Architecture Document

Project Pijper Media

Version 2.0

Client: Kevin Klomp, Eelco Luurtsema, Sanne Vast

PIJPER MEDIA
a family business

Team names:
Daniël Scheepstra
Julian Pasveer
Medhat Kandil
Dilan Adel
Jeroen Klooster
TA:
Alina Matei

Table of contents

Introduction

Pijper Media is a family business in Groningen, with a printing company, a magazine publisher and a large online branch (WeBlog Media). For our project the important part of pijper media is the online branch, they have an editorial team that writes articles about trending and viral posts/posts. We have been asked to create a new and improved version of their “social wall”, this is a web page that shows the trending and viral posts of a given set of accounts on different social media platforms. This application will show the posts from Facebook, Twitter and Instagram.

Important terminology

The trending algorithm is an algorithm designed by our team to have two very important characteristics:

- Being ultra-customizable
- Detecting posts that are getting more attention than others

In our last 2 weeks, we decided to run our program on the localhost as a trial for the upcoming deadlines. We needed to see what’s the best way to detect the posts that get more attention than others. We eliminated the followers count as it is in no way a good marker for that. The thing that turned out to be most important is the increase in engagement between each update. Of course when testing we need as many prototypes as we can get. So we decided to have the following simple algorithm:

```
oldEngagement = post.engagement;
update(post)
newEngagement = post.engagement;

engagementDifference = newEngagement - oldEngagement;

coefficient = 0.0X;
threshold = 10^Y;

if (engagementDifference > oldEngagement*coefficient
                      && newEngagement > threshold)
      post.isTrending(true);
```

We chose our coefficient to be 0.01 (1%) and our threshold to be 1000 (10^3) for testing purposes but this can be changed easily.

Architectural overview

API Design and Endpoints

1. Retrieve home/social wall page
2. Retrieve select category preferences page
3. Store category preferences
4. Retrieve activity page
5. Retrieve my activity page
6. Retrieve calendar
7. Retrieve search function
8. Retrieve viral posts

Retrieve home page/social wall

Description
Retrieves the home page view of the application. This is the social wall which shows all posts in the database gathered from different sources from facebook, twitter and instagram. It also shows the recent activity of users that are writing about a post.

URL

```
Get /home
```

Response format
Code 200

```
return view('home', ['user' => $user, 'posts' => $posts, 'accounts' =>
$accounts], ['categories' => $categories]);
```

Code 200
Retrieving of user model

```
{
  "id": {
    "Type": "Integer"
  },
  "name": {
    "Type": "string"
  },
  "email": {
    "Type": "string"
```

```
  },
  "email_verified_at": {
    "Type": "timestamp"
  },
  "password": {
    "Type": "string"
  },
  "remember_token": {
    "Type": "string"
  },
  "created_at": {
    "Type": "timestamp"
  },
  "updated_at": {
    "Type": "timestamp"
  }
}
```

Code 200
Retrieving of post model

```
{
  "Id": {
    "type": "integer"
  },
  "post_id": {
    "type": "string"
  },
  "caption": {
    "type": "binary"
  },
  "post_url": {
    "type": "text"
  },
  "image_url": {
    "type": "mediumText"
  },
  "is_trending": {
    "type": "boolean"
  },
  "is_viral": {
    "type": "boolean"
  },
```

```
  "engagement": {
    "type": "unsignedBigInteger"
  },
  "old_engagement": {
    "type": "unsignedBigInteger"
  },
  "writer_id": {
    "type": "unsignedBigInteger"
  },
  "posted_at": {
    "type": "dateTime"
  },
  "account_id": {
    "type": "unsignedBigInteger"
  },
  "created_at": {
    "type": "dateTime"
  },
  "updated_at": {
    "type": "dateTime"
  }
}
```

Code 200
Retrieving of account model

```
{
  "id": {
    "Type": "Integer"
  },
  "category": {
    "Type": "string"
  },
  "platform": {
    "Type": "string"
  },
  "data_source": {
    "Type": "string"
  },
  "followers_count": {
    "Type": "unsignedBigInteger"
  },
  "created_at": {
```

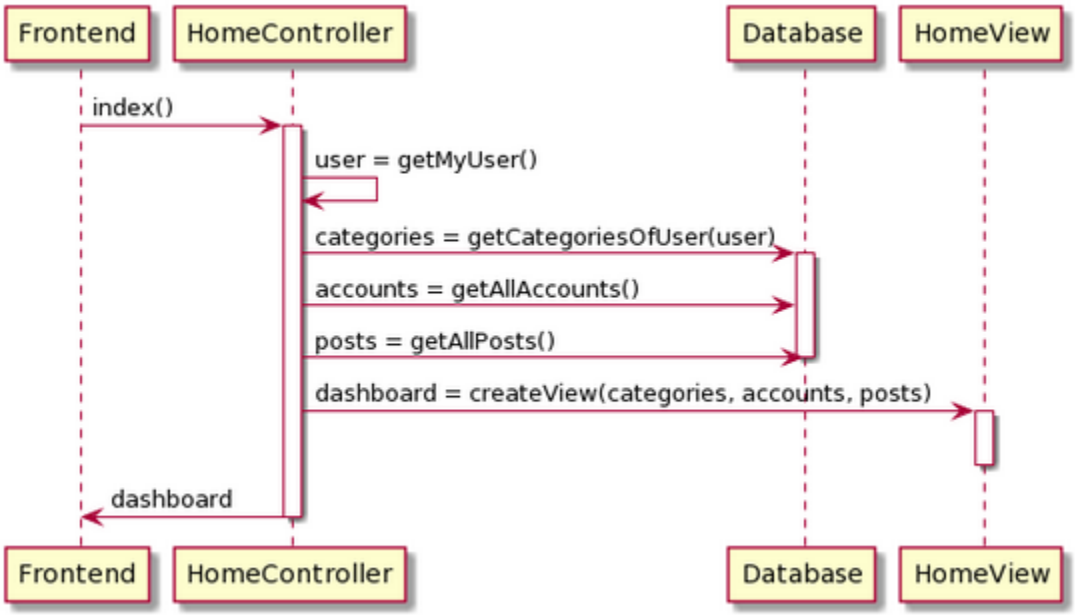
```
    "Type": "timestamp"
  },
  "updated_at": {
    "Type": "timestamp"
  }
}
```

Code 200
Retrieving of category model

```
{
  "user_id": {
    "Type": "unsignedBigInteger"
  },
  "News": {
    "Type": "tinyInteger"
  },
  "Showbizz/Entertainment": {
    "Type": "tinyInteger"
  },
  "Royals": {
    "Type": "tinyInteger"
  },
  "Food/Recipes": {
    "Type": "tinyInteger"
  },
  "Lifehacks": {
    "Type": "tinyInteger"
  },
  "Fashion": {
    "Type": "tinyInteger"
  },
  "Beauty": {
    "Type": "tinyInteger"
  },
  "Health": {
    "Type": "tinyInteger"
  },
  "Family": {
    "Type": "tinyInteger"
  },
  "House and garden": {
    "Type": "tinyInteger"
```

```
  },
  "Cleaning": {
    "Type": "tinyInteger"
  },
  "Lifestyle": {
    "Type": "tinyInteger"
  },
  "Cars": {
    "Type": "tinyInteger"
  },
  "Crime": {
    "Type": "tinyInteger"
  }
  "created_at": {
    "Type": "timestamp"
  },
  "updated_at": {
    "Type": "timestamp"
  }
}
```

Flow of Endpoint



Retrieve Select category preferences page

Description

Retrieves the page in which the user can select the categories he/she wants to base their feed on.

URL

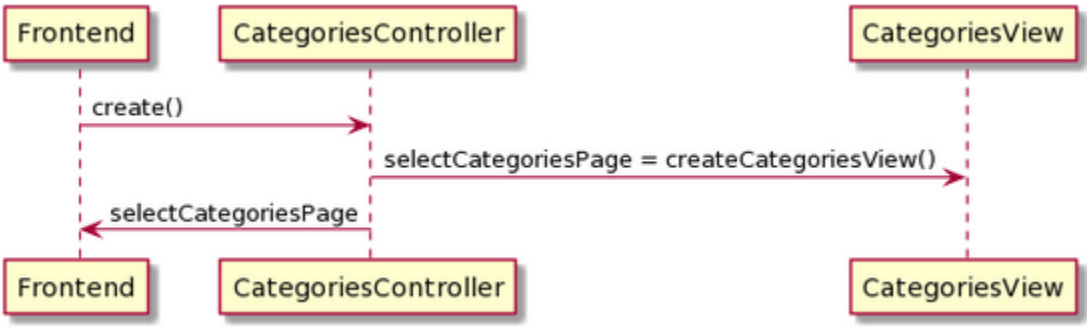
```
Get /categories
```

Response format

Code 200

```
return view('categories');
```

Flow of Endpoint



Store category preferences

Description

Once the user registers, the categories need to be stored in order to base the feed on those categories.

URL

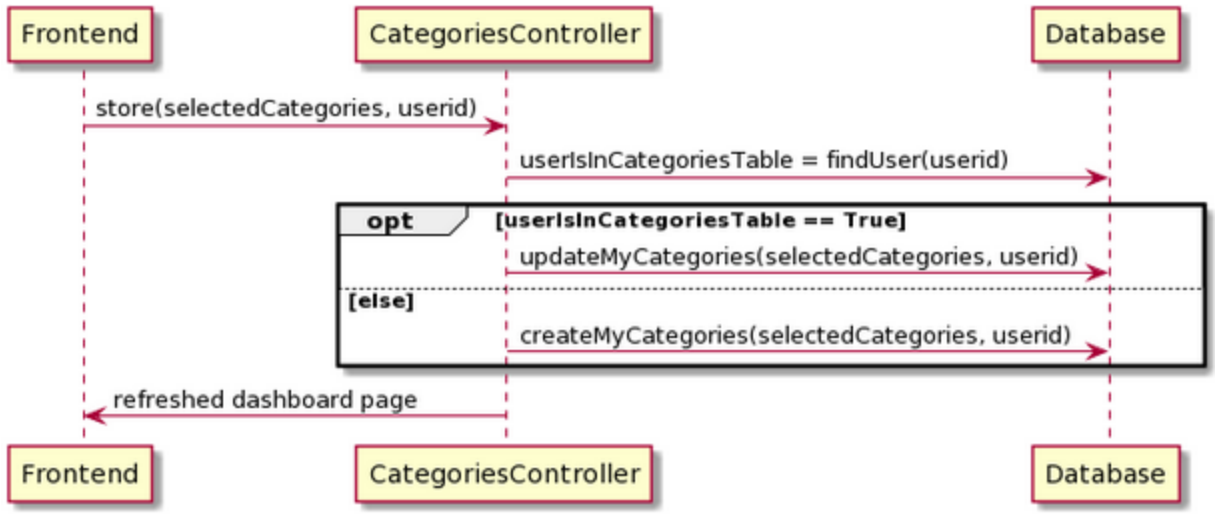
```
Post /categories
```

Response format

Code 200

```
return redirect('/home');
```

Flow of Endpoint



Retrieve activity page

Description
Once the user marks a post as “being written about”, the post is removed from the feed and will be shown in the activity page. This endpoint will return a view of all posts that are being written about (the activity page). The user id is known by using the auth()->user() function, which gets the id of the user who requested the endpoint.

URL

```
Get /activity
```

Response format
Code 200

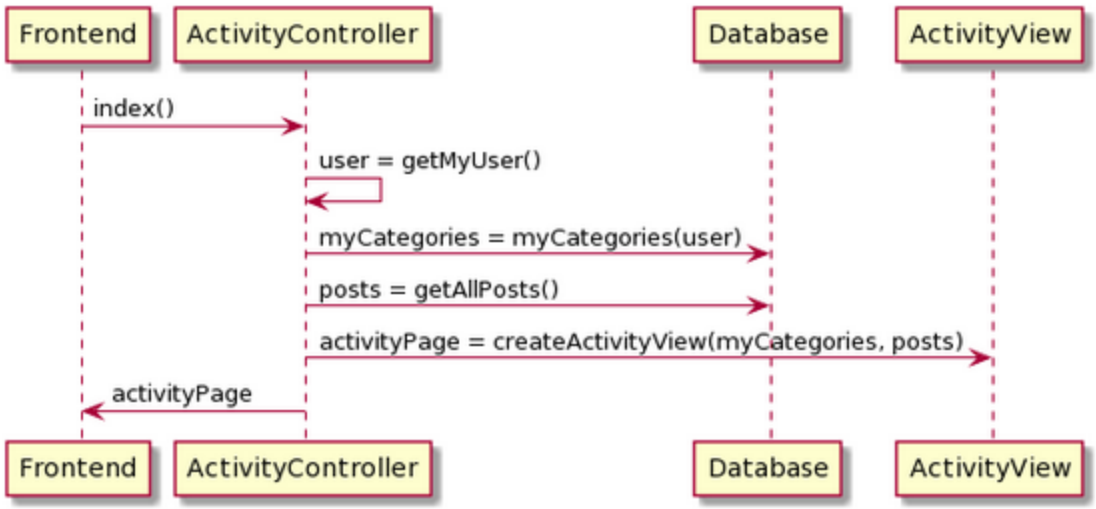
```
return view('activity', ['user' => $user, 'posts' => $posts], ['categories'
=> $categories]);
```

Code 200
Retrieving of user model as seen in a previous endpoint

Code 200
Retrieving of post model as seen in a previous endpoint

Code 200
Retrieving of categories model as seen in a previous endpoint

Flow of Endpoint



Retrieve my activity page

Description
This endpoint will return a view of all posts that are being written about by the current user (the my activity page).

URL

```
Get my_activity
```

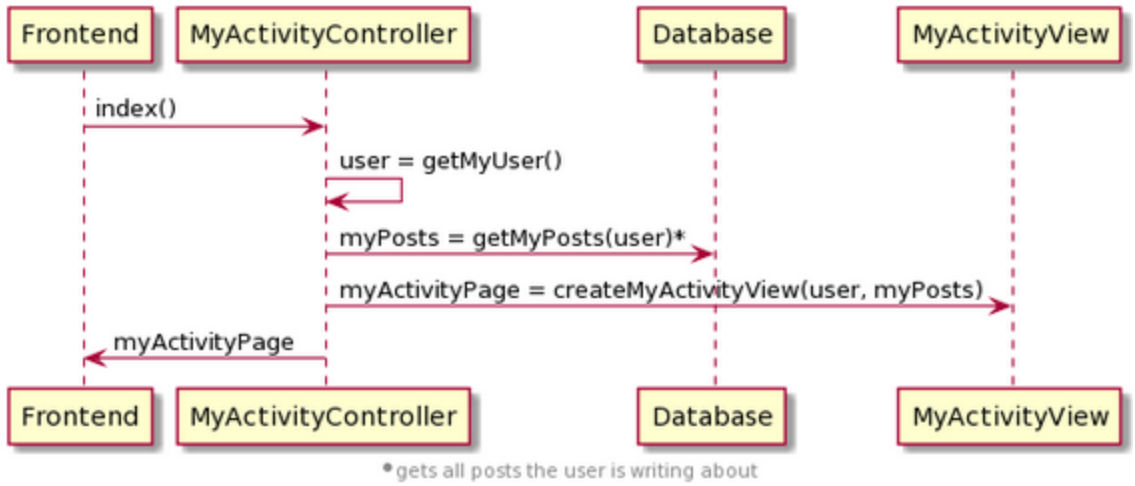
Response format
Code 200

```
return view('my_activity', ['user' => $user, 'posts' => $posts]);
```

Code 200
Retrieving of user model as seen in a previous endpoint

Code 200
Retrieving of post model as seen in a previous endpoint

Flow of Endpoint



Retrieve calendar page

Description
The calendar is a separate page with a calendar that's mostly filled and used for with birthdays of celebrities.

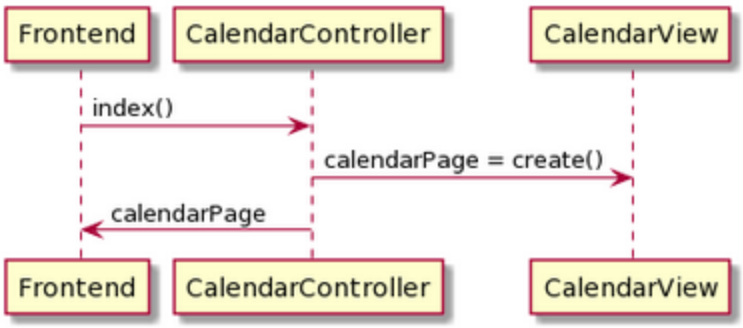
URL

```
Get /calendar
```

Response format
Code 200

```
return view('calendar');
```

Flow of Endpoint



Retrieve search function

Description
Retrieves the posts found when using the search function, in other words, retrieves the posts that contain the given text. We know what the given text is, by gathering the text that has been input in the text field of the search bar.

URL

```
Get /search
```

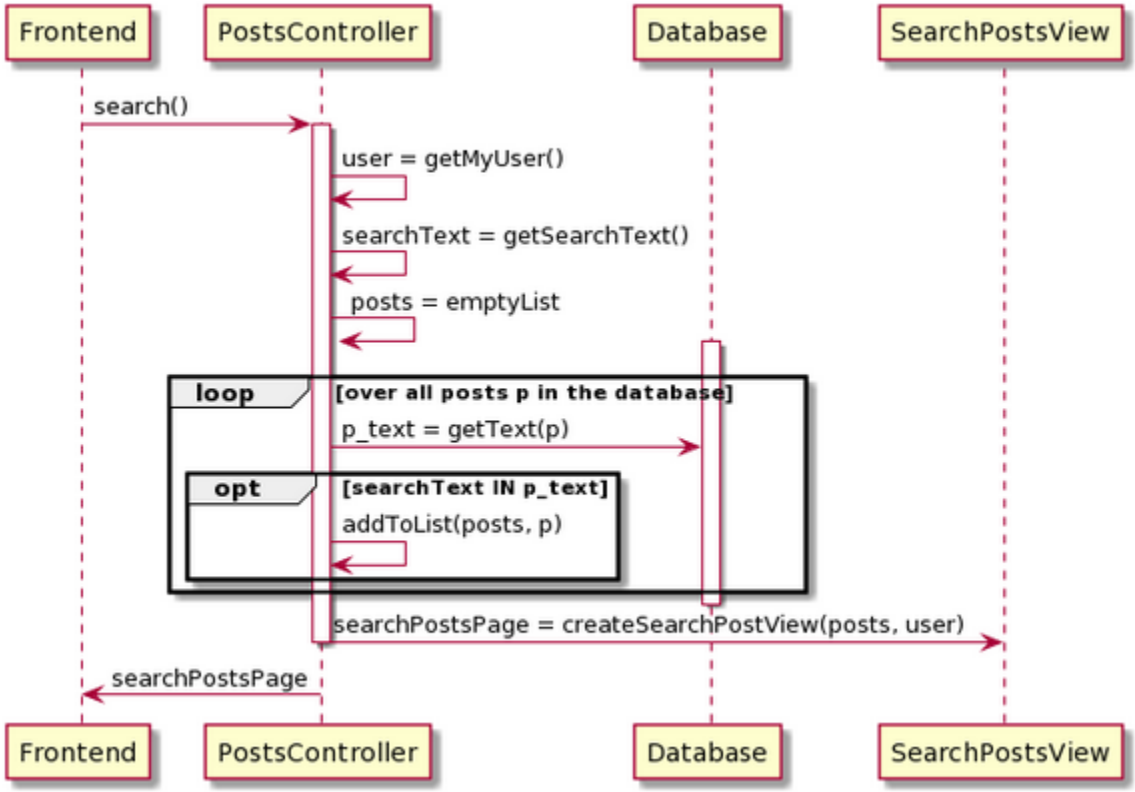
Response format
Code 200

```
return view('search', compact('posts', 'user'));
```

Code 200
Retrieving of user model as seen in a previous endpoint

Code 200
Retrieving of post model as seen in a previous endpoint

Flow of Endpoint



Retrieve viral posts

Description
Retrieves the posts that are considered by our algorithm to be viral.

URL

```
Get /viral
```

Response format
Code 200

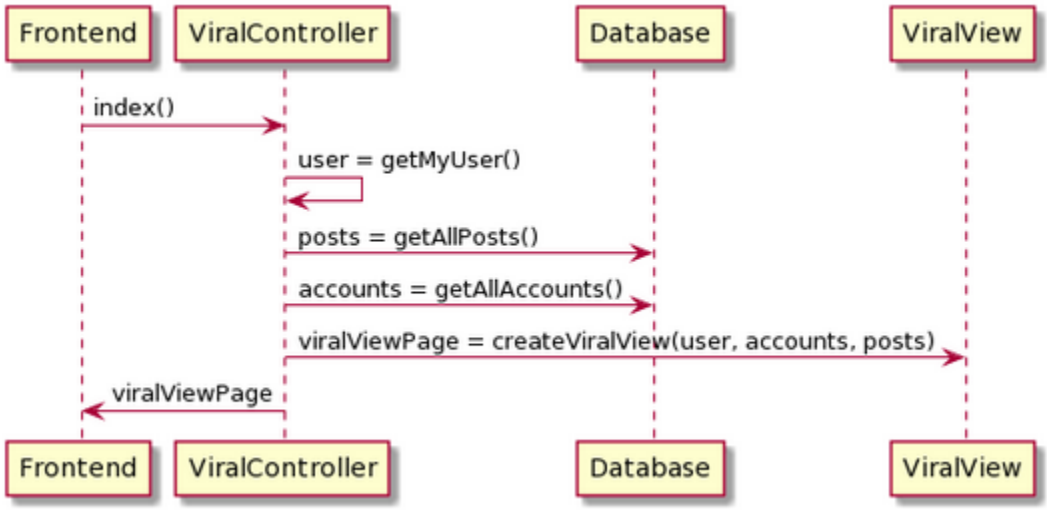
```
return view('viral', ['user' => $user, 'accounts' => $accounts, 'posts' =>
$posts]);
```

Code 200
Retrieving of user model as seen in a previous endpoint

Code 200
Retrieving of account model as seen in a previous endpoint

Code 200
Retrieving of post model as seen in a previous endpoint

Flow of Endpoint

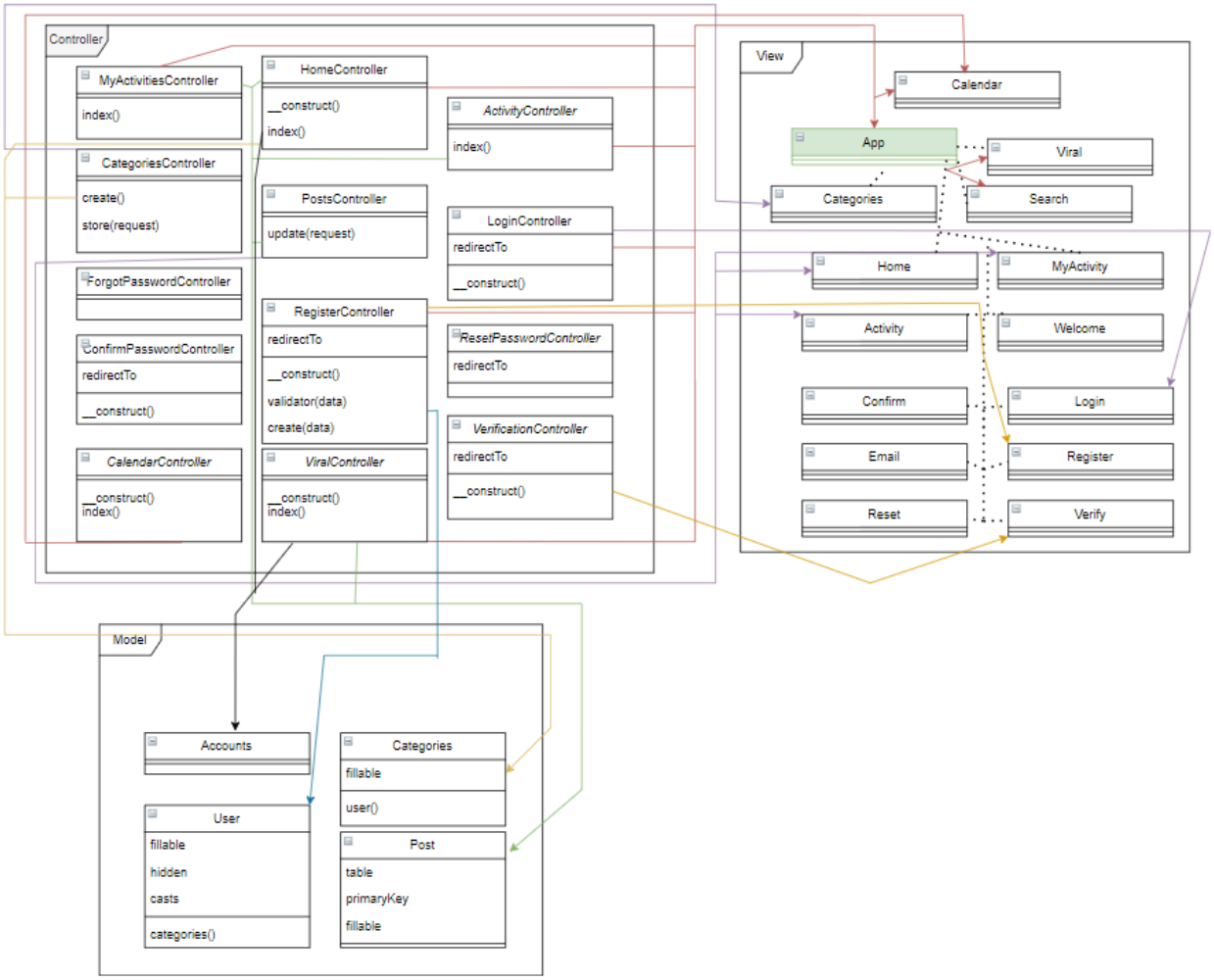


Overall Design Decisions

We are working with Laravel, which is a PHP-web framework. It is meant to be used for web applications that are based on the Model - View - Controller architecture. We are using this architecture.

Class Diagram

The different coloured arrows are meant to show the connections and routes. The different colours don't mean anything. They are there just to make it not confusing when the arrows cross. In view and controller we have the classes and in the View we have blade.php files. The highlighted App in view is an abstract file, which is extended by most other blade.php files (as can be seen by the dotted line).



Back-end Design

Database Design

The main tables in our database are depicted in the following diagrams. We have decided to leave two columns (created_at and updated_at) out of the tables, we do this so the diagram will be more readable.

In the categories table, we store the categories that the user would like to be shown in their personal dashboard. The user_id is a Foreign Key from the ID in the table Users. We have left some categories out of the shown table, because the table would be too long otherwise. This is why the last column of the table is “etc…”.
The accounts table contains information about the different accounts/sources from which we gather posts. For clarification purposes, one company (for example, Tasty) might have an account on each platform, in this case a company will have one account for each platform.
In the posts table, we store the necessary information to show the user of the social wall what the post is about and how much engagement it has. It also uses two Foreign Keys, which are account_id (from the id in the accounts table) and writer_id (from the id in the users table).
The users table is a table generated by Laravel and it takes care of the authentication of users when they login.
Laravel automatically creates Primary Keys for the tables, which is ID in each table.

categories

id	user_id	News	Showbizz/Entertainment	Royals	Food/Recipes	etc...

accounts

id	category	platform	data_source	followers_count

posts

id	post_id	caption	post_url	image_url	is_trending	is_viral	engagement	old_engagement	writer_id	posted_at	account_id

users

id	name	email	email_verified_at	password	remember_token

The following tables in our database are necessary for using laravel, but these do not affect our application that much.

password_reset

email	token	created_at

migrations

id	migration	batch

failed_jobs

id	uuid	connection	queue	payload	exception	failed_at

Data gathering

The data gathering for our application is done with the use of the Facebook Graph API, Twitter API and an Instagram Scraper. With the use of these API's and scraper we gather posts from a set of different sources/accounts. The data we gather from these posts is then stored in our database.

To make sure that the posts are kept up to date, we make use of a task scheduler in Laravel. This scheduler will run the data gathering process every ten minutes. When this scheduler runs every ten minutes, there will of course be posts that are already in the database. In this case we just update the engagement, old_engagement and updated_at columns of the post. The engagement column will contain the newly acquired engagement value, and the old_engagement column will contain the previous engagement value.

Front-end Design

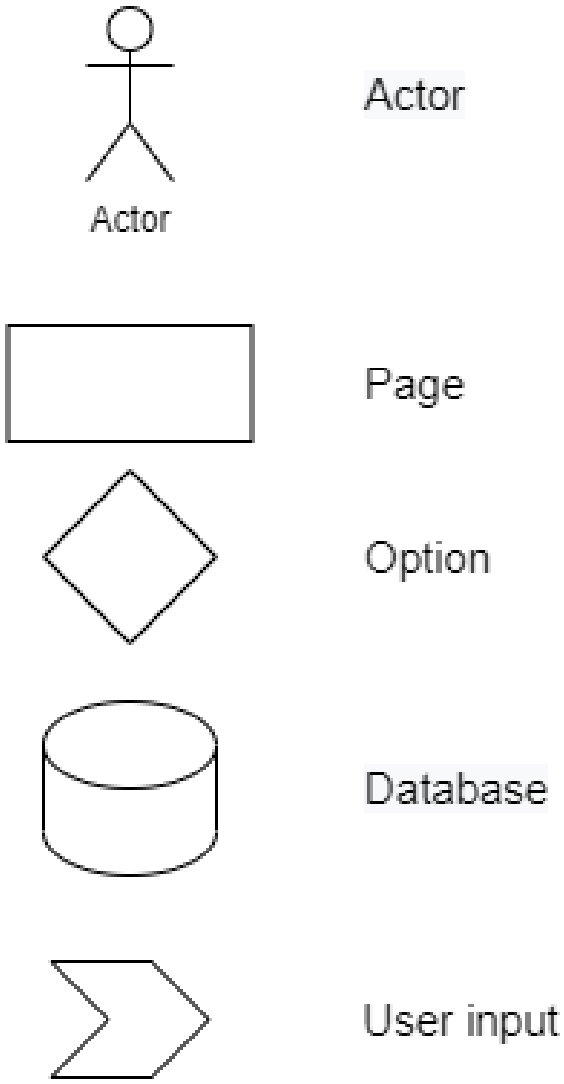
The frontend consists of a desktop web application that the user can use. We use Laravel in order to create the view of this desktop web application. This is done via the blade template, in which the view of the page can be easily split up in sections.

The first page to which the user is brought to is the welcome page. Next, the user has the option to log in to the application and it will bring the user to the login page. In case the user enters his/her correct credentials (checked by the backend), he/she will be redirected to the categories page, otherwise the user will get notified that he/she entered incorrect credentials. Once the categories have been checked off, the user will be brought to the homepage. The homepage consists of several sections and options. When the user is on the homepage, there is also the

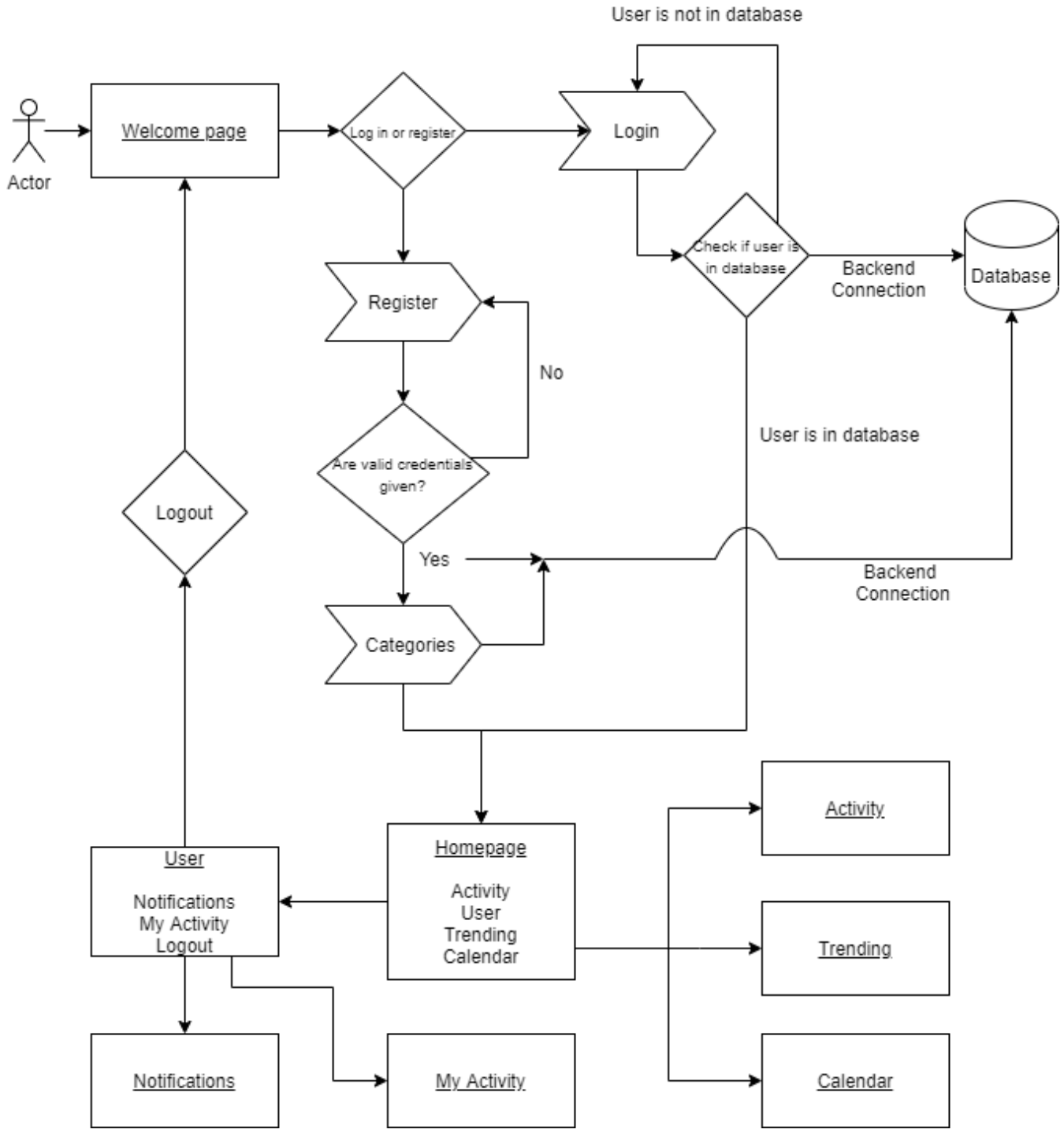
option to go to the trending page, the activity page, changing the categories and a dropdown section in which the user can either see his/her checked off posts or log out of the application.

In order to make this all a bit clearer, we will show a diagram corresponding to the above:

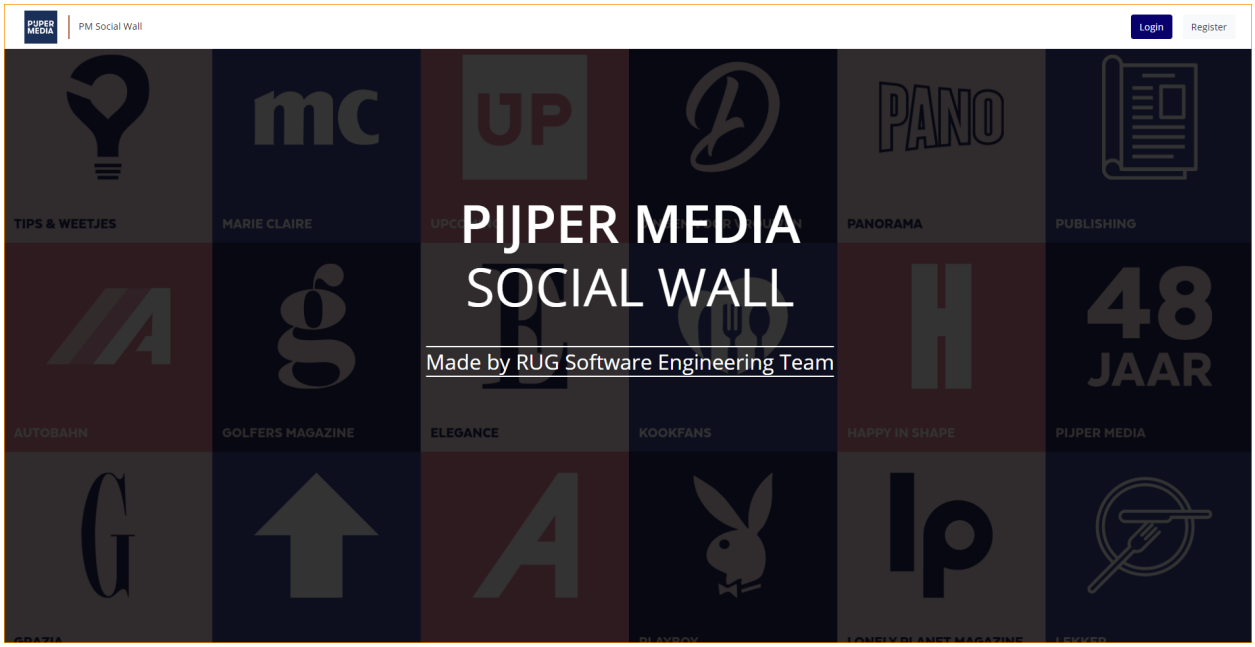
Attributes:



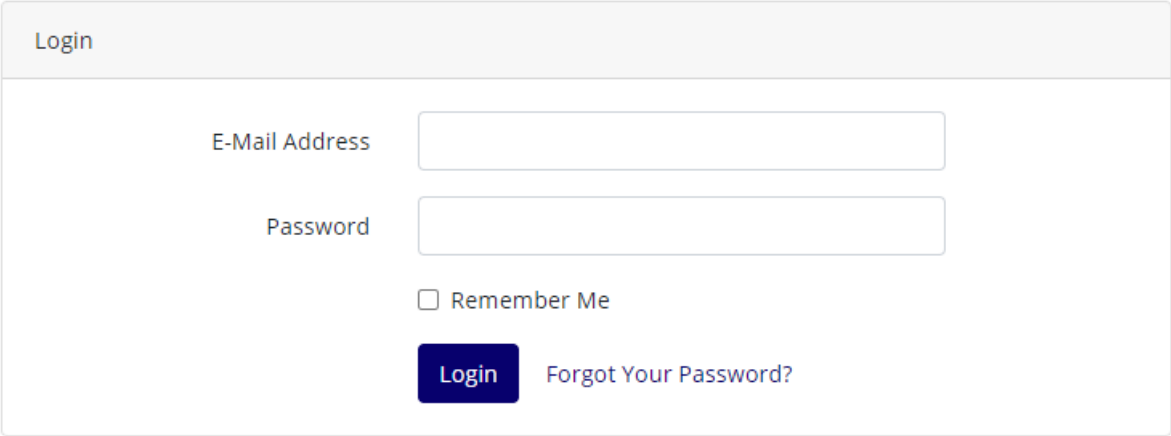
The diagram:


User is not in database
Actor
Welcome page
Log in or register
Login
Check if user is in database
Backend Connection
Database
Register
No
User is in database
Are valid credentials given?
Logout
Yes
Backend Connection
Categories
Activity
User
Notifications
My Activity
Logout
Homepage
Activity
User
Trending
Calendar
Trending
Notifications
My Activity
Calendar

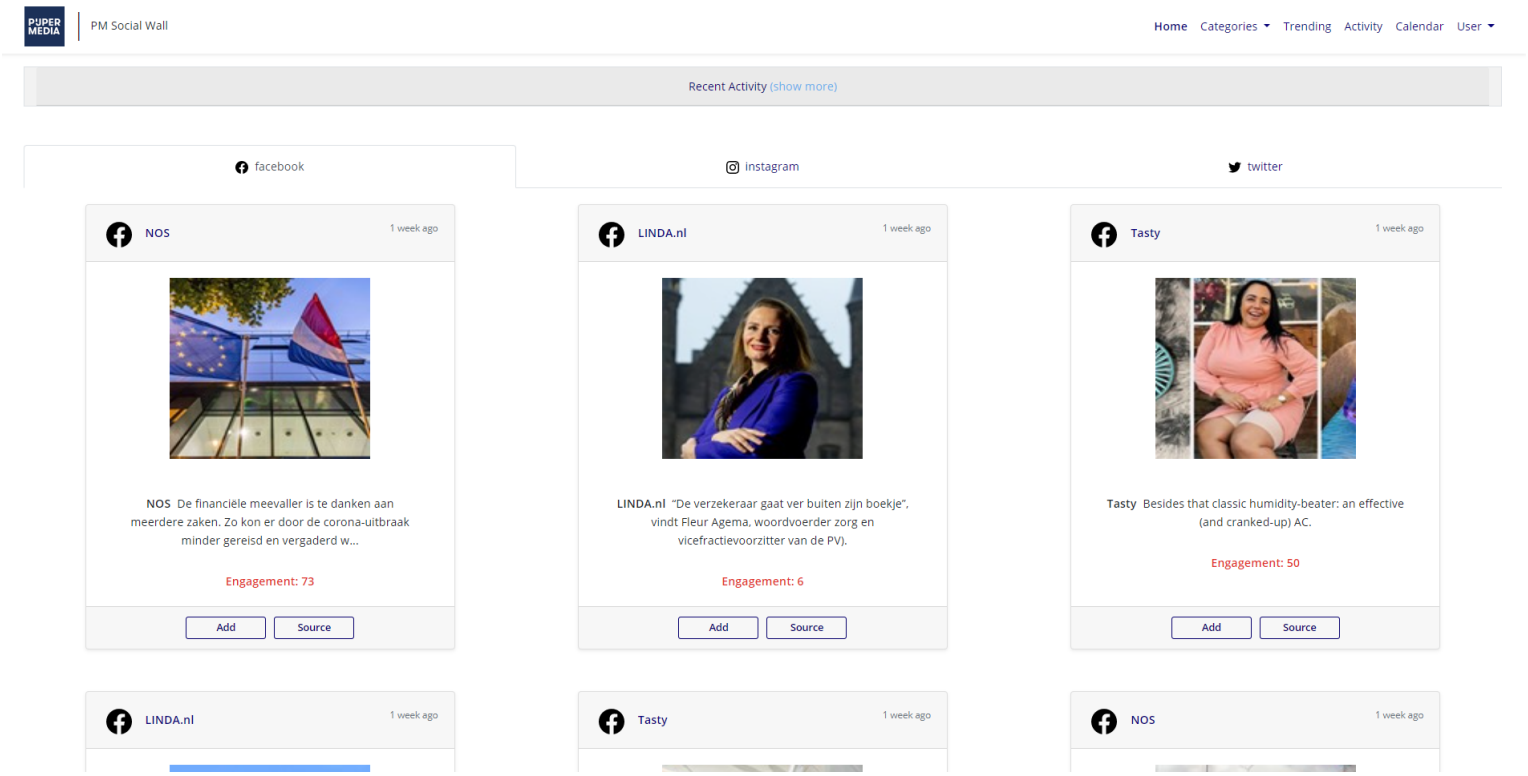
Visualisation of the most important pages:
Welcome page:



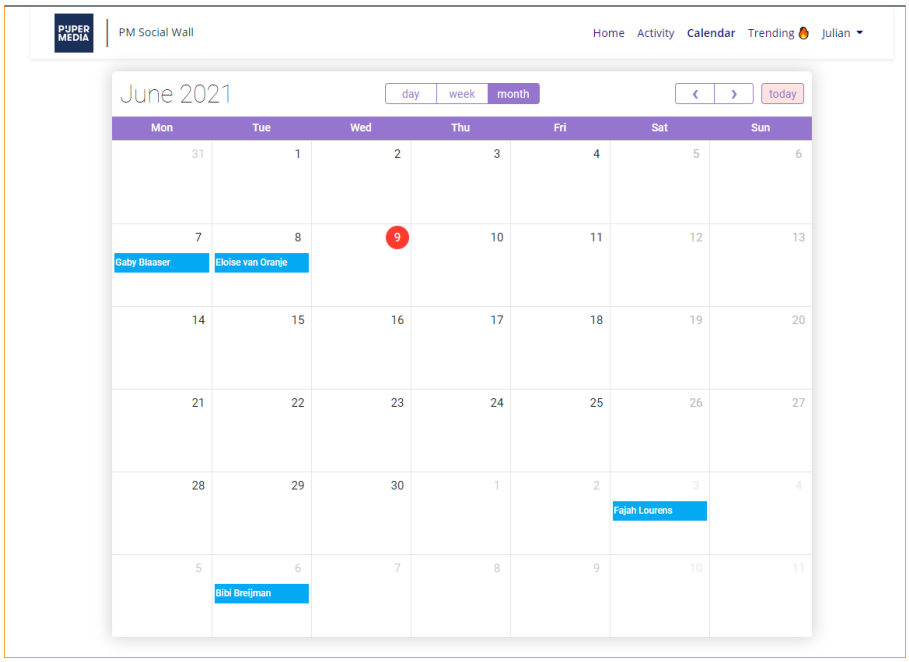
Login page:



Homepage:



Calendar:



Technology Stack

The programming languages that we will use in this project are the following; PHP, Javascript, CSS, HTML.

To develop this application we will use the laravel framework for the back- and frontend and we will also use bootstrap for the frontend. We will create a MYSQL database to store all the necessary data.

There are a couple of reasons why we use this technology stack. Firstly, our client asked us to work with this technology stack, because they work with the same technology stack.

We used Laravel front- and backend, because with this framework it is very easy to create and connect models, views and controllers. Laravel does a lot of the work for you.

Team organization

We have split the team up in the frontend, backend and the inbetween/connector. The frontend will be done by Julian Pasveer and Dilan Adel. The backend will be done by Daniël Scheepstra and Jeroen Klooster, everything will be connected by Medhat Kandil.

Future Improvements

Of course the time frame given to us was not optimal to finish each and every requirement in the best way possible as well as maintaining our own courses. For this, we decided to ask the clients if they would like to continue working with us during the summer. We have decided as a group to do the following if we get the chance to continue working with the company in the summer:

- Cleaning the code, the code does not need refactoring but in some blade.php files code duplication can be neatly retired.
- Testing the program as it is, some testing has been done of course, but unfortunately front end testing hasn't been done in the best ways possible. Especially when looking at testing the frontend side of posts. Due to lack of time we decided to not do this and focus on more important tasks.
- Adding functional requirements, some requirements like: having a database for the calendar, having posts from other pages, adding/removing posts and pages

from individual feeds etc.. haven't been implemented yet. We would add them in the case of working in the summer.
- Launching and serving the website.

The calendar has been implemented in such a way that events can be added to the calendar via using the website, however, this will not be stored in a database and hence is not very useful. We tried adding a database to this in order for it to work properly, however it took us a long time and eventually we decided to use our time for other tasks. Although the calendar cannot be edited via the website itself, it can be edited via the code. This is really easy to do: in the calendar.blade.php file, there is a lot of javascript. Here one can find a section with events. An event can be added here to the list of events and the event will be added to the calendar.

Of course along that we are going to be working on documenting our web app so that whoever comes after us can work on it smoothly.

Change log

Date	**Changes**
Friday 05-03	First draft of the document
Sunday 07-03	Added team organization
Sunday 07-03	Added technology stack
Monday 29-03	Updated architectural overview (everything)
Monday 31-05	Rewrote a lot of unclear text and took the feedback into account
Tuesday 01-06	Added API Design and Endpoints
Monday 07-06	Cleared up some points of feedback
Wednesday 09-06	Cleared up more points given in feedback

Saturday 12-06	Added Future Improvements
Monday 14-06	Updated class diagram