Project Report R&D Team-48

Nikita Kuprins, s1080434 | Role: team lead, interconnections, backend, frontend(only some parts of discussion page), project report, database.
Jiaben Zhao, s1041444 | Role: frontend, project report, database.
Michal Wojtasik, s1069431 | Role: project report(only introduction and evaluation).

1. Introduction

At the end of the year 2019, a highly transmissible disease, known as COVID-19, broke out in Wuhan, China. Not soon after, WHO declared COVID-19 a global pandemic, putting our world under enormous uncertainties. Two years on, COVID-19 is still an ongoing trend and dominates headlines in many media outlets.

Information about covid is booming, and so is disinformation or misinformation. That is why we want to create a covid-themed social platform which gathers news across multiple news sources. It contains a forum where people can discuss Covid by starting up a thread. In addition, the platform has a dashboard which displays Covid-related data per country, ranging from infection cases to death toll.

Our project has its own originality. Reddit is currently a popular community site; people can create a mini forum to discuss topics of their interests. While Reddit provides the land of free expression, it falls short of containing all the related discussions. For example, it is rare for Reddit users to discuss the update of the game inside the subreddit of World of Warcraft because there is already a specific platform for this subtopic. Contrary to Reddit, our product will be a self-contained place for the discourse dedicated to Covid. Also, Reddit has long been the heartland of the young generation, usually filled with dialogues on pop culture; it may have failed to attract users on the other side of the spectrum such as the senior. Our product will brand itself as being user-accessible.

Our project also appeals to many users. For example, a study*(Motta, M., Stecula, D., & Farhart, C. (2020). How Right-Leaning Media Coverage of COVID-19 Facilitated the Spread of Misinformation in the Early Stages of the Pandemic in the U.S. Canadian Journal of Political Science, 53(2), 335-342. doi:10.1017/S0008423920000396)* found the media coverage of Covid-19 is often reflected by the media's political stance and as a result, a reader's perception of the pandemic will likely be distorted from this single media outlet. People usually also express frustration with gathering credible information as much information is scattered around the Internet. Their frustration becomes more prominent in the case of Covid because Covid opens up so many sub-topics for discussion such as vaccination, virus strains, local preventive measures and so on. It is very hard for people to get a full picture by consuming one media source, let alone the lack of interactivities on those common media platforms.

Identification of potential users:

As we mentioned, the product caters for all people because Covid is a topic concerning everyone on the planet regardless of their age or gender. In terms of user interface design, it highlights the need for the so-called 'Inclusive design', making the interface as simple and intuitive as possible.

2. Description

2.1 & 2.2 Properties and Product justification

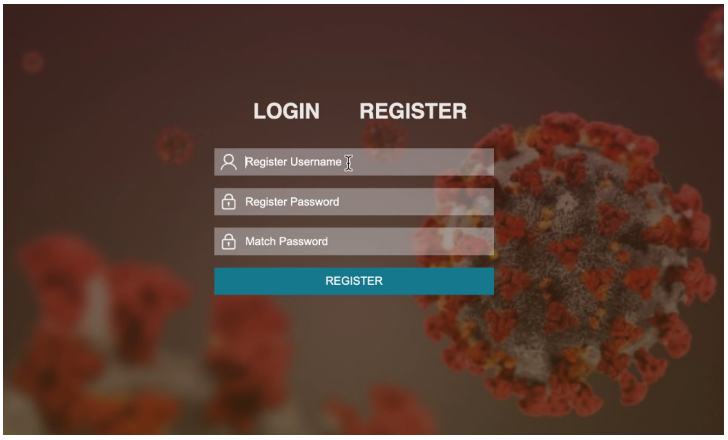
Our project includes three main functionalities:

- A dashboard displaying Covid-related data. It consists of an interactive world map and two bar charts on the confirmed cases and death toll respectively.
- A news aggregation page featuring many articles across multiple outlets.
- A forum where users can create a thread or post, and respond to a post.

Those functionalities are arranged on different web pages for a clear layout to avoid confusion among users. There is a user-friendly navigation bar so that users can switch across different functionalities. In the following sections, we will walk you through each web page with illustrations.

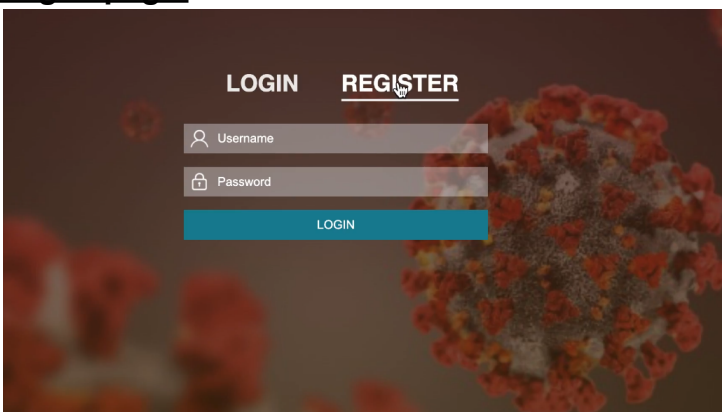
To access those web pages, a user must be already logged in:

Account Register page:



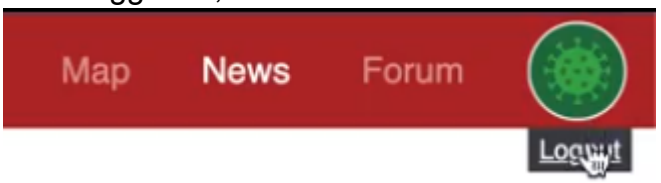
The data required for creating an account is kept minimal. There are three text fields: `Username`, `password`, `repeated password`, each attached with an icon, rendering in transparent background. This backdrop is the close-up of the virus, which shows off the ‘individuality’ of our product. Contrary to convention, our product does not collect any personal data for registration. This is also part of the universal design strategy. However, it might backfire, as it leaves users with no other authentication means to reset their password.

Log in page:

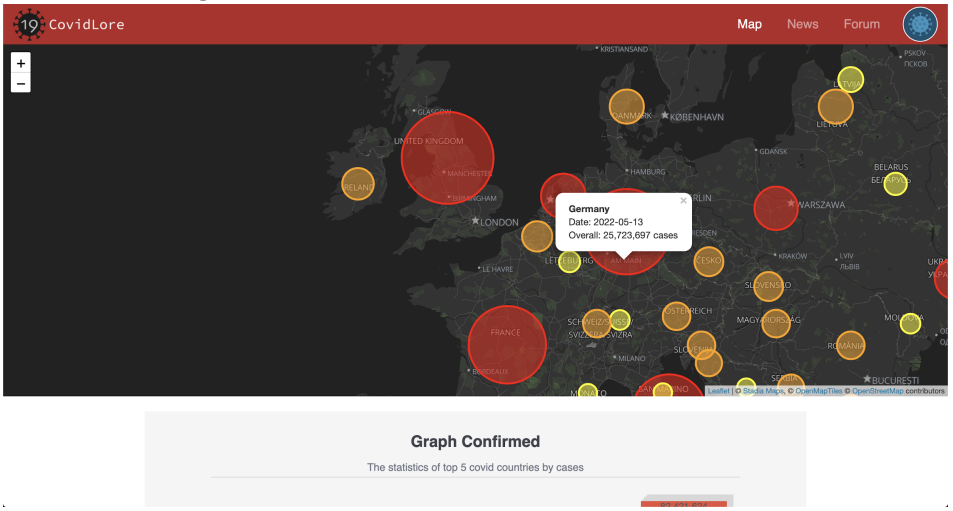


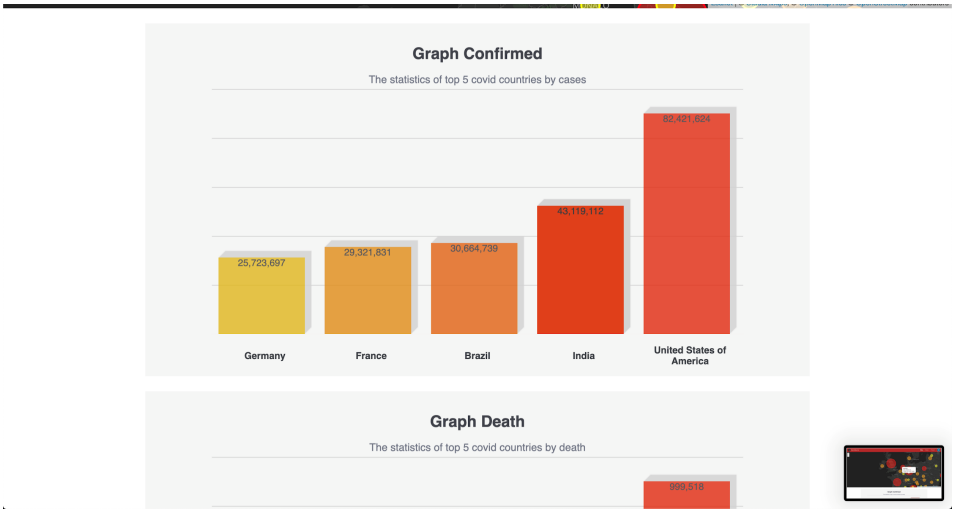
This page is symmetrical to the registration page. Once a user is logged in, it will be redirected to the Forum page (explain later) by default.

Once logged in, the user can whenever choose to log out by clicking logout.



Dashboard page:



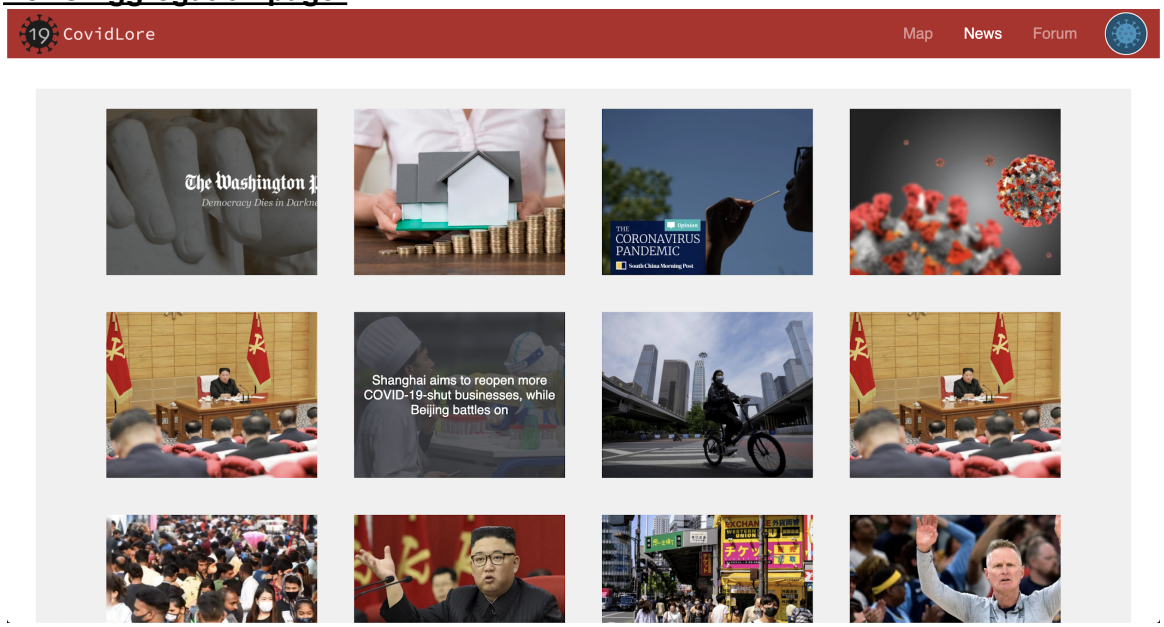


At the top, there is a full-width world map spanning the entire width of the browser. The map discloses two pieces of information per country: the total number of infection cases and the date of data collection. Note that, for infection cases, the data takes into account the recovered covid patients, not just the active cases. Because the number of total infections is usually very large, users might then become insensitive to the large number with a series of digits. To better illustrate the severity of the confirmed cases, we place a dot on the map at the centre of the country – the larger the dot, the more confirmed cases a country is experiencing. We also put the data into a global context, drawing comparisons from other countries – the red dot indicates a country is taking up a very large proportion of covid infections worldwide. More specifically, we use the three-colour scheme to indicate the severity, which is: red, orange, and yellow.
Regarding the second data (the date of collection), we feel it is necessary to show when the last time the data was updated. The date is shown in the format of 'yyyy-mm-dd', because, as it stands, covid seems to be lingering for a few years ahead. The map is highly interactive in that it enables users to zoom in, zoom out and drag across, therefore they should feel very easy to navigate. The map is facilitated by the open-sourced API, known as Leaflet.

The map is followed by two bar charts at the centre of the webpage. The first bar chart ranks the top five countries with the most confirmed cases. The bars are progressively shown in an ascending order, where the magnitude is also reflected in the brightness of the background colour. Each bar contains a label of the exact number. Because the bar chart is very straightforward, we specifically leave out the legend for simplicity of illustration. Keep in mind that the bar height is not skewed to avoid misinterpretation.

Next is followed by another similar bar chart which ranks the countries in terms of death. We recognize that the previous statistics on the total infection cases can be swung by a wide range of uncontrolled factors, thus it is a less reliable data. For example, a particularly high number of infections occurring in a country can be attributed to the fact that the country is characterized by a large population density, or the country is currently suffering from a more deadly virus strain, or the country is very high in people mobility due to its geographic location. We believe the bar chart on death toll is more important as it gives people a clearer idea of the impact of the virus on a country as well as a country's performance in handling the pandemic.

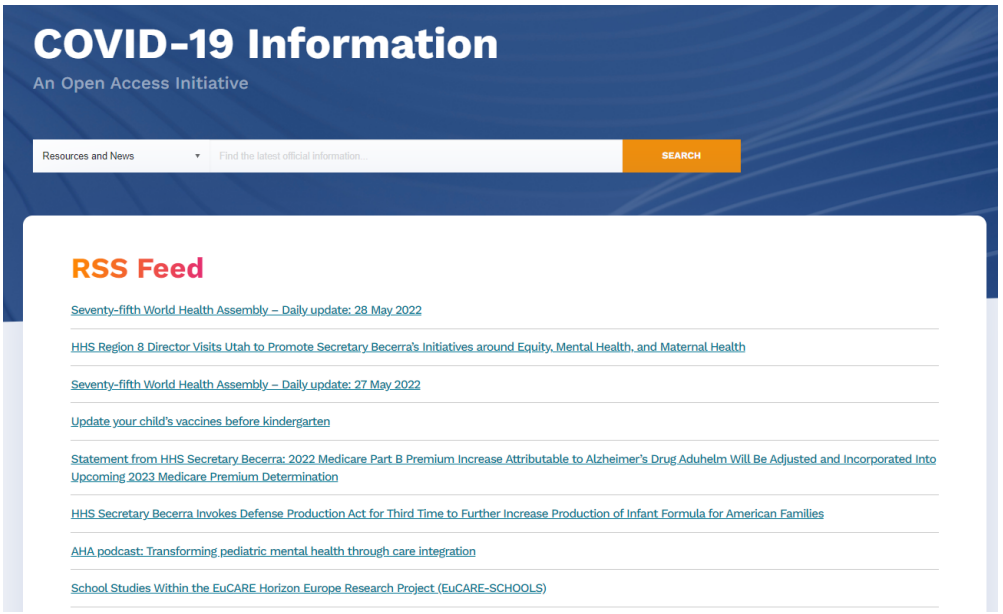
News Aggregation page:



This page provides a place for users to browse through the news on Covid which are collected from numerous well-established public presses. As part of our efforts to bring in a well-balanced mix of information, our news sources include The Washington Post, BBC News, South China Morning Post and so on. The selection is purely based on the new aggregator API which captures news around the globe including countries such as North Korea.

The whole page renders articles in the semi-waterfall fashion with four columns, giving users an immersive viewing experience. Each item is filled with the image featured in the news article. When a user hovers his mouse around the item, the exact title will appear, dimming out the image. If a user feels interested after reading the title and further clicking on the item, he will be redirected to the external link pointing to the article.

There are some websites with covid news aggregation features. We present two examples here.



COVID-19: Legal responses in Asia Pacific

- APAC News Aggregator

APAC news aggregator

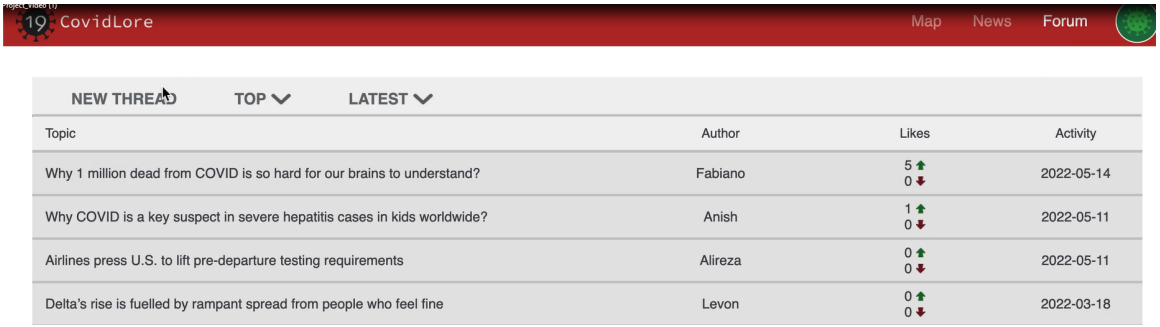
Date	Jurisdiction	Title	Author
25-May-2022	China	Airbnb ends rentals in China to focus on outbound tourists	ABC News
24-May-2022	Thailand	Cabinet okays extension of hotel subsidy scheme	Bangkok Post
23-May-2022	Cambodia	Effective economic steps boost tax revenue	Khmer Times
20-May-2022	China	China cuts key mortgage reference rate as Covid bites	Malay Mail
19-May-2022	India	Hybrid work to heatwave: why India is facing its worst power crisis in over six years	Times of India
18-May-2022	China	China's April new home price index falls for the first time in seven years as Covid-19 adds to the slumping property market's cash woes	South China Morning Post
17-May-2022	China	Economy likely to recover after April doldrums	China Daily
13-May-2022	Australia	Northern Territory Hospitality Industry warns of staff shortages ahead of Darwin dry season	ABC News
12-May-2022	South Korea	Top 1000 firms' sales hit new high in 2021 amid pandemic	The Korea Times
11-May-2022	Japan	Japan's FY2021 current account surplus shrinks 22% on fuel price rise	Japan Today
10-May-2022	Thailand	Keeping tourists coming back	Bangkok Post
9-May-2022	China	Xi Warms up China's economy, but COVID-19 narrows options	Channel News Asia
6-May-2022	Cambodia	Cambodia received 220,000 international tourists in the first 4 months of 2022	Khmer Times
5-May-2022	South Korea	Korea to allow visa waiver for international travellers to Jeju, Yangyang	The Korea Times

Both examples present the news articles in table format and as a result, the whole page is loaded with much the textual information, which might hinder reader experience. Furthermore, our project already entails a dashboard which is a very data-intensive page. It is our design goal to lessen the amount of information our users need to digest in one go. That is the reason why I arrange the articles into small square items and present them in a waterfall-like format.

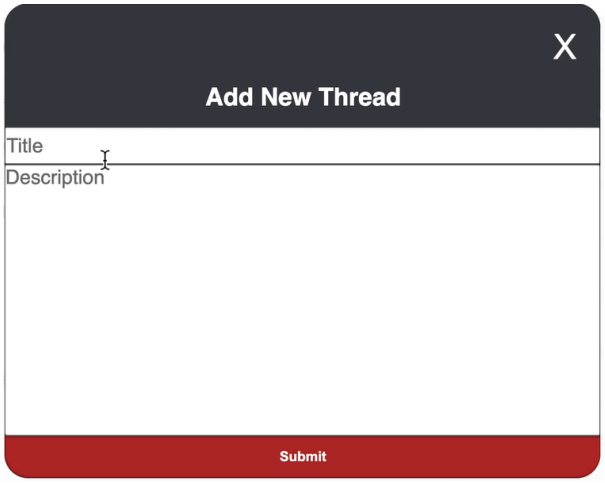
Forum

Our product contains a forum for users to exchange their dialogues on covid. To achieve this functionality, there must be at least two types of pages:

- An overview page where all the threads' title are listed
- A single page for each thread, where the body of the thread is displayed, and users can comment on.

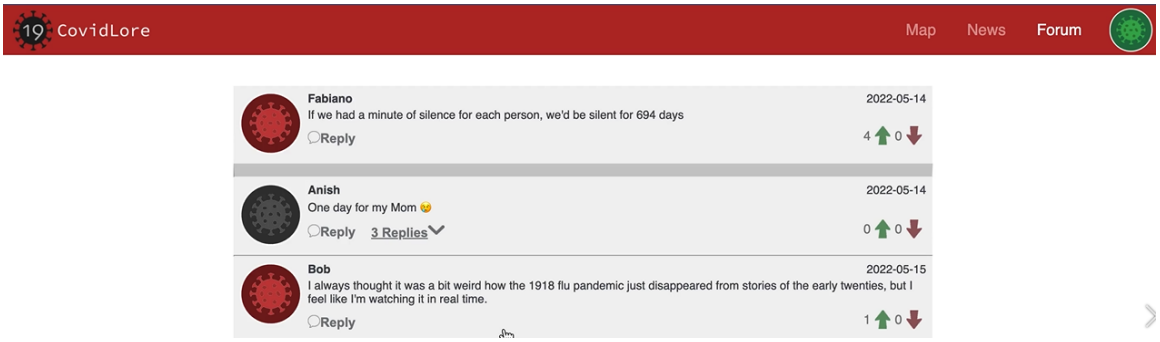


For the overview page, users can create a new thread by pressing the button `NEW THREAD`. Users then can input the thread title and the description before hitting the `submit` button. Once a new thread is submitted, the corresponding thread will appear at the table with author name being username; the activity field is filled with the current date and the scores are all initialized as 0.



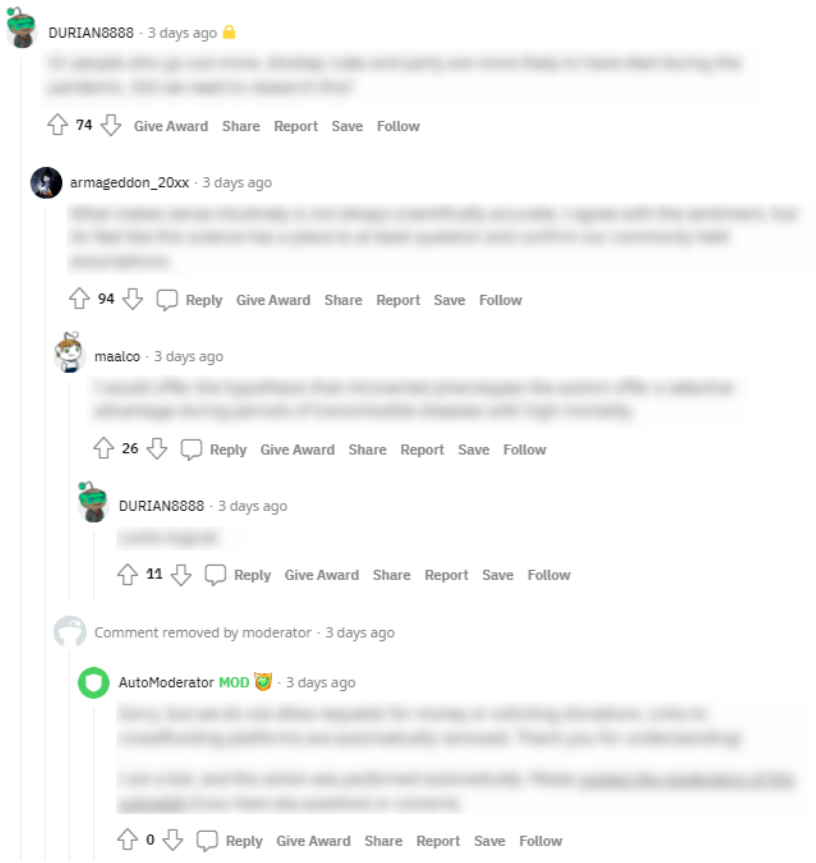
The overview page by default lists the threads in terms of the number of likes so that the most valuable thread also tops the thread list. In addition, users are free to sort the threads alternatively, by their release date. Whenever the mouse hovers a thread at the table, the table body's background color is switched to gray.

Inside a thread page, the thread content is shown along with the avatar of the original poster. Users can respond to the thread by giving reactions `like` or `dislike`. Users can leave a reply to a thread if they click the button `Reply`.



Each reply section contains the similar structure as a thread, that is, the username of the poster, poster's avatar, the date of post and the reaction functionalities are all included.

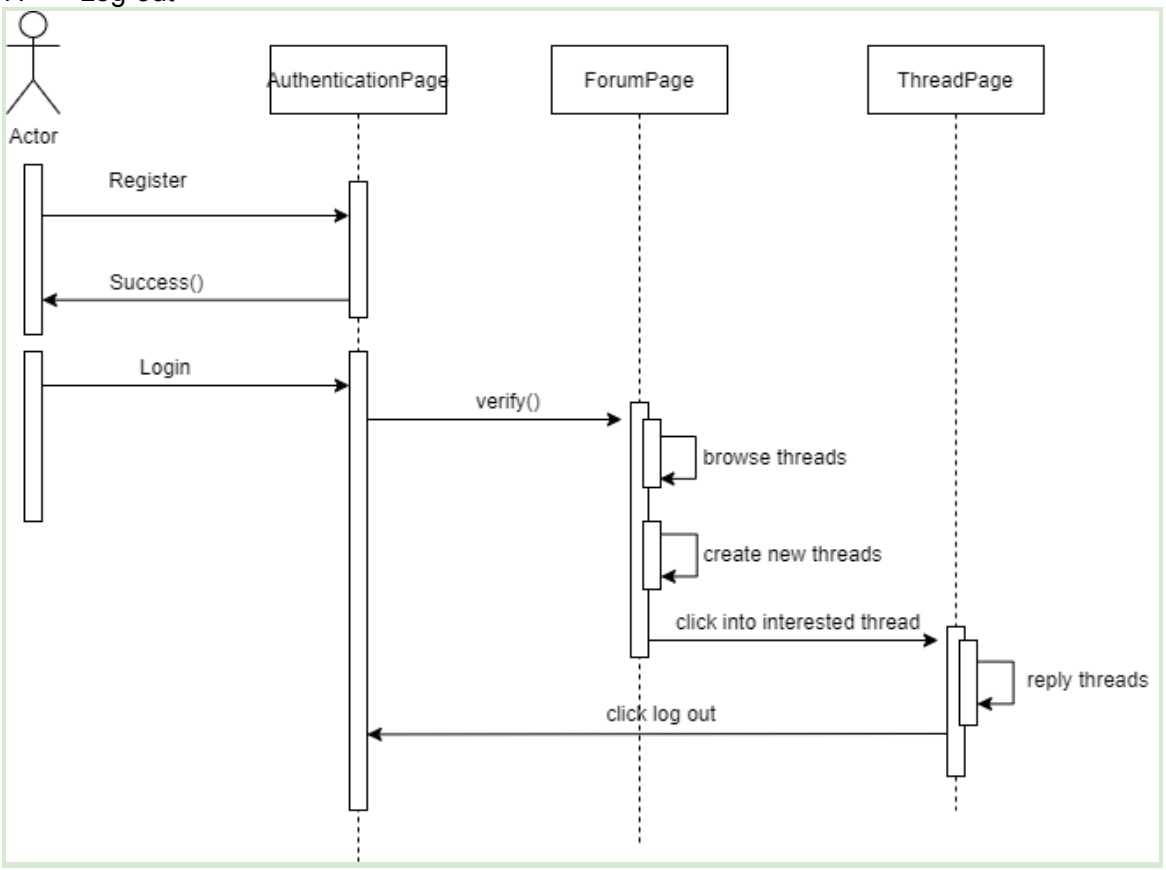
In fact, users can further respond to a reply reclusively, resulting in many collapsible reply sections. We deliberately close off all the inner reply sections by default. This move gives the whole thread page a more straightforward layout, which is different from the Reddit counterpart. The traditional Reddit's style of displaying replies could be off-putting for others as some find it very hard to navigate among a sea of replies.



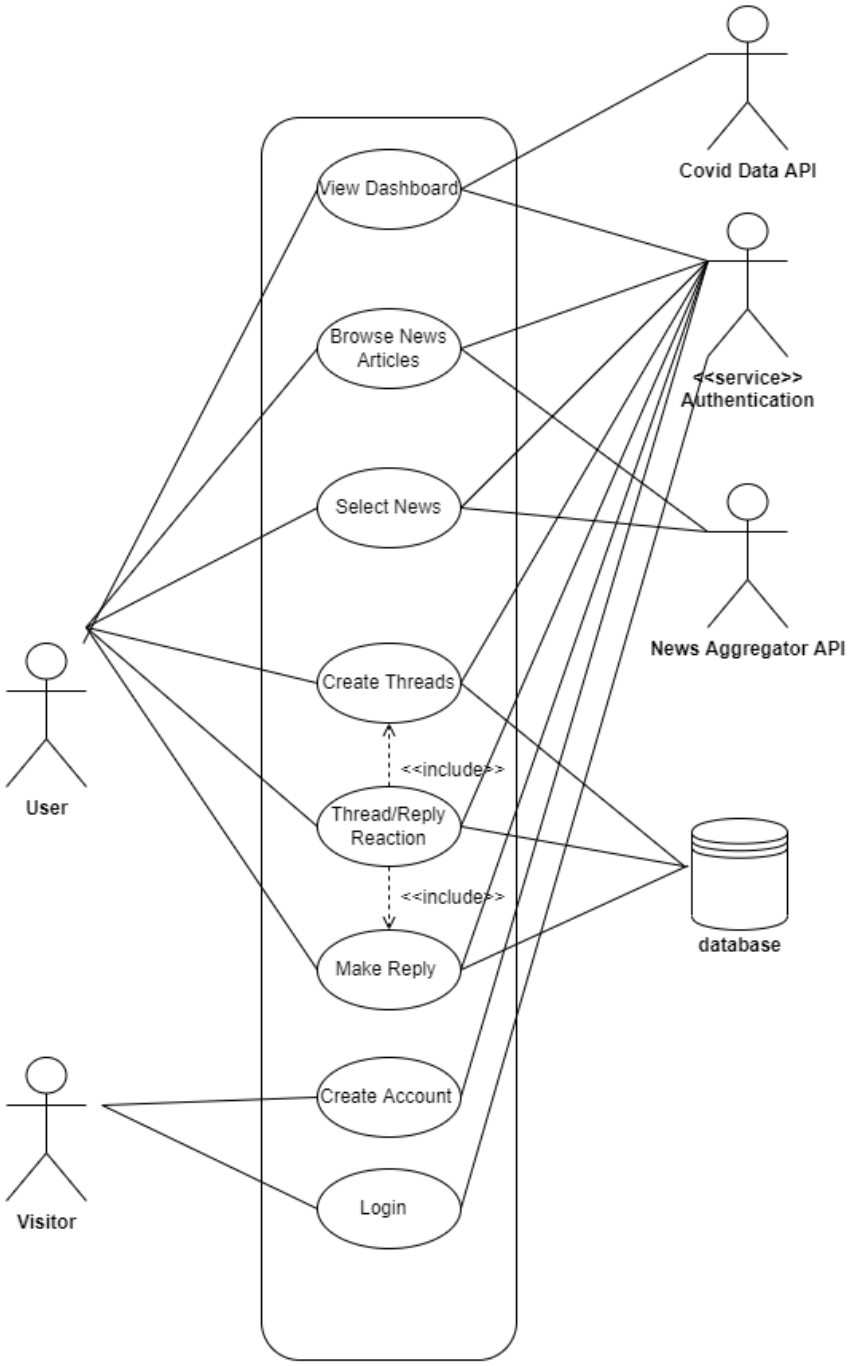
Sequence diagram for the overall forum functionality:

Since the forum has a relatively more complex implementation than other functionalities, we use an informal UML diagram to illustrate the following use case:

A visitor:
1. Register an account
2. Login the newly-created account
3. Browse threads
4. Write multiple new threads
5. View a particular threads
6. Make some comments/replies
7. Log-out



2.3 Specification



We wrap up everything we just touch upon into the Use Case diagrams above.

Use case 1: A visitor creates an account.

A visitor creates an account by entering the username and the password twice. He then clicks the button `Register`

Data	Type	Note
Username	String	Required field, cannot be already existed in the database
Password	String	Required field
Password (repeated)	String	Must match with the password that just entered.

Use case 2: A visitor login with his account.

A visitor can choose to login with his account by entering the username and the correct password. He then can click the button `LOGIN`

Data	Type	Note
Username	String	Required field, must be existed in the database
Password	String	Required field. Must match with the username.

Use case 3: A user browses the dashboard.

When at the dashboard page, a user can perform the following actions:

1. View, zoom in, zoom out and drag across the word map; hover over the country on the map to see the total cases of the country.
2. View the bar chart which ranks the top five countries in terms of covid cases.
3. View the bar chart which ranks the top fives countries in terms of death.

Data	Type	Note
An interactive word map	?	
Bar chart ranked by total cases	Canvas / graphic	
Bar chart ranked by death	Canvas / graphic	

Use cases 4,5: A user browses through the news items and selects the interesting one.

When at the news page, a user can perform the following actions:

1. View the news item, hover over the interested news item and the title will be subsequently displayed.
2. Click on the new item and see the pop-up windows giving a summary of the article and the date of publication.
3. Click the `Resource` button of the pop-up window, and the user will be directed to the external news source.
4. Click the `X` button at the top right of the pop-up window, and the user can close off the pop-up and go on browsing.

Data	Type	Note
cover image of the new item	PNG, JPEG, JPG etc.,	Must be clear and contain enough pixels.
News Title	String	
Date of publication	Date	
News Briefing	Paragraph	
News source	URL	

Use case 6: A user creates a thread.

When at the forum page, a user clicks the button `NEW THREAD` where a window will be popped up. The following actions need to perform in order to create a thread successfully.

1. Input the thread title
2. Input the content.
3. Click the button `Submit`.

Users can opt out of creating a thread by clicking on the `X` in the pop-up window.

Data	Type	Note
Thread title	String	
Thread content	Paragraph	
Date of publication	Date	implicitly registered
Number of likes	Integer	implicitly registered, initialized at 0.
Number of dislikes	Integer	implicitly registered, initialized at 0.
ID	Integer?	Implicitly registered.

Use case 7: A user responds to a thread or a reply by giving reactions ‘like’ or ‘dislike’.

When at the specific thread page, a user can give a ‘like’ or ‘dislike’ reaction.

Use case 8: A user leaves a reply.

When at the specific thread page, a user can reply to a thread directly or further reply to other users’ comments. The following actions *can be* taken:

1. Users choose to expand/collapse the reply sections.
2. Users reply to the inner comments.

After clicking the `Reply` button, a simple edition will be shown where user can drop their reply. Once finished, he can press the `Reply` button.

Users can opt out of making comments by clicking on the `cancel` button.

Data	Type	Note
Reply content	Paragraph	
Date of reply	Date	implicitly registered
Number of likes	Integer	implicitly registered, initialized at 0.
Number of dislikes	Integer	implicitly registered, initialized at 0.
ID	Integer	Implicitly registered.
Reply-to-ID	Integer	Implicitly registered, link to the thread/reply.

3. Design

3.1 Global design

Prepare core build to connect (frontend-backend)

The project is separated into 2 parts: frontend and backend. The core of all interconnections starts with parent/pom.xml file. This is the core file to interconnect the backend and frontend. It packages 2 modules in one go and the modules themselves are in different files.

Frontend frameworks and bundle JavaScript

The frontend module is located at frontend/pom.xml. It mostly contains JavaScript technologies but as we have to include it in the Spring Boot app(backend) we have to bundle it in .jar format and for that, we use 'frontend-maven-plugin'.

Next, package.json should be created to declare development dependencies and other frameworks. The known 'npm' or 'yarn' tool allows installing those dependencies. The 3 main dependencies are charts.css(for charts CSS), leaflet(for the similar map to GoogleMaps) and WebPack. WebPack is configured in its config file in such a way that when it runs it bundles JavaScript file and creates a '.jar' file in a defined folder that is later recognized by the backend.

Backend frameworks and access to JavaScript

The backend module is located at backend/pom.xml. It mostly contains Spring and Hibernate framework dependencies but also it has a dependency for our frontend file. The continuation of this interconnection is at WebMvcConfig.java , where the 2 resource handlers are added. In general, the server will look for a specific classpath, that was defined in our frontend project, so the request by this path will finally return JavaScript code.

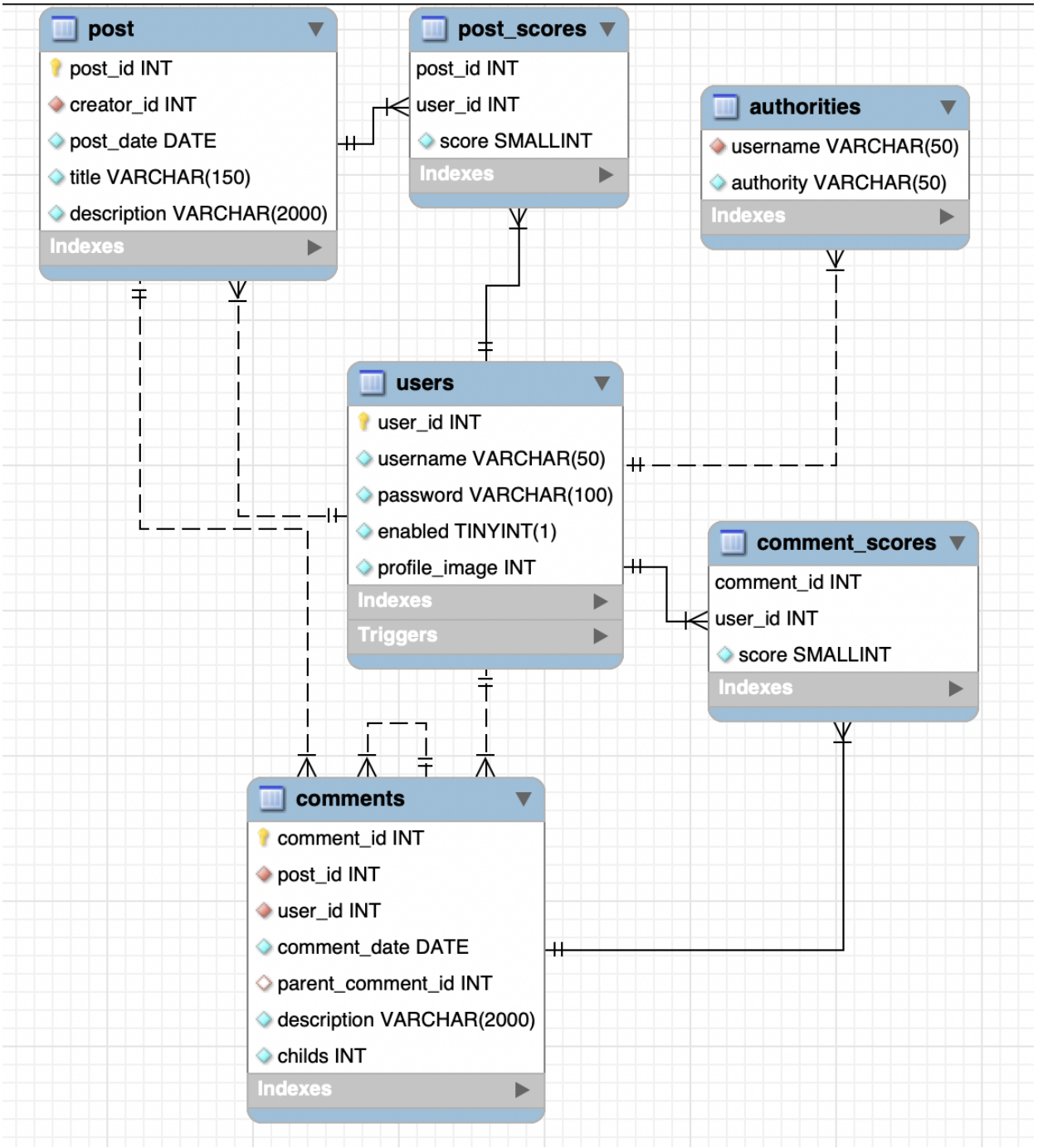
Backend-HTML(Thymeleaf)

After our main interconnections between the backend and frontend are ready, we still have to define the interconnection between the backend and HTML. We use Thymeleaf for that, which is a template engine. The servlet sends the client request to Spring MVC controller, which processes the request and adds model data as the attribute. This attribute is then processed by Thymeleaf into the HTML page.

Finally, the project also has trivial interconnections such as images and CSS files, that are simply connected by their path.

3.2 Detailed design

Database



1. **Users** - We need this table for authorization to the social network. It has user_id, username, password(obviously hashed), enabled(indicated if the account is still valid, which is mainly needed due to Spring Security), profile_image(indicates the id of the profile image. This number is assigned in the following way ‘image${profile_image}.png=’image3.png’’)
   It has ‘one to many’ mapping to authorities(one user can have many authorities)
   It has ‘one to many’ mapping to post(one user can post many posts)
   It has ‘one to many’ mapping to post_scores(one user can grade many posts)
   It has ‘one to many’ mapping to comments(one user can have many comments)
2. **Authorities -** We need this table for authorisation due to Spring Security. It has a foreign key of username. Another column is authority, which in our application is always(ROLE_USER), but in the future development, it can contain something like(ROLE_MODERATOR)
3. **Post** - We need this table to keep track of posts(questions). It has post_id, a foreign key creator_id to Users, post_date, title, and description.
   It has ‘one to many’ mapping to post_scores(one post can have many scores(like/dislike))
4. **PostScores** - We need this table to keep track of post scores. It has dual primary keys of post_id and user_id. And also score column is defined in the following domain -1<=num<=1.
5. **Comment -** We need this table to keep track of comments. Note that it is the most interesting table, as it is done recursively. Remember, our social network looks like Reddit, so it has hierarchical data. It is done thanks to the parent_comment_id column that has ‘one to many’ mapping to the **Comment** itself(recursively). Other columns are comment_id, foreign keys: post_id(keep track of to which post the comments belong), user_id, comment_date, description, childs(number of children that a comment has)
6. **CommentScores** - similar to PostScores but instead of post_id has comment_id.

Frontend

View.js
Can be extended by other classes. It contains common functionality that can be used in different classes. Namely, render loading spinner and inserting elements on-page.

NavView.js
The class is used on the forum, map, news, and discussion pages. It handles the hovering effect of the navbar.

Map page

covidData.js

- The method **loadData()** loads data from localStorage if the data was cached, otherwise call **_getCovidDataAjax(offset)**.
- **_getCovidDataAjax(offset) -** method processes the API call. The offset parameter determines the date of the data to be fetched(data = currentDate-dayOffset). At the start, the offset is 0, but if the API of the current date has not been published yet, then it calls recursively **_getCovidDataAjax(offset+1)**

mapController.js - acts as a connection between mapView.js, navView.js, chartView.js and covidData.js

- It has an **init()** method that creates an instance of navView.js and adds handlers to it.
- Another main idea is that it calls the asynchronous **loadData()** method of covidData.js. After the data is fetched, MapView and ChartView have it.

mapView.js

- The main method is **generateMap(covidData)**, which with the help of the Leaflet framework sets a view of the map and then calls the method **_addMarkerToEachCountry(covidData)** - traverses through covidData, processes it and sets a view of the marker on the map.

chartView.js

- the main method is **renderChart(data, indexOfBody),** which processes data and queries HTML element according to indexOfBody. Moreover, it also calls method **_addChartToSection(key, value, body)**
- **_addChartToSection(key, value, body)** has parameters(key=country title, value=covid cases, body=HTML element). This method creates markup with the help of the charts.css framework and covid data. Then it inserts markup on the page.

News page

newsController.js - the idea is the same, as in mapController.js, but instead of class covidData we simply create the Rest API call(there was no need to create a separate class for it) to fetch data of the news and then add handlers to newsView.

newsView.js

- ❖ main method is **showNews(newsData)** , which iterates newsData and with each iteration calls **_generateMarkupForTable(news, i)**
- ❖ **_generateMarkupForTable(news, i)** generates table HTML markup. In the end, it inserts HTML on-page.
- ❖ All other methods are simply event handlers for design.

Forum

forumView.js and **forumController.js** do not contain any unique and difficult methods. They contain the functionality of event handling to rotate images or press the button.

Discussions

discussionData.js

- static _lastPrimaryKey field keeps track of the primary key of the last comment in the database and acts as an **ID(primary key)** for new comments on the thread. It is assigned after the call of **setInitialData()**, which creates a Rest GET call to the backend(Spring Rest) to fetch data about the last primary key so far.
- static _user field keeps track of the current logged in user. It is also assigned with a Rest GET call to the backend(Spring Rest)

- ❖ method **sendCreateCommentRequest(...)** send Rest POST call to the backend(Spring Rest) to insert a comment to the database
- ❖ method **sendChangeScoreRequest(...)** send Rest POST call to the backend(Spring Rest) to insert a comment/question score(like/dislike) to the database
- ❖ method **loadQuestionData()** send Rest GET to the backend(Spring Rest) to get the question of the current thread.
- ❖ method **loadCommentData(level)** sends Rest GET to the backend(Spring Rest) to get comments on the current thread.
  Where the ‘level’ parameter determines the number of comments to load(we do not want to load immediately all comments of the thread, as it would violate performance. Hence, we load only if the Show Replies button was pressed or when the page is opened. If we would have implemented a system in a way that after the page is loaded all comments(including hidden under replies) are loaded, then the system would become pretty slow if we had to deal with a thread that has 100 comments under each main comment)

discussionController.js

- **init()** method calls discussionData.js methods: **setInitialData()**, **loadQuestionData(level)** methods. Only after **loadQuestionData(level)** we call **processCommentsDataLoad(parentId)**.
- **processCommentsDataLoad(parentId)** - think of 'parentId', as the block(body) of replies.
  If the discussion data of the 'parentId' was already fetched(Rest) for the current replies(replies button was already pressed previously), then return false, because the data was already fetched and we just show it. Otherwise, Rest GET and show these replies.
- **processCommentsDataCreate(...)** - is called when the submit button of the form was pressed. Uses discussionData instance to assemble comment data and to send Rest POST call. Returns assembled comment data

commentRow.js

- Acts as a view of a comment. The markup creation, insertion to HTML, and contains methods of event handling. The constructor has **(parentDiscussion, commentData, isNewInsert)**, where
  - parentDiscussion is also a CommentRow or a QuestionRow, so it is a recursive HTML page. It helps us to define CSS and understand after which element we should insert a new CommentRow.
  - commentData - contains data to create markup and define CSS for it.
  - isNewInsert - Tells a view, if we created a new reply or we just load data from the database.
- Other than that this class only contains complex(recursive and CSS) design conditions for markup and location of the element.

questionRow.js

- It extends the commentRow class, as in our application a question and comment are similar by design. However, it has a little bit different CSS options, so we override **_getRowMarkupDesignOption()**, which simply returns the object of the CSS option. Wee also override the score listener(remember, as the discussion controller has 2 different processes of score creations(for question and comment).

replyFormView.js

- It simply acts as a view for a form, that should insert a new comment.
  An instance of ReplyFormView is created when the Reply button of CommentRow is pressed.
  The submit button of this form creates a new instance of CommentRow.

discussionView.js

- **showQuestion(data)** - assebmles and shows view of QuestionRow
- **showComments(..., data)** - assembles and shows view of CommentRow

Backend

Config

- Class WebMvcConfig - the purpose of this class was already explained in Global design. It acts as an interconnection and adds resource handlers for JavaScript code.
- Class DataSource has method **dataSource()** that configures parameters for our database connection.
- Class SecurityConfig implements predefined WebSecurityConfigurerAdapter. The **configure(HttpSecurity http)** method builds formLogin security parameters, adds csrf protection to Cookies(which is later accessed in JavaScript) and lastly, adds ant matchers for files that do not require authorization.
  configure(AuthenticationManagerBuilder auth) method in this class configures authentication to JDBC for our database.

Entity

- The idea behind all classes in the package ‘entity’ is nearly the same. We use @Entity and @Table annotation to specify our “table” that comes from the database as an entity. Such annotations @Column and @ManyToOne help to specify a connection to the specific column and mapping.
  The most interested one is PostScoreId and CommentScoreId. These classes only specify an id for entities PostScore and CommentScore. It is required to create a unique class for that, as in our database model PostScore and CommentScore have dual primary keys, and simple @Entity can not have them. Hence, we create a @Embeedable class and then use it as @EmbeededId.

Dao

- The package dao contains only interfaces(for example **UserRepository**) that extend predefined JpaRepository. These interfaces act as a repository and contain SQL methods already implemented in JpaRepository or written explicitly with @Query annotation.

Service

- The package service contains all the business logic. The idea behind all of the classes in it is the same, so I will give one example, which is the most important.
  The interface UserService contains all the methods required by businesses, like:
  User findByUsername(String username);
  Moreover, it also extends the predefined UserDetailsService interface that is needed for authentification.
  The class **UserServiceImpl** has @Service annotation, implements **UserService** and has our dao **UserRepository** that is bundled as composition. Then, this class simply overrides **UserService** methods and uses **UserRepository** to implement business logic.

Controller

- The class **RegisterController** contains different mappings for registration and init binder. The annotation @InitBinder identifies methods that initialise **WebDataBinder**, which is used for data binding. In our example, we use it as an editor of String.class to trim all the strings(we do not want to have a username/password with spaces).
  The method with annotation @GetMapping simply shows the registration page and adds data to the Model. This data is a new instance of **PrototypeUser** class, which should be later fulfilled in the HTML form with the help of Thymeleaf. Note that we create **PrototypeUser** and not **User** because the **PrototypeUser** has a different model, that has only a username, password, and matching password fields. We also add javax constraints to this class for validation and use a custom constraint to ensure that password and matching password are equal.
  The method with annotation @PostMapping receives a @ModelAttribute of fulfilled **PrototypeUser** and uses the @Valid constraint to validate it. The validation result is transferred to the **BindingResult** parameter.
  If **BindingResult** has errors or if by using our **UserService** we find in our database a user with the same username, the method will redirect to the registration form again, otherwise, we encode the password field of **PrototypeUser**.
  BCryptPasswordEncoder is initialised in this class with constructor injection. Finally, we save it to the database.
- Another interesting controller is **ForumController** class. It again uses constructor injection to get instances of **PostService**, which we use to invoke findAll() method and then parse the List of Post to the Model data. This list is later processed with the Thymeleaf and shown on the HTML page.
  Moreover, this class also has @Postmapping methods that have a similar idea to the previously explained **RegisterController**, except the fact that we use @RequestParam and not @Modelattribe, because our controller receives Strings(title, description) from the form, and not a model data that was parsed in the @GetMaping. If we used a Model idea similar to RegistrationController, then we would have created only one instance of Post and later use the setter to set the fields, which would spoil immutability and readability.

Rest

- The rest controller DiscussionController is the core of communication between JavaScript and Java during **runtime** not compile. With this class, we want to parse the necessary data from our database to JavaScript. Namely, we use constructor injection for **CommentService**, **PostService**.
- Other than that it just comprises @GetMapping methods for the Rest API.

3.3 Design justification

3.3.1 Global design

At first, we did not consider creating complicated interconnections using pom.xml files with backend and frontend. However, after a while, it turned out that it is inconvenient to work in one project directory and it was decided to separate the frontend and backend. As a result, the project became much cleaner and it was easier to collaborate. Moreover, we think that such a decision is how the real applications in the real environment are developed.

3.3.2 Frontend design

For developing with JavaScript we used the MVC pattern. For example, Model(CovidData), View(mapView), Controller(mapController). This really helped to separate our code and make it cleaner.
The most interesting decision was deciding to create View class, as we found that we were duplicating code. Hence, now some classes extend the functionality of the View class.

Another one is that in the first version commentRow and questionRow were in one class. However, the code was very difficult to read and it was decided to separate them and override some methods in questionRow.
Finally, as JavaScript is not a class-based object-orientated language we did not always use classes, as sometimes it was much easier and cleaner to implement something without them. For example, helper.js or config.js just have export methods and constants without a class.

3.3.3 Backend design

In general, as we used Java we obviously dealt with the MVC pattern; however, we also used Spring, so it is a little bit harder.
We can separate the design if the following way(note that every line title is straightly related to the name of the package where it is located):

1. Entity - entity has an associated table in the database. Each instance of the entity represents a row of the table.
2. DAO - stands for data access object. It provides a CRUD repository for specific entities.
3. Service - intermediate layer for custom business logic and multiple DAO.
4. Controller - A controller is a Spring component that processes the request(Get, Post…). It also usually has services as fields that are injected by dependency.
5. Config - classes that have @Bean methods. The Spring container processes the class and generates Spring Beans to be used in the application.
   @Bean - is an object that is instantiated, assembled, and managed by a Spring Inversion of Control.

For example, if we want to represent all ‘Post’ entities that are in the database, the abstract path of it in the application will be the following:
Database(Entity) → DAO → Service → Controller → View

All that is a very common pattern for Spring Boot applications; hence, there was not any other option for how it could be developed, as otherwise, it would not work sufficiently.

4. Evaluation

First, we would like to say a few words about the possible but slight disadvantage of our design. HTML and CSS files are still in the spring application folder and not frontend. Webpack is able to bundle only javascript, and with our current knowledge, we were not able to find a way to interconnect them in a different folder with the backend. However, it might be that it is not possible at all, as Thymeleaf will give an error at compile-time if the HTML file is not in the resource folder.

Second, we are satisfied with our project. We have not expected to implement so many things, as the project is quite huge. It is a social network, so there is obviously plenty of ways how it can be further developed. For example, add a proper profile page, which should be easy with our quality of design.

Finally, we have learned quite a lot about how to create a real project and significantly improved our collaboration and programming skills.