Project DVA231 - Web Application Development - Project Group 3

ERNEST POKROPEK, Mälardalen University, Sweden
FRANCESCO MOSCHELLA, Mälardalen University, Sweden
LUKAS SCHNITTCHER, Mälardalen University, Sweden
NANCY ACHIENG OGUTU, Mälardalen University, Sweden

This document contains all important information about our project in the DVA231 Web Development course. We will give you an introduction to our application. Then, we will explain how we approached and planned our project in terms of time schedule and work distribution. Next, we will show all the important front-end functions in detail. Subsequently, all the back-end APIs will be exhaustively explained. Lastly, we will give a summary of the experience that we had in this project. In the end, we will show the list of technologies which we used with the links to their relative documentation.

1 INTRODUCTION

Our project is a client-server web application that refers to the search, creation and evaluation of cocktails.
There are many level of access.
The **Visitor users** can search for cocktails on the search page by writing some ingredients in the search bar, for example gin, or just ask for a random cocktail. The search results will include picture, name, ingredient list and a detailed recipe to make the cocktail. The Normal users valuations for the chosen cocktail are also displayed with rating and comments.
The **Normal users** can add their own cocktails, rate existing cocktails and bookmark their favorite ones for later use.
The **Moderators** can ban users, remove reviews and normal user's cocktails.
Our application uses an already existing RESTful service which provides cocktail data (2) to display results in order not to wait for cocktails created by registered users and be able to offer our service immediately.

2 WORKFLOW

Our first step in the project was to find an interesting and useful idea for a web application . We discussed and brainstormed for a while and then agreed that we wanted to build something that could provide to the user interesting information with a clear interface. It was also important for us to create something where the users can exchange knowledge each other about an interesting topic.
While we were looking for some cool and new RESTful API, we found the Cocktail REST API (2), and so we decided to build our own cocktail website.
Afterwards we decided about all the features that our application needed to have.
We started with the idea that the user wants to make a cocktail with some ingredients that has at home and wants to use or has not enough time to buy other ingredients.
Then the main purpose of the web application is to find all the cocktails that can be done with the ingredients that the user has.
Then we thought about a random cocktail function, that can be used if the user has time and wants to surprise his friends with a fancy cocktail.
Furthermore we decided to add the possibility for the users to create their own cocktails, in order to expand the database and to be able to show a much bigger collection of recipes.

After we have decided what we want to build, we had to split the work and decide how we want to work. We are a group of 4 people, therefore we separated the work so that 2 of us can work on the back-end to provide all functionalities and the others on the front-end to create a clear and minimal interface, which can put the user at ease.
Dividing the work was a good choice, also because this way we could work at both front-end and back-end at the same time.
After we have separated the work, we decided to connect the two sides using a RESTful system.
Finally we decided to use the Django Framework 3.1 (3) as a base for the project, because of the continuous and increasing interest that the community is having about it. Then we found the REST framework provided by Django itself (4) and decided to use the version 3.12, which is the latest one.

We separately worked on both sides (Front-end and Back-end).
After the main functionalities were ready, we started combining them.
The Back-end was completed before the Front-end. Because of that, after the main features were ready, we started to test and improve the application while adding new features.

A summary of our time schedule can be seen on the picture below.

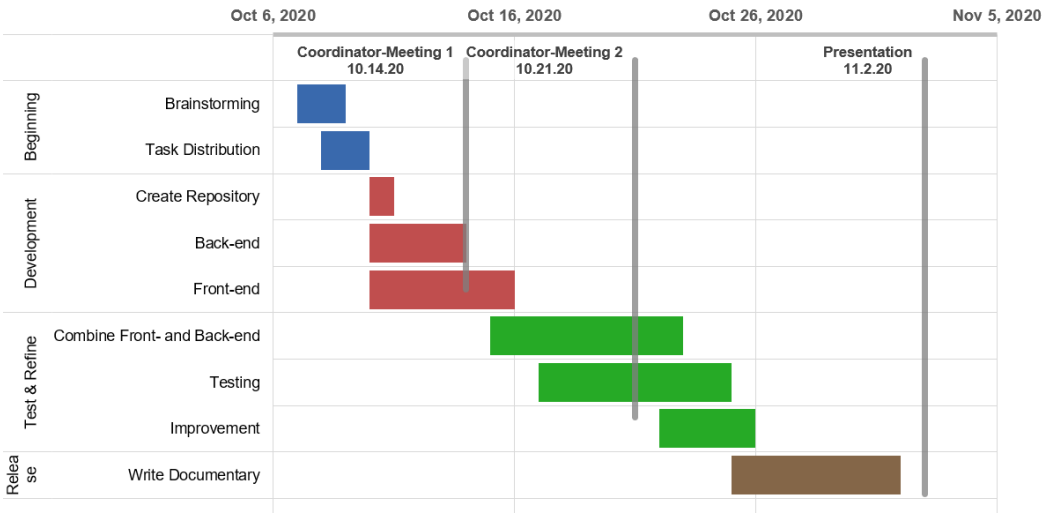

Fig. 1. Time-Schedule for Project

3 USER INTERFACE

To offer a flawless user experience for our application we integrated the following technologies:

- HTML5
- Bootstrap 4.5 (6), a CSS3 framework. From it we used the grid system, which allowed us to align items easier inside the pages, and the modals as a base for ours
- JQuery 3.5 (7), a JavaScript library that allowed us to write JavaScript code easier
- Tags Input 0.8 (8), a JavaScript Plugin, that helped us in the search
- Rater JS (9), a JavaScript Widget, that was used for the review system
- Font Awesome (10) for the icons

The User Interface is composed of two pages: the home and the user profile.
All communications between the front-end and the back-end are done using Ajax and by calling a specific RESTful API with the appropriate method and the correct parameters.

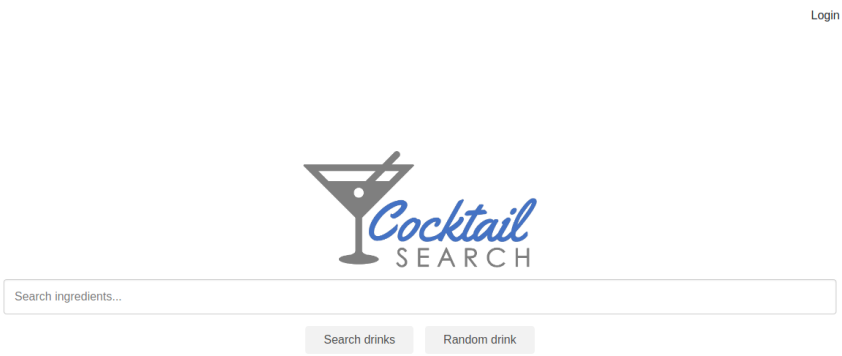

Fig. 2. Home page of the website

This is the appearance of the Web Application Home Page. As it can be noticed, we created a clear and concise Google looking page.
Having a clear and concise page helps the user to focus directly on what really matters and what should be done on the web site. Also making the page similar to Google, allows a correct, easy and direct use. Furthermore, the visitor can add more than one ingredient which will be displayed clearly in the search bar.

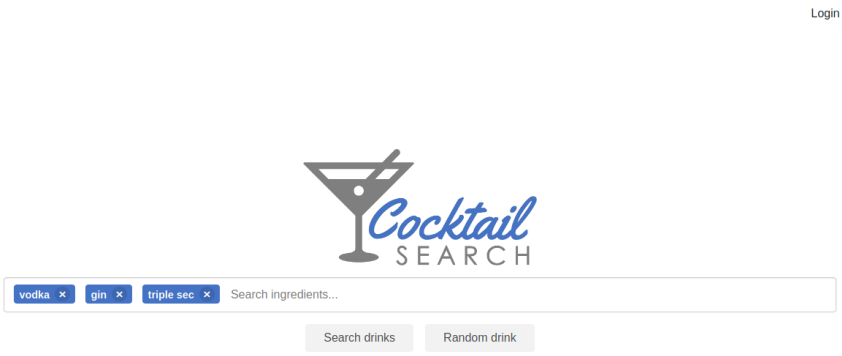

Fig. 3. Home page of the website with tags in the search bar

After the user starts inserting ingredients in the search bar, will be noticed that some tags appear and replace the inserted ingredients. This is accomplished by using the Tags Input Plugin (8) with a certain dose of custom styling.
When the user clicks on the search button, these tags are sent to the back-end and the search is started.

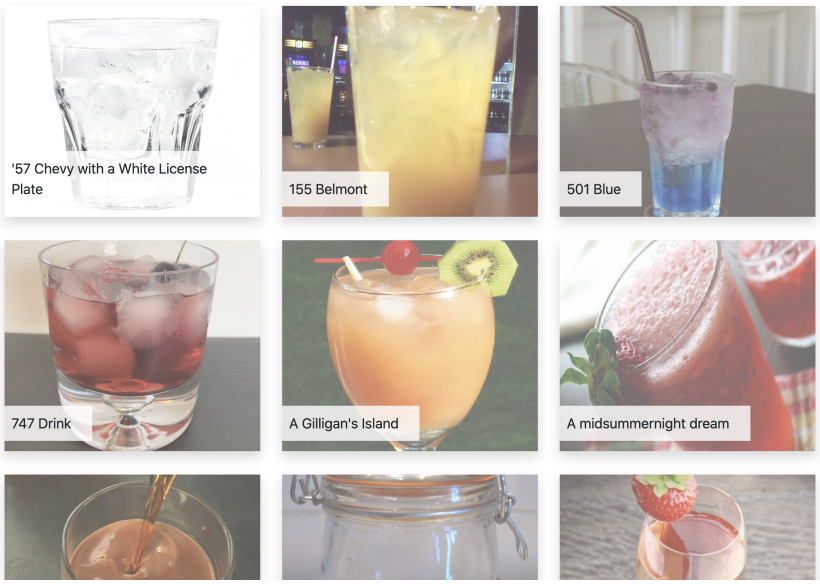

Fig. 4. Search results

When the search is complete, some cocktails will be shown and the page will scroll in a way to hide the search bar and let the user focus on the cocktail list. Only part of the complete list of cocktail is shown. At the end of the list, a show more button is provided. By clicking this button the user will be able to see some cocktails more.

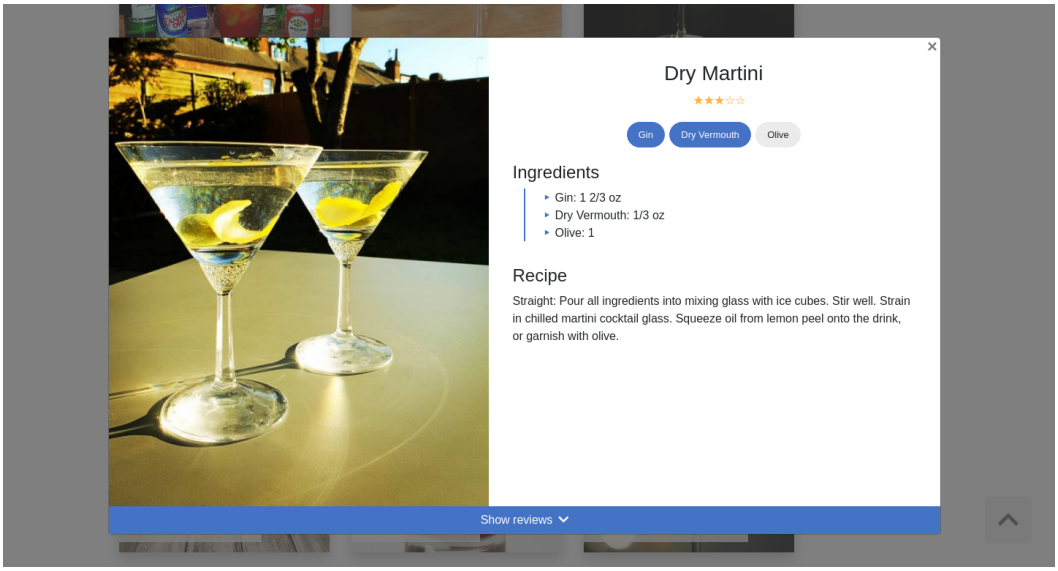

Fig. 5. Modal of the selected drink. Note that the ingredients from the initial query are highlighted in the top list.

After the user has done his choice, is possible to see more details about the cocktail, specially the complete ingredients list, the rating, the recipe and all the reviews that the cocktail has.
To do that the user has to simply click on the desired cocktail and instantaneously a Bootstrap Modal will cover the page and the appropriate information stored in the server will be shown. The main part of the modal regards all the cocktail details. To read the comments is necessary to click the blue bar on the bottom of the modal. After the click, the reviews are retrieved and showed to the user.

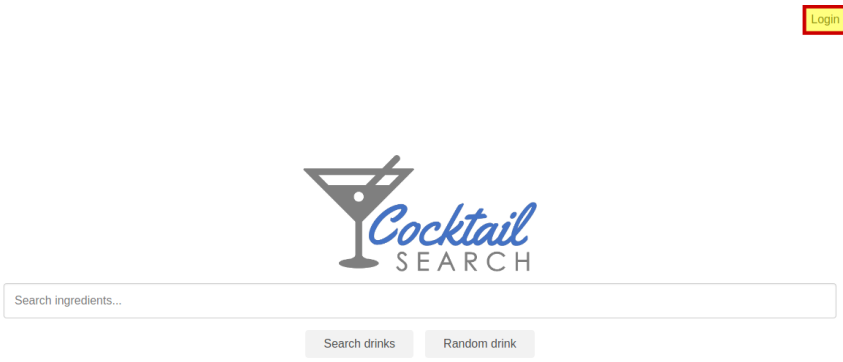

Fig. 6. Registration form displayed in modal

On the top bar, in the right corner, we have 2 buttons: *Login* and *Register*.

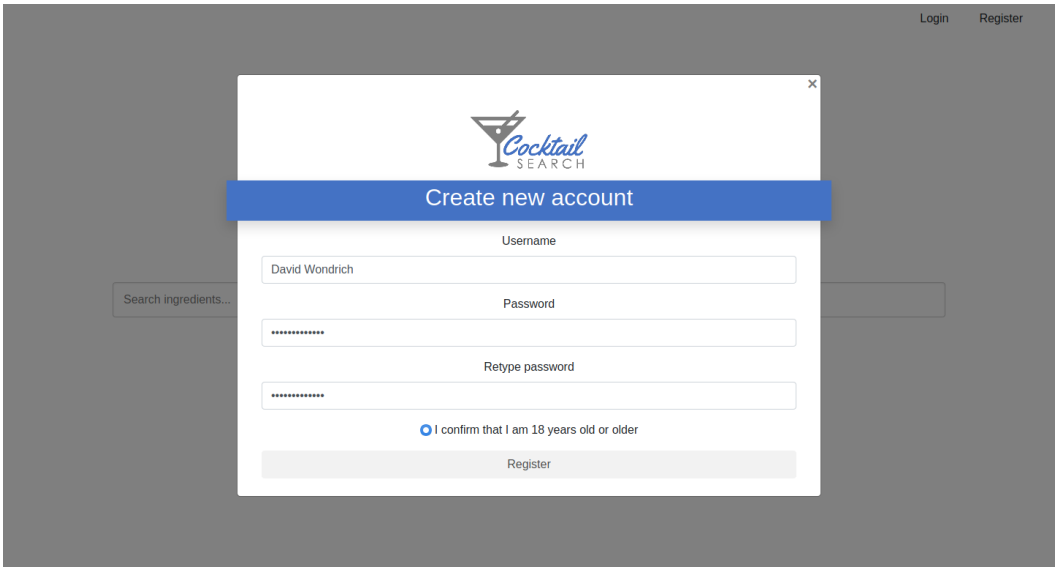

Fig. 7. Registration form displayed in modal

When a user wants to Sign up to our web site, username, password and the confirmation, that is at least 18 years old, are required.
Unfortunately, as all the other websites, we cannot have the complete certainty that the user is not lying about the real age.

After the user submits the filled out form, the data are sent to the back-end, the user gets registered into the database and a confirmation message is shown.

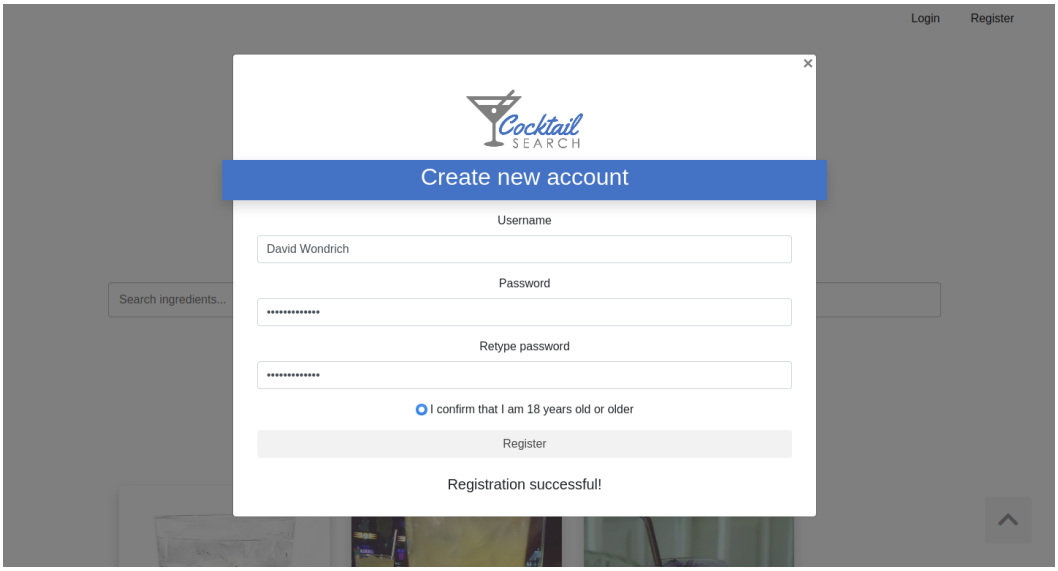

Fig. 8. Registration confirmation displayed in modal

Finally, as it can be seen in the image, the Registration confirmation is shown to the user.

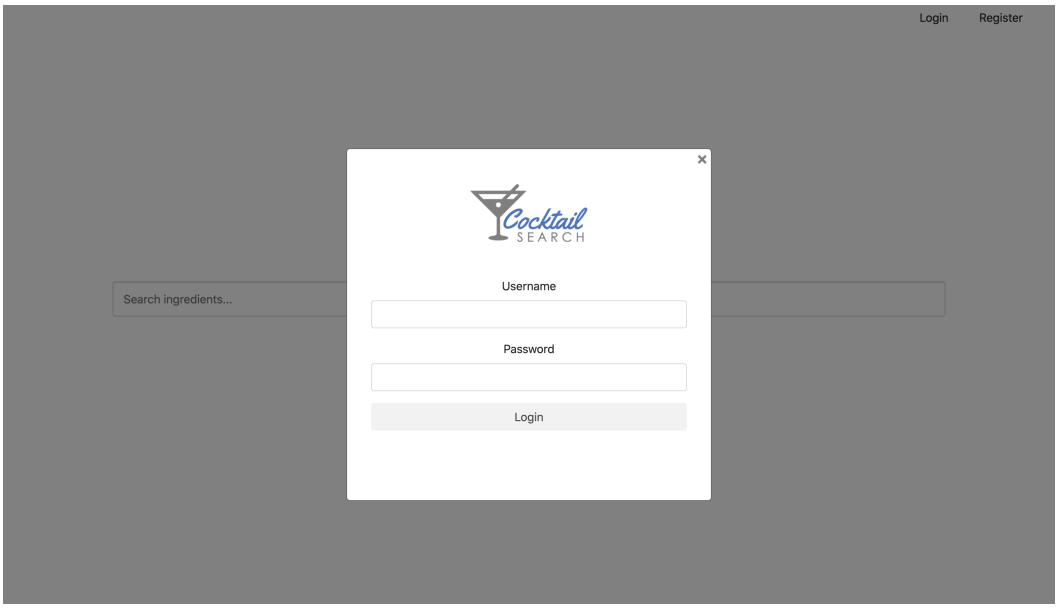

Fig. 9. Login form displayed in modal

After the registration, when a user wants to login, the login form has to be filled and sent. When the user sends the form, the data are sent to the server and the user instantaneously gets logged in.

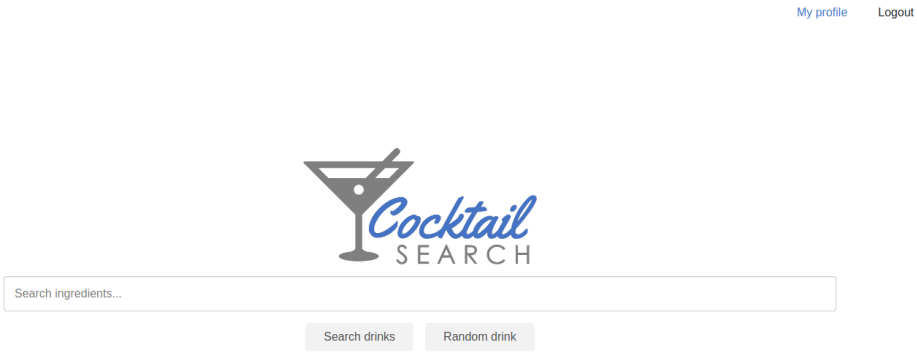

Fig. 10. Home Page after a successfully login

After the login, the user is not redirected to his private profile page, instead a slightly different home page is shown. This happens because the main functionality of the web application is to search, review and bookmark cocktails. After the login, the user has some new rights.

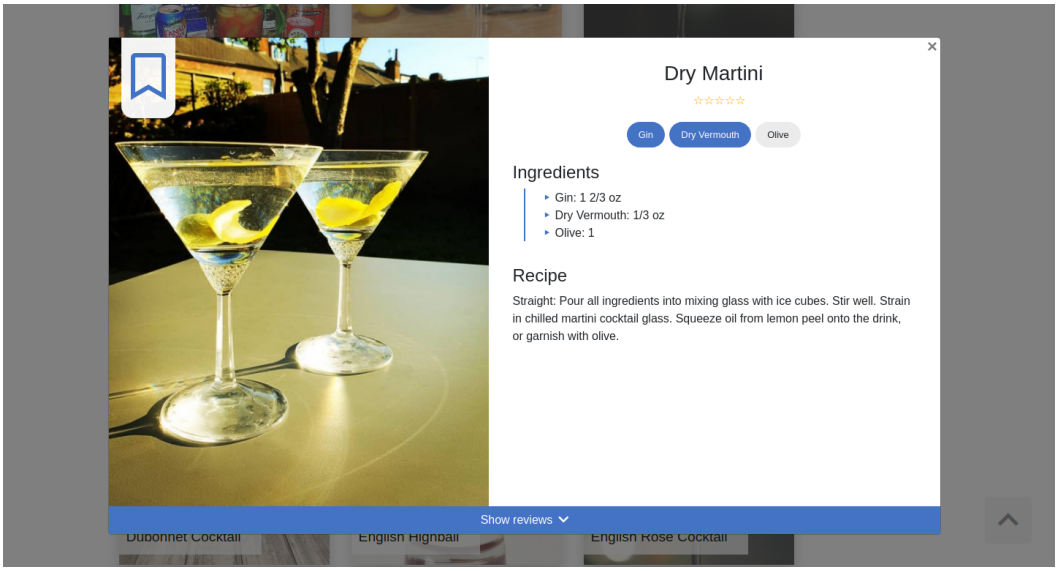

Fig. 11. Cocktail Modal with bookmark right

The new rights consist in the bookmark ability. Logged in users in fact can select some cocktails and save them, then their favourite cocktails will be shown on their profile. When a user bookmarks or unbookmarks a cocktail, an appropriate request to the API is done.

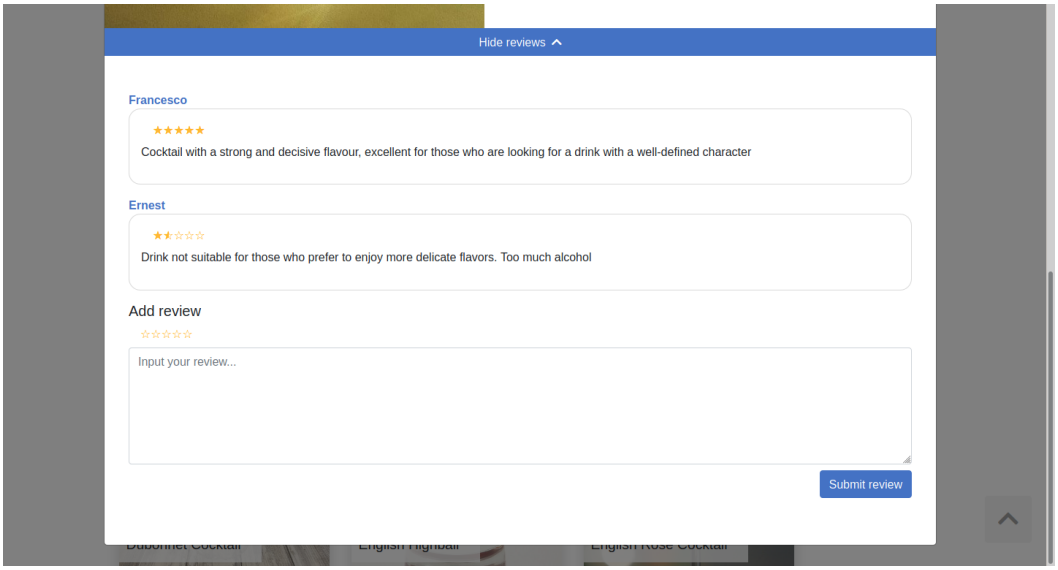

Fig. 12. Cocktail Modal, Review section with add review right

The other right is the possibility of adding, editing and deleting their own Reviews. The reviews consist of a vote, from 0 to 5 stars, and a comment, up to 500 characters. To avoid spam voting, a user can make only one review under a cocktail. To do that there is an appropriate JavaScript function helped by one in the back-end.

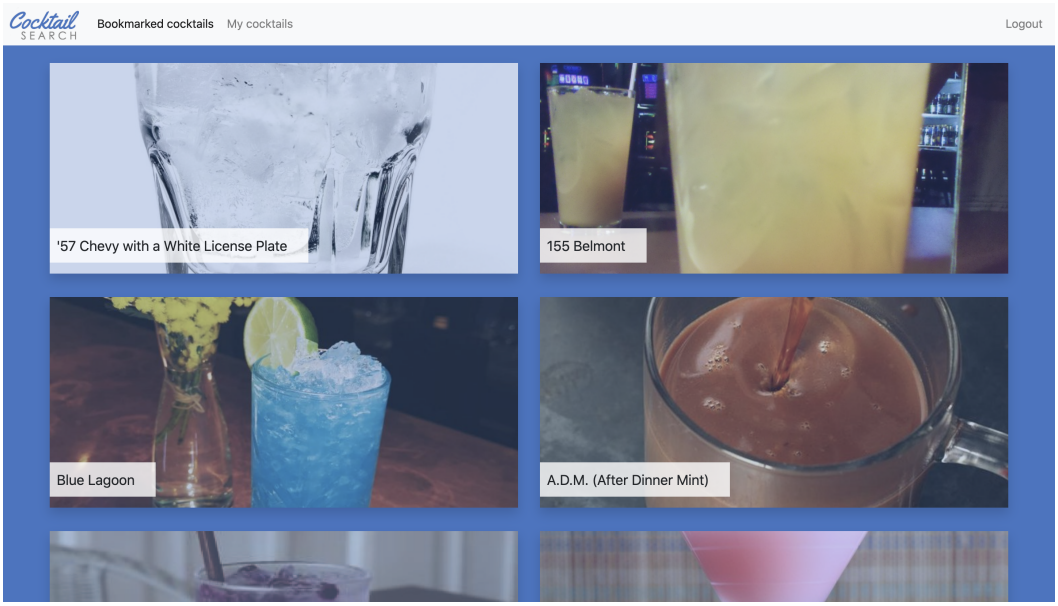

Fig. 13. Home page of the profile, with bookmarked cocktails displayed

When the user enters the profile page, something like this will be shown. In order to let the user see all the necessary data, some requests are performed and all the necessary elements get retrieved and putted in the page.

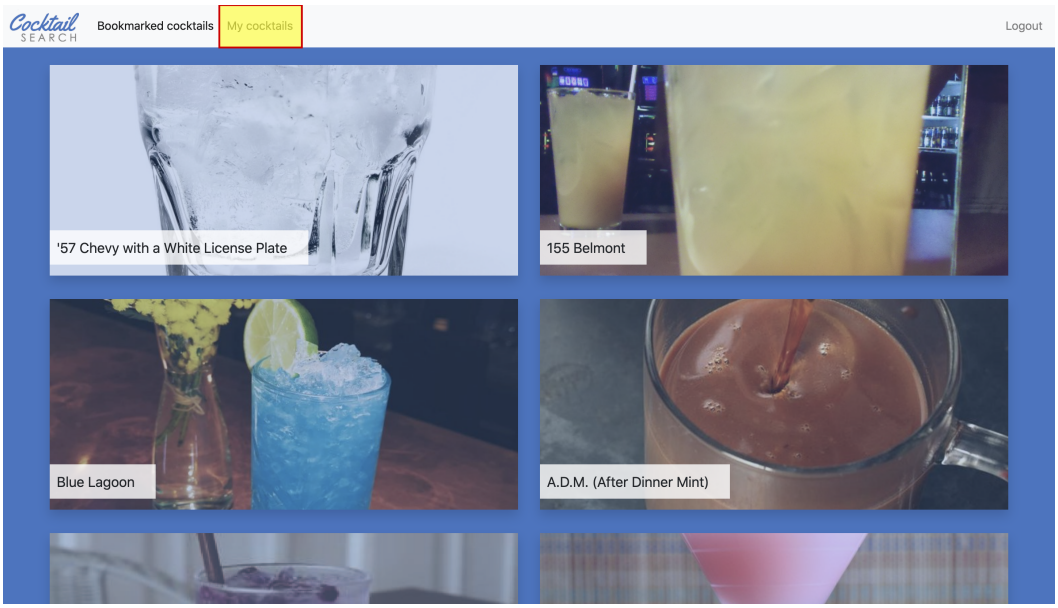

Fig. 14. Home page of the profile

From the personal page, the user is allowed to manage (create, edit, remove) the personal cocktails.

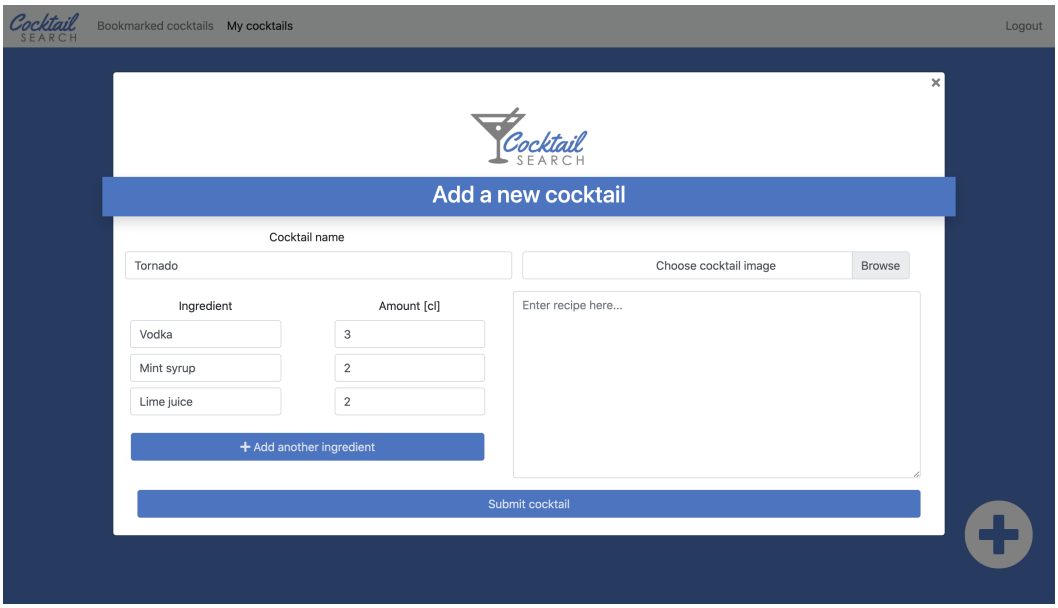

Fig. 15. Form for adding a new cocktail in the profile section

When a user wants to add a cocktail, a form has to be filled out and sent. After the server response a confirmation or error message is shown to the user.

When a user wants to edit a cocktail that has previously created, a slightly different form has to be filled out and sent. Then, accordingly to the server response, the correct message is shown to the user.

A user also may want to delete a cocktail. In this case, the delete function can be used. After the delete confirmation, the cocktail gets deleted and a message is shown to the user.

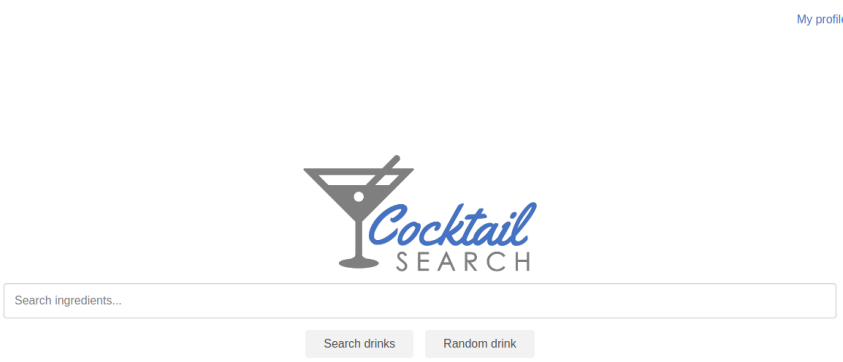

Fig. 16. Home page logout button

When a user wants to logout, a special button is provided in the top right angle of the page. After the user clicks, it will be immediately redirected and logged out.

If the user that logs in is a *Moderator*, the pages will look similar to the normal user ones. The only difference is that the moderator will have access to these features:

- removing other users comments (from the review section of a cocktail modal)
- banning or unbanning cocktail (from the cocktail modal)
- banning users (from his personal page, where there is a special section to search users and ban them)

4 FUNCTIONALITY

The Cocktail REST API (2) allows us to use cocktails which are already available and publicly accessible. This way, we can provide thousands of cocktails, even without the users personal cocktails. We access these cocktails by sending *GET* requests to the REST API and making use of the different search functions, like searching for cocktails which contain specific ingredients or even a random cocktail. The Response is a JSON Object which contains all the necessary information.
The functions from the REST API that we use are *filter* with parameter *i* for searching by ingredients, the *lookup* with parameter *i* to get all information given for a cocktail and the *random* to get a random cocktail.

We use the Django Framework 3.1 (3) for all necessary components of our web application.
We use SQLite3 (5) as the default RDBMS which Django provides.
To create the database tables, we make use of the Object-Relational Mapper (ORM), also included in Django, because of that we do not have to write SQL code.
The tables can be described as classes in python and are later translated by the framework to the equivalent SQL code.

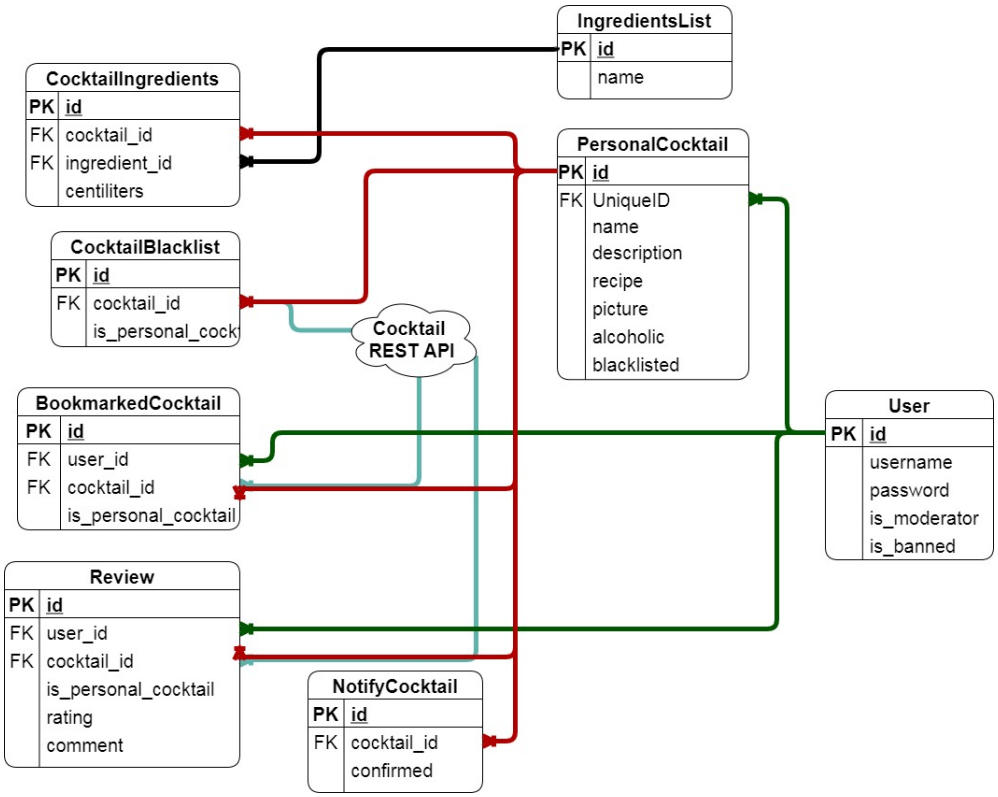

Fig. 17. ER Diagram of The Web Application database

Our database is composed of 8 Django models.

Cocktail Rest API. Even if the Cocktail REST API is not a real part of the Web Application database, we decided to add it in the ER model in Figure 17, in order to show that also our database works in contact with this API by containing some cocktail identifiers in combination with a flag (later also referred to as "database flag") that let us understand the origin of that identifier.

User. This Model (in the next paragraphs described as "table") contains all the user data.
We decided to collect as less data as possible, in order to follow the *privacy by design* rules and prevent the necessity of creating a privacy policy.
The user password is hashed with SHA256. This way passwords are not in clear-text and all of them have the same length, which is 64 hexadecimal characters.

Personal Cocktail. This table contains all the data that are necessary for a reasonable representation of cocktail. As is possible to see from the Figure 17, every personal cocktail must belong only to one user.

Ingredients List. This table contains all the ingredients that can belong to a cocktail. To save space we decided to create a many-to-many relation to the personal-cocktail table.

Cocktail Ingredients. This table is used for the many-to-many relation between personal-cocktails and ingredients. The only field that is contained, except for the foreign keys, is the centiliters which define how much of the ingredient is required for a specific cocktail.

Cocktail Blacklist. This table contains all cocktails which have been blacklisted by a moderator and is used to remove some cocktails that might violate the netiquette.

Bookmarked Cocktail. This table is used for the many-to many relation between users and cocktails, in order to allow users to save their favourite cocktails and have them in a separate list.

Review. This table contains all reviews that users can make for a specific cocktail. It contains a rating, floating point number value, from 0 to 5, and a comment, string of maximum 500 characters.

Notify Cocktail. This table is used to save all the notifications that users receive when one of their personal cocktail is blacklisted.

In order to be able to work remotely and separated, we decided to use a RESTful system. We made some functions that could be called by the front-end through requests with *HEAD*, *GET*, *POST*, *PATCH*, *PUT* or *DELETE* request methods. Each function responded with an HTTP STATUS CODE (200, 400, 401, 404 are the most likely to be returned) and, if needed, a JSON-Object containing the desired information.
The REST API functions in our web application are:

User. This function can be called with *GET*, *HEAD*, *POST*, *PUT* and *DELETE* methods.
Only the moderators are allowed to use the *GET* method. With this method they can get the complete user list that contains the unique user identifier and the nickname.
The *POST* method is used to perform a login.
It requires the parameters username and password. After the check, if data are correct, the session is created, with the user unique identifier and the two flags to check if the user is logged and if is a moderator.
The *HEAD* method is used to logout. It works only if user is logged in and, when called, destroys the session.
The *PUT* method is used to register a user. The required parameters are username and password. There is also another parameter, which is not necessary, that is a flag to set the new user as a moderator.
If the user wants to destroy his own account with all of his/hers comments and cocktails, the *DELETE* method can be used. By passing it username and password, the user account will be destroyed.
If a moderator wants to ban/unban an user, the *DELETE* method has to be utilized. In this case, the user unique identifier is required; after this call, the ban flag of a user is toggled.

Bookmark. This function is used to save the favourite cocktails of a user.
It can be called with a *GET*, *POST* or *DELETE* request.
The *GET* request is used to get the complete list of bookmarked cocktails that a user has.
The *POST* request is used to add a bookmarked cocktail and requires two parameters: the cocktail unique identifier and the database flag.
The *DELETE* request is used to remove a cocktail from the bookmarks list. It requires the unique identifier of the cocktail and the database flag.

Cocktail. This is the main function of the Web Application.
It is called to

(1) search cocktails from the home page bar
(2) obtain all the necessary data when a user clicks on a cocktail cell
(3) get a random cocktail

(4) get the list of personal cocktails that belongs to a user
(5) let a user create a new personal cocktail
(6) allow a user to edit a personal cocktail
(7) remove or blacklist a cocktail

Only the first 3 of the above functions can be used without the login; all of the others require the user to be logged in.
This API can be called with either *GET*, *POST*, *PATCH* and *DELETE* methods.
When a *GET* request is performed, based upon the *GET* parameters, one between the functions 1, 2, 3 or 4 is done.

Function 1. Is used when, after a user searches a cocktail from the home page, a list of ingredients is sent, in form of CSV, as a parameter to the *GET* request.
The function searches for the specific ingredients in the database and also in the Cocktail REST API (2) and returns a list with narrow information (Name, Image-URL, Rating) that are enough for showing the smaller results on the page.

Function 2. Is called when, after the user chooses one cocktail out of the search results, the cocktail identifier is sent as a *GET* parameter with the database flag.
The function searches and returns the full information (Description, Ingredients and Measurements, Recipe, Name, Picture, Reviews, Rating) of the specified cocktail.

Function 3. Is utilized when the random flag is passed as a parameter to the *GET* request.
This function, using a pseudo-random choice, selects and returns all the Cocktail information (Description, Ingredients and Measurements, Recipe, Name, Picture, Reviews, Rating) from the Database or the Cocktail REST API.

Function 4. Is called when no parameters are passed to the *GET* request.
If the User is a moderator, this function returns a list with the minimum data (Name, Image-URL, Rating) about all the personal cocktails that are in the database.
Otherwise, it returns a list with the minimum data about all the personal cocktails that a user has created.

Function 5. Is called when a *POST* request is received.
It is used to let a user create a new personal cocktail.
It requires name, description, a list containing all the ingredients, recipe and image of the cocktail.
To avoid naming conflicts, all images are saved inside of the user personal folder with a new name, which consists in the concatenation of date and time.

Function 6. Responds to *PATCH* requests.
Allows a user to edit a previously created cocktail.
It requires the cocktail unique identifier and all the fields that have to be edited (name, description, a list containing all the ingredients, recipe and/or image of the cocktail).

Function 7. Is used when a *DELETE* request is performed.
It permits the removal of a cocktail by the cocktail owner and allows a moderator to blacklist/unblacklist a cocktail.
In both cases (user and moderator) the function requires the cocktail unique identifier, then if the user owns the specified cocktail, the cocktail is removed with all of its reviews and it will never appear in the bookmarked cocktails anymore.
If the user is a moderator and the cocktail was not previously in the blacklist, it will be blacklisted

and a notification will be shown to the owner on the next login.
If the user is a moderator and the cocktail was previously in the blacklist, the cocktail will be removed from the blacklist and the notification will be removed.

Notifications. When a moderator blacklists a cocktail, the user has to be notified, to be aware and to be able to complain.
This function can be called with two methods: *GET* and *POST*.
The *GET* request is used to obtain the list of the blacklisted cocktails that the user has to confirm yet; the list contains the cocktail identifier and the notification identifier.
The *POST* request is used to confirm that the user has seen the notification, in order not to show again the notification that has been checked.
It requires an array containing all the notification identifiers.

Review. The users are also able to interact with the web application by adding some reviews to their favourite cocktails.
This API can be called with a *GET*, *POST*, *PATCH* or *DELETE* request.
The *GET* request method, the only request method that can be used also when the user is not logged in, is used to obtain all the reviews that a cocktail has. It requires two parameters: the cocktail identifier and the flag that specifies if the cocktail was created by a user, or if it comes from the Cocktail API (2).
This API returns a JSON containing all the reviews for the specified cocktail.
The *POST* method allows the user to add a review for a cocktail.
This method requires the cocktail identifier, the database flag, the rating and the comment that the user gave to the cocktail.
The *PATCH* method is used to edit a review that a user has done before.
It requires the review unique identifier and rating and/or comment.
The *DELETE* method allows the user and the moderator to remove the Review. The user can only remove his own reviews; the moderator can remove both: his own review and other users reviews.
The parameter required by this method is the Review unique identifier.

5 CONCLUSION

No one of us has worked with Django before, but we all wanted to learn it. Because of that we started by reading the documentation and learning how to use it.
After these 3 weeks of work, we all achieved a good knowledge of the Django Framework which allowed us to work on project.
Before starting the project, we needed a couple of days to understand the basics of this framework, after we have got that knowledge, we created the project and worked on it.

We managed successfully to collaborate as a group, by dividing the work, making some weekly meetings, helping each other to understand what resulted complicated or hard to understand.

Everyone contributed to the project with an equivalent amount of task to complete.
Francesco M. and Lukas S. were in charge of the back-end, while Ernest P. and Nancy A. O. had to create the front-end, making sure to grant a perfect user experience.
More specifically, Francesco M. was in charge of the user related functionalities, including the personal cocktail; Lukas S. was in charge of the cocktail search components and the correct use of the results from the Cocktail REST API (2); Ernest P. had to do the home, login and cocktail pages with related JavaScript scripts; Nancy A. O. had to do the User profile page with related JavaScript Scripts.

Further explanations about the work separations are in the README.md file of the Github Repository (1)

Unfortunately, because of the time constraints, we were not able to implement all the provided functionalities. Therefore we want to implement in the future the following list of features:

- Add an 18+ years old confirmation when the user enters in the website
- Show firstly the cocktail that matches more ingredients
- Public the website online
- Add an SSL certificate in order to use the HTTPS protocol, instead of the HTTP one
- Encrypt the login and registration data before sending them to the back-end
- Add constraints for the password length and strength
- Remove banned users Reviews
- Adding a Salt for safe encryption of the user password
- Search for non-alcoholic or alcoholic Cocktails
- Create privacy/cookie policy for the website
- Add email, birth date to the user registration
- Add profile picture and friend-list feature
- Make Public Profile Pages
- Use OAuth 2 for login
- Edit some API to lower the possibility of XSS
- Use a CDN and save images on the cloud in order to lower the space needed by the website
- Add the Admin user with the ability to give and remove moderator rights to users

6 REFERENCES

(1) Github Project
(2) Cocktail REST API
(3) Django Framework
(4) Django REST Framework
(5) SQLite3
(6) Bootstrap Framework
(7) jQuery Library
(8) Tags Input Plugin
(9) Rater JS Widget
(10) Font Awesome