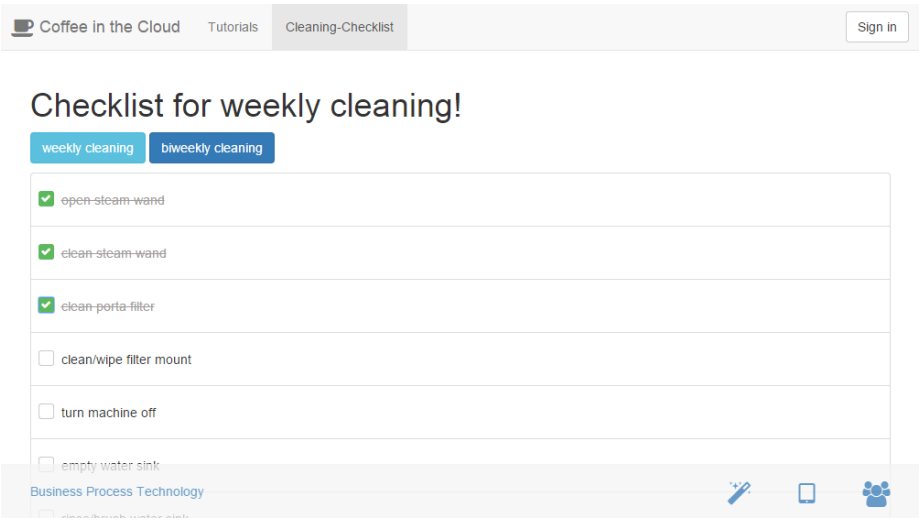


Figure 10: cleaning checklist

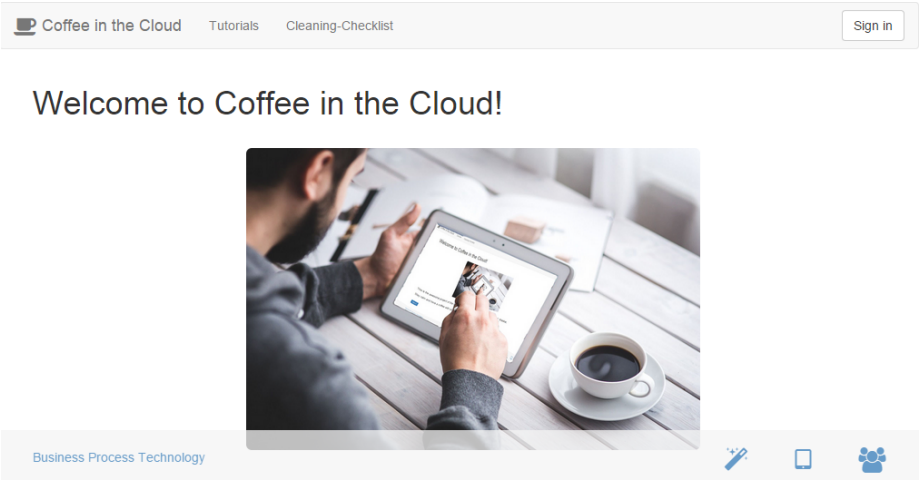


Figure 11: welcome screen

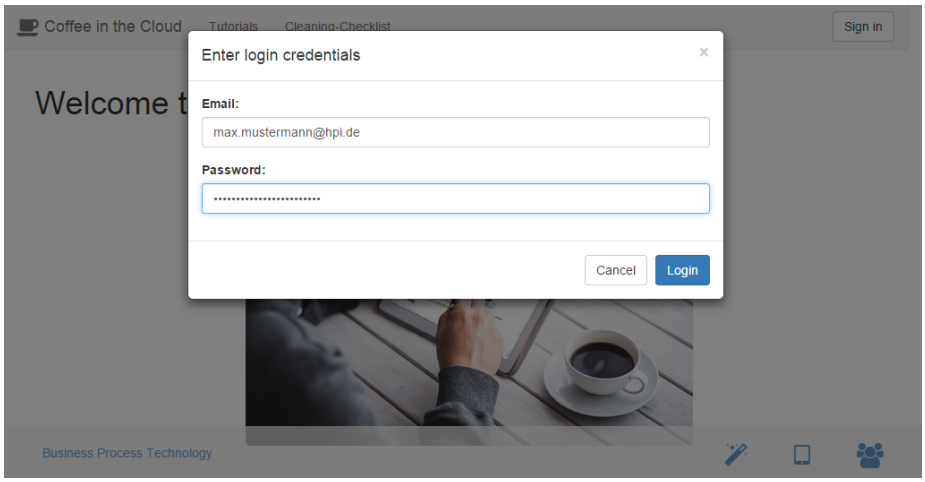


Figure 12: login screen

In the top right corner you can sign in using your email and password. These credentials should be provided by a system administrator.

After successfully logging in you will be greeted personally and can see an overview of your account like the amount of coffees drunk or your current balance.

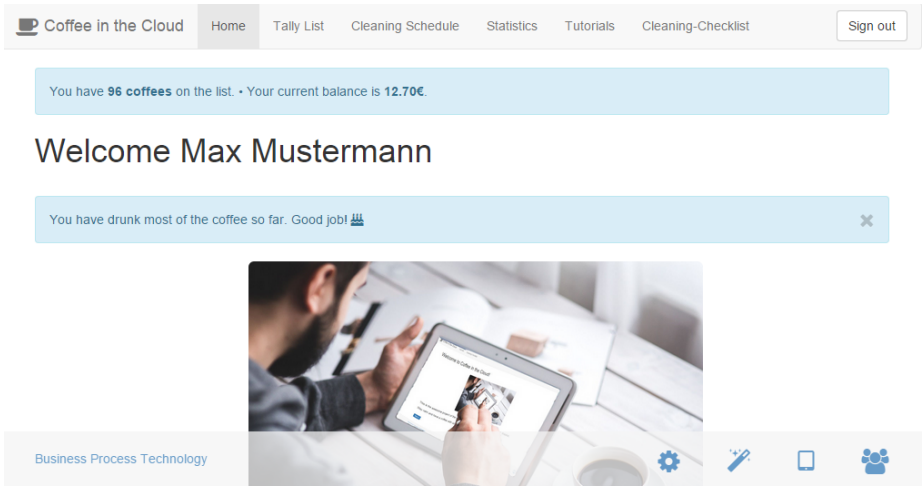


Figure 13: welcome screen

You can now access more modules than before.

Tally List

The tally list screen gives you an overview of the last ten coffees that have been booked to your account. Apart from that you will see your current balance.

In case you wrongfully booked a coffee you can remove it by accessing this page within half an hour.

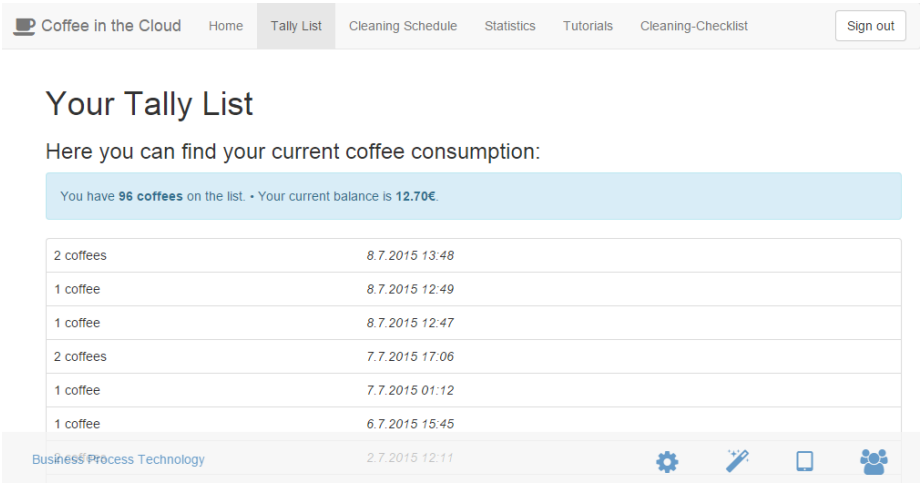


Figure 14: tally list screen

Cleaning Schedule

The cleaning schedule shows who is assigned for cleaning the kitchen in the next weeks. It differentiates between weekly, biweekly and other cleaning. If a cleaning has successfully been done it will be crossed out.

If you are assigned for cleaning you will also get a notification and can mark it as done.

Statistics

In case you wonder about how your own coffee consumption changed over time or how it compares to everyone else you can do so using this page.

Settings

In the bottom line you will find a gear symbol under which you have the option to change your account settings.

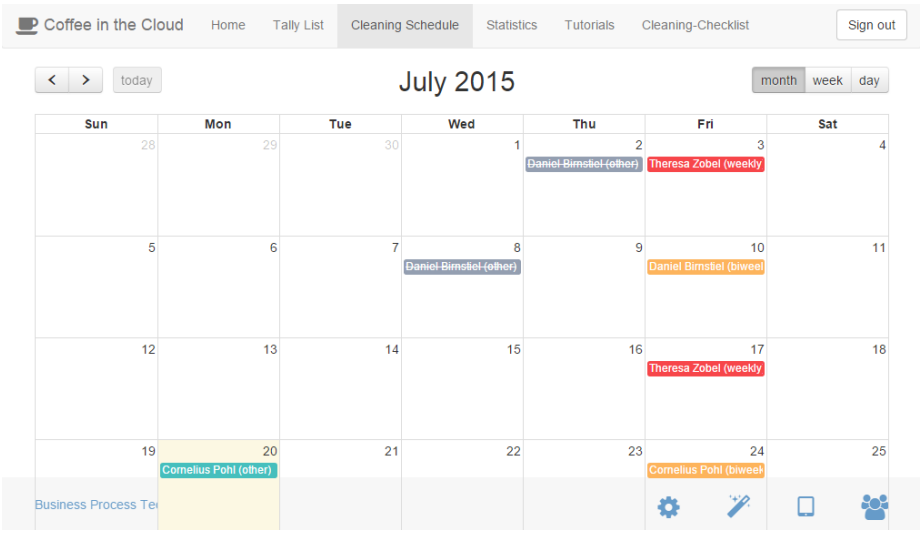


Figure 15: cleaning schedule

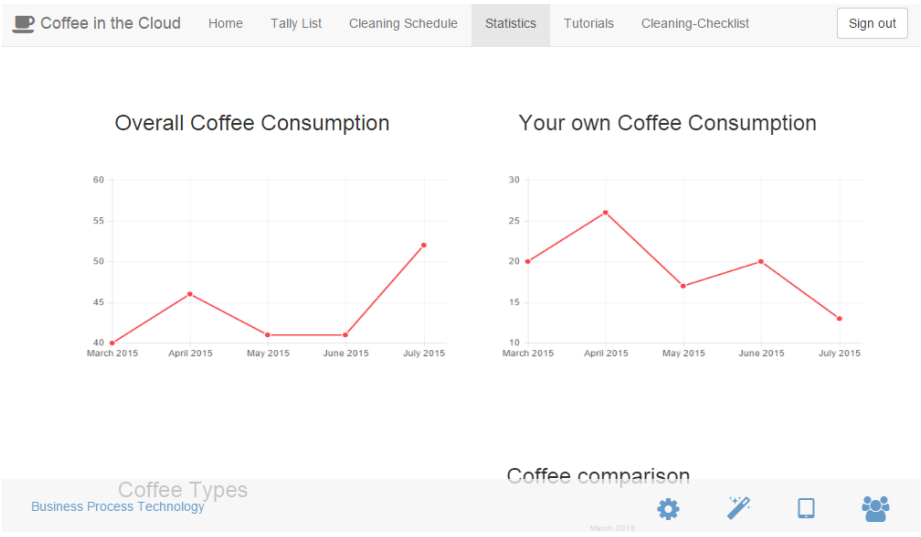


Figure 16: statistics screen

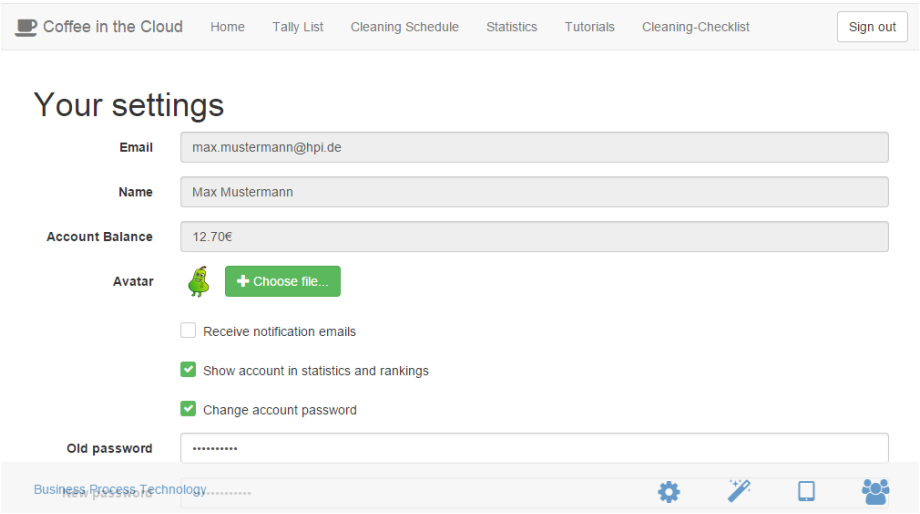


Figure 17: settings screen

In case you have the permission to manage the global balance you can do that here as well.

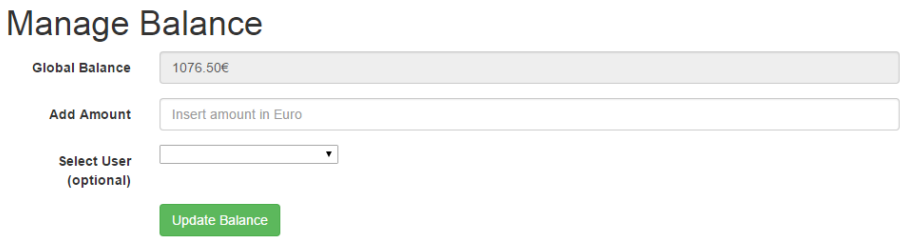


Figure 18: global balance settings

Conclusion

The main goal was to implement a webapplication in order to support the BPT chair. Important features were to have tallylist and and payment balance aswell as the posibility not only to log in by the useres credentials but also with picture login. That's because the chair wanted to remove their old-fashioned tally list in the kitchen with a tablet where the coffee drinkers could login by simply clicking on their face and add the coffee. Although we could not achieve all the aspects, like payment options or autmatic ordering, we are proud of the project. During the process of implementation we were supported optimally by our Seminar

tutors Marcin, Adriatik and Rami. All in all one could say it was a fascinating project with a great team.