ApartRent

DDWT Final Project Report

Project members

Hylke van der Veen - s3348431 - H.F.van.der.Veen@student.rug.nl
Folkert Leistra - s3360695 - F.A.Leistra@student.rug.nl
Thijmen Dam - s3360695 - T.M.Dam@student.rug.nl

Abstract

In this project report we address ApartRent, an application that tries to tackle the pressing issue regarding the lack of student housing in Groningen. We tackle this issue by creating a CRUD application where property owners can easily list their rooms which can be applied for by tenants.

1 Introduction

The increase of students coming to Groningen brings up difficulties regarding housing. The issues are getting so pressing that multiple foreign students have had to sleep in tents due to lack of available rooms and the difficulty of finding one. It should be easier for room owners and tenants to connect, which is why we have developed ApartRent, a platform where tenants can find the room that fits their needs and gets them in touch with its owner.

Owners can list their rooms and specify the information regarding that room. Tenants can respond to an available room by applying and leaving a message. Room applications are visible at a glance and allows owners to check the potential tenants profile and contact them. This allows for a quick letting process, which reduces the time between applying for and getting a room, while owners do not have to miss a single month of rent.

2 Data

Creating an account is necessary to make full use of ApartRent. On register, users have to specifcy their role, either property owner or tenant, in addition to their general account information. Our application functions differently based on their selected role. Both property owners and tenants have to enter their personalia (for identification purposes), account details (for logging in) and information about themselves (viewable by other users of the platform). The id of the user is automatically generated.

Property owners can add, update and remove rooms. Room information consists of one or more images, address details, size, price, type and description. Tenants can opt-in on available rooms. When applying for a room, tenants have to add a message. The details of the room application will then be visible for the owner of the specific room. Data of each individual available room is visible by all users, even if not logged in, and can be edited by the owner of the room. Property owners can look at a potential tenants profile to see if they may be a suitable candidate.

After comments on the ER-diagram of the project proposal, we changed the owner and tenant entity to a single user entity with an ISA relation (see figure 1). In the database this is expressed through a user table containing all personal information and a unique user ID, which is linked to either the owners or tenants table. The room table contains all data about a room and its unique room ID. The opt-in relation is expressed as a table containing a room ID, a tenant ID and a message. The own relation contains a room ID and an owner ID. Because each room has a many-to-one cardinality with owner, we decided to implement owner as a column in the room table.

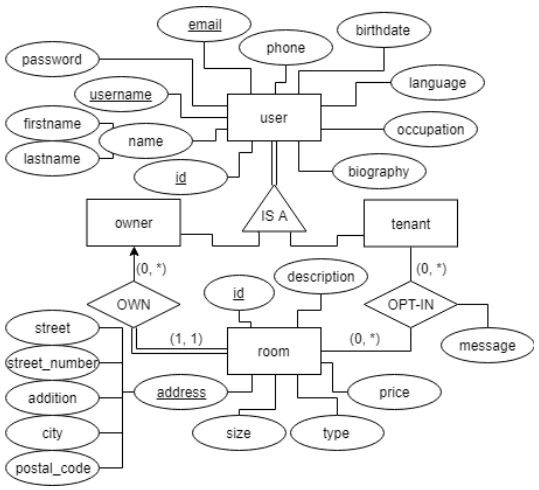


Figure 1: ER-diagram of the ApartRent database structure.

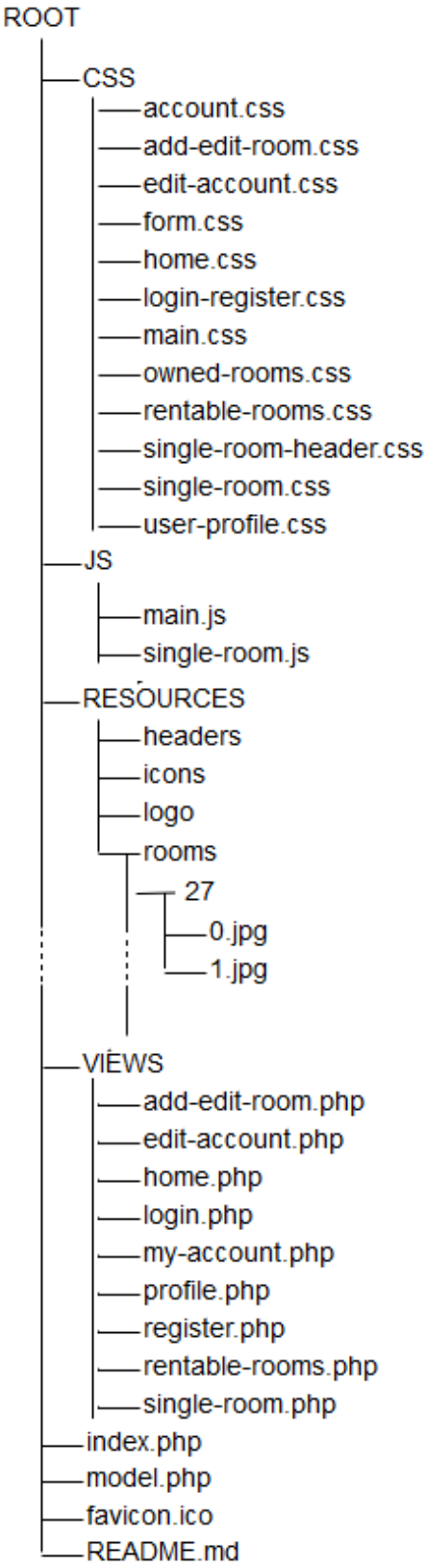
3 Method

This section describes how we handled the development of ApartRent. This is done by addressing the structure of the application, key features, usability and maintainability. The end of this section addresses the division of labor.

Available pages

- Landing (all users)
- Login (not logged in users)
- Register (not logged in users)
- My Account (tenant/owner)
- Edit Account (tenant/owner)
- Tenant personal pages (owner)
- Individual room (all users)
- Room for rent (all users)
- Owned rooms (owner)
- Add room (owner)
- Edit room (owner)

File tree



Structure

Our application defines as a CRUD application, which means that our application supports create, read, update and delete functionality. This is

reflected by our application as described in the following sections.

Create
Users that visit our application have the possibility to register an account, registering as either a property owner or a tenant. Users that registered as a property owner have the possibility to list their available rooms.

Read
There is plenty of information displayed on our application's website. Everyone that visits the website can view the currently available rooms - having an account is not mandatory for this functionality. Users can view their account details on the my account page. Users that have registered as tenant can also view the rooms that they have applied for on the my account page. Users that have registered as owner can view the rooms they own and possible opt-in messages on their account page. Owners can also view the profile of tenants that have applied for one their rooms.

Update
Users of our application that created an account can update their account information. Users that registered as owner also have the possibility to edit the information of the rooms that they listed. Users of our application that registered as tenant have the possibility to apply for a room by clicking the opt-in button and leaving a message.

Delete
Users have the possibility to delete their account, whether tenant or owner. Users registered as owner also have the possibility to remove the room(s) they listed. Users registered as tenant have the possibility to remove their opt-in for a room they applied for.

Model View Controller
Our application has been structured according to the MVC-model. This means that we divided our program into a model, seperate views and a controller. This structure improves the readability, usability and maintainability of the project. Also, separating the project's code in different sections makes for easier designing without interfering with the data model and vice-versa. The sections below explain the different parts of the MVC-model in more detail.

Model
The model consists of functions that retrieve, store and delete data of the database. In addition, several functions generate HTML based on the function's input. This modular approach makes it easy to reuse website elements on multiple pages.

View
In order to be able to represent our data we created different templates and views. Most pages have their own unique view, with the exception of pages that have a similar design. The views are responsible for most the design and user interface elements of the website.

Controller
The main priority of the controller is to specify each route's functionality and control the navigation. Each route specified in the controller is listed below, giving a brief explanation of the route's general purpose.

Route overview

Landing page
Route: `/(GET)`
View: `home.php`
Description: Handles the landing page.

Room overview
Route: `/rentable-rooms/(GET)`
View: `rentable-rooms.php`
Description: Is mainly used to display all the current rooms that are for rent by including our rentable-rooms view.

Single room
Route: `/room/(GET)`
View: `single-room.php`
Description: Displays information about a single room. This is done by retrieving the room ID from the database. This page looks completely different for property owners, tenants and for people that have not logged in. Property owners will see an edit and remove button if and only if they own the room. Tenants wont see these buttons, but will see a box at the bottom of the page where they can apply for the room. If they have already applied for the room, they will see their opt-in message and a

button to remove their opt-in. When a user visits this route without logging in, they will see the information about the room, but can not perform any actions on the route until logging in.

My account
Route: `/my-account/(GET)`
View: `my-account.php`
Description: Provides account information if a user is logged in. Owners will see their listed room(s) and possible opt-ins. If a tenant has applied for a room, the owner can view their profile by clicking on the tenants name. Tenants will see the rooms that they have applied for. Both owners and tenants can edit their account details on this page.

Edit account
Route: `/edit-account(GET)`
View: `edit-account.php`
Description: Is used by account owners to edit their account details. The account that has to be edited is determined by retrieving the user ID from the SESSION variable. Once the edit button is clicked, the route continues on the post route.

Edit account
Route: `/edit-account/(POST)`
View: `edit-account.php`
Description: Is used to validate and push the update account details to the database.

Remove account
Route: `/remove-account/(POST)`
View: `none`
Description: Is used to to remove an account from the database.

Register
Route: `/register/(GET)`
View: `register.php`
Description: Is used to display our register form. Once the user clicks on the register button, the form data is sent to the register POST route.

Register
Route: `/register/(POST)`
View: `none.php`
Description: Is used to validate and push the register information to the database. If this succeeds, the user will be redirected to the my account page and the user information will be added to the user table. If the this fails, the route will redirect to the register page whilst preserving form data that has been filled in.

Login
Route: `/login/(GET)`
View: `login.php`
Description: Is used to display the login page If a user has already logged in, the route will redirect to the my account page. Once the login button is clicked, the route will redirect to the login POST route.

Login
Route: `/login/(POST)`
View: `none`
Description: Is used to log a user in. If this succeeds, the function will redirect the user to the my account page. If this fails, the function will redirect to the login route, where an error message will be displayed.

Logout
Route: `/logout/(POST)`
View: `none`
Description: Is used to log a user out, doing so by destroying the current session.

Add room
Route: `/add-room/(GET)`
View: `add-edit-room.php`
Description: Is used by room owners to edit their room information. The route checks if the user is logged in and if the user is an owner. If the user is not logged in, he will be redirected to the login page. If the user is not an owner, he will be redirected to the my account page and an error will be displayed. If the user clicks on the add room button, the form data will be sent to the add room post route.

Add room
Route: `/add-room/(POST)`
View: `none`
Description: Attempts to add the information of the form into our database. If this fails, the function will return and the route redirects the user to the add room GET route. The form data that was filled in will be preserved. If the this succeeds, the room is added to the database to the room table and the user will be redirected to the single room page.

Edit room
Route: `/edit-room/(GET)`
View: `add-edit-room.php`

Description: Is used owners to edit their room(s). If a user is not logged in and browses to this route, the user will be redirected to the login page. If the user is a tenant, the user will be redirected to the users account page where an error message will be displayed. If the owner does not own the room that the owner tries to edit, the owner will be redirected to the single room page where an error message will be displayed.

Edit room
Route: `/edit-room/(POST)`
View: `none`
Description: Attempts to add the new room information to our database. If this fails, the user is redirected to the edit room page where an error message will be displayed. Form data will be preserved. If this succeeds, the room information will be added to the rooms table and the user will be redirected to the single room page of that specific room.

Remove room
Route: `/remove-room/(POST)`
View: `none`
Description: Attempts to remove the room information from the database. If successful, the user will be redirected to the rooms overview page and a success message will be displayed.

Opt-in
Route: `/optin/(POST)`
View: `none`
Description: Attempts to store the opt-in information in the optin table. If successful, the user will be redirected to the single room page of the room the user opted-in for. A success message will be displayed.

Opt-out
Route: `/optout/(POST)`
View: `none`
Description: Attempts to delete an opt-in from the optin table. When the If successful, the user will be redirected to the single room page of the room the user opted-out for. A success message will be displayed.

Version Control System
The CVS GitHub was used in the development of this application. This version control system allowed us to work together on the application on one repository. This repository always contained the most recent version of our application, while preserving older version to look at.

Security
We used multiple methods to defend our website from various exploitations. Against unwanted visitors, we set up numerous checks in our GET routes to ensure that only authorized users would be able to access these pages. We also applied these checks to several functions as extra measure. Most safety features were implemented in the routes that deal with accounts, profiles, rooms and opt-ins. These checks include login checks, role checks (owner/tenant) and checks to confirm if a user updating the data is the actual owner of the data to be changed.

To prevent Cross Site Scripting, we implemented the htmlspecialchars() function when echoing form data. Our image upload system also has numerous checks to ensure only .png and .jpg images are uploaded in addition to checking if the file is really an image file.

To prevent SQL injection, we implemented PDO prepare statements in any function that uses SQL. We also added form validation to control the form input.

Key features regarding design
User friendly design: We consider the following key values to be fundamental for a user friendly design:

- The design should be pleasing to the eye.
- The design should not distract the user from the content.
- The design should be functional.

The above values make for easier user interaction and engagement, whilst not distracting the user from the sites purpose: make users either find a room to live in or list a room available for rent.

Responsive: Our website is made fully responsive to improve its general accessibility. Whether the user visits the website on mobile, tablet or desktop, all information will be displayed correctly on each device whilst also preserving user friendliness.

Clear data overview: All website sections that contain data are designed to present this data as clear as possible. A clear data overview prevents user for having to look any further for the data they

consider important, such as room information or a tenants personal details.

Multiple image upload: Our website supports multiple images per room. Being able to view more than one image makes for a better picture of the room. If a room contains multiple images, the images are displayed in a slider that preserves their aspect ratio. Preserving the aspect ratio prevents the removal of potential crucial information.

Room locations are displayed on a map: Each single room page contains a map with the rooms location. This gives users an immediate overview of its location and eliminates the need for users to look it up themselves.

Navigating is easy: The navigation of our website is presented such that users can find the information they are looking for with minimal effort. Not only is the navigation header permanently easily accessible, the website also contains page references on sections where it may be useful. For example, the view room button on the featured room section of the homepage redirects to the corresponding single room page.

Creating, updating or deleting a room or account: The process of creating, updating or deleting a room or account is straightforward. In addition there are multiple failsafes to improve the user experience when doing so. For example, deleting a room or account can be done by the press of button, but requires additional user confirmation to prevent accidental updates or deletes.

The websites dynamically changes based on a users state or role: Aspects such as the navigation header, sections or buttons are displayed differently for different roles or states. To illustrate, certain aspects are or are not visible for specific roles or states, the navigation header changes based role or state, and so does button content and functionality.

Clear user instructions, notifications and warnings: Information directed to the user is displayed in a clear way using a fading message box at the top of the viewport. This box can contain notifications, instructions or warnings. For example, you have successfully logged in after logging in, a phone number consists of 10 digits when entering too few digits or you are not authorized to view this page when viewing a page one does not have access to.

No redundant URL-bar content: GET information that is not important for the user to be displayed in the URL-bar is filtered out. For example, error messages should not be visible in the URL-bar, but a room ID does.

Dynamic single room header: The header image of a single room page is dynamically selected based on a rooms thumbnail.

Key features regarding maintainability

Clear documented code: The code of our application is split up in different sections that are provided with documentation explaining what kind of code each section contains. In addition, each function contains a docstring that gives a short explanation of the function. We also added plenty of comments to make understanding specific code even easier.

Easily maintaining navigation: The navigation bar is initiated using a navigation template to determine which elements should be visible. The bar changes depending on the state and role that is specified in the template. Changing the bar can be easily done through adjusting the navigation template.

Modular code: Our code is written as modular as possible such that plenty of functions can be reused on different parts of the website. For example, the function generating the HTML for the single room section is also used on the homepage to generate the featured room content.

Clear file structure: The applications file structure is well organized and easy to understand. Files for a specific purpose are located in the purposes corresponding folder. In addition, the directories containing the images of a specific room are named after their room ID.

Specific and general CSS: The websites styling is done by making use of specific and general CSS to prevent redundancy. Specific CSS is styling that is specific to a certain view or element, general CSS is styling that is applied to recurring elements of the website.

Easy to backup: Rooms and accounts are easy to backup due to the nature of our database and filesystem. Each room has a unique ID which corresponds to the name of the directory containing the rooms images. The images are automatically linked to the right rooms on restoring a backup.

Division of labor

We did not have a very strict division of labor, though each group member had a main focus. Thi-

jmen's main focus was the design of the website. This includes front-end and also back-end work, e.g. developing the functions that generate HTML. Folkert has mainly focussed on the back-end and did a bit of styling. Hylke's main focus was the database design and implementation, while also working back-end and its correspondence with the database.

Taking our GitHub log into consideration, it is fair to say that every group member delivered an equal amount of work. It is also important to state that we only worked on the project when we were physically together, ensuring that every group member spent exactly the same amount of time working as the other members. In addition, always physically working together sped up the developing process. We would develop the front-end and back-end of an application element simultaneously, to make sure these parts were immediately compatible, avoiding problems later on.

4 Discussion

Development

Overall we are satisfied with the development process of our application. We created clear, well documented and maintainable code and an application that is easy to use for everyone. We have already discussed our key features, so in this section we would mainly like to talk about the difficulties we faced during development.

One of the biggest difficulties we faced during the development of our application was safety, mainly because we were new to application development of this proportion. We made our application as safe as possible using the knowledge we had through our classes, but we think we could have added more security checks if we had the time and knowledge to do so.

Another difficulty we faced was deciding where the images of rooms that were uploaded by the users would be stored. We decided to write a function that stores the images in an automatically created directory that corresponds to the right room ID.

Starting the development of our application, we were not fully sure what the best database design for our application would be. In the end we managed to create a suitable database, but we have adjusted it multiple times during development.

We made sure that every aspect of the fron-end is responsive. Using Bootstrap to do so sped up the (responsive) design process significantly.

Future work

Unfortunately there are still a lot of things that we would have liked to add, but we simply could not due to lack of time. In this section we will go over the aspects that could be added in the future.

We would like to have added the possibility for owners and tenants to contact each other through our application. This would have created more security for the users of our application and would require the user to supply us with less sensitive data. Another major aspect we wanted to improve was the application process for rooms. In the current state of our application it is possible for tenants to apply for a room, after which they have to wait for the property owner to contact them.

We would also have liked to give property owners the possibility to respond to an application, by either rejecting or accepting a request or sending a message to the tenant via our application. A feature that notifies tenants if their opt-in has been accepted or rejected would also be a welcome addition. In the current state of the application, the only way for owners to connect with tenants would be through sending them an email, texting them or calling them. We think it is preferable to have all communication go through the application itself.

Another major aspect that can be improved is posting error messages in the $_SESSION variable. We wanted to keep the URL as clean as possible, which means clearing it from form data and error messages. However, due to lack of time we were not able to implement this functionality.

Another feature that could be added in the future is giving tenants the option to filter the available rooms page. This would make it easier to find a suitable room, especially if the quantity of available rooms grows.

Another feature that could enhance the usability of the application is making the application multilingual. This would be useful because there are a lot of international students looking for a room, and not all international students master the English language. Giving them the option to translate the website to their language could therefore be a useful option.

The last feature that we would have liked to add is allowing property owners to edit their room images, and allow them to choose a thumbnail. This would prevent the property owner from having to

delete the whole room to be able to change the room images. Allowing the owners to choose a thumbnail would prevent them from having to upload the images in a specific order to achieve the thumbnail they want.

5 Conclusion

With the creation of this application we tried to solve the problem regarding housing for new students in Groningen. We tried to create an application on which users could view rooms that are for rent in Groningen and other cities. We have created a flexible CRUD application, which is built using the MVC structure. The application allows users to register either as a property owner or as a tenant. Property owners have the possibility to list their rooms on the website, and get in touch with the tenants that have applied for the room. Tenants can view the rooms that have been listed on the website and can apply for a room. Although there are still numerous features we have not been able to implement due to time constraint, we think that the current state of the application is a large step in the right direction to solve the problem regarding housing for students in Groningen.