Final Project

Authors:

Anton Nahhas – 315248708 – [antonna@edu.hac.ac.il](mailto:antonna@edu.hac.ac.il)

Sameer Jbara – 322542408 – [samirjb@edu.hac.ac.il](mailto:samirjb@edu.hac.ac.il)

PART 1: Installations

According to our past knowledge in full stack development, we have already built projects using React as the front-end and Java as the back-end, we chose spring boot for this project in order to complete the tasks of the back-end. Thus, we already have JDK installed in our system, IntelliJ IDE, and git hub desktop (since we already used it to submit other projects). However, for the databases, we only used MySQL, so I had to ask chat-gpt to provide me with the installation process of PostgreSQL. I told the ai engine that I want to build a website using react and spring boot on IntelliJ, and want to use PostgreSQL for handling my databases, what is the best version for such a project?

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

A picture containing text, screenshot, font

Description automatically generated

In addition, for my front-end part we are required to install the npm module, thus in my project we navigate to the front-end directory, on the terminal, and run the command ‘npm install’, we use the command ‘npm start’ to start our react project. Also for the front-end design we use the Bootstrap library which we also have downloaded already, in the HTML files we only need to include the required files of CSS and Bootstrap, as you can see in the code, in order to implement it.

Another program we used that is not required, however very helpful, is Postman, it is an application where you can design a custom HTTP request for your API and receive response content.

PART 2: front-end:

Part 2.1: front-end configuration

As mentioned before, we use the npm library, installed by the command npm install, to start our react application, using the command npm start. The front-end is listening to port 3000. It is automatically configured, however, to reach our react app, use <http://localhost:3000>.

PART 2.2: react programming:

The desired website has two pages: the registration page and the summary page. We will use react routers to create these two pages. Let us start with the registration page:

We asked chat-gpt to create a suitable component for the desired tasks, the desired tasks were taken from the exercise form itself. We also added the description of the website. And got this result:

A screen shot of a computer program

Description automatically generated with low confidence

A screen shot of a computer

Description automatically generated with medium confidence

A screen shot of a computer program

Description automatically generated with low confidence

A picture containing text, screenshot, design

Description automatically generated

A screen shot of a computer program

Description automatically generated with low confidence

We got a page with no design, simply a suitable component that updates using the useState function, all the data the user has provided, and an html page with no design, but a full form as we asked him to do. Now our experience with React comes in handy, we will divide the code into smaller components, each field will have its own component, as well as add a simple design using the bootstrap library, just to make the form look friendly and simple for the user. In addition, we will add a description box so the user can understand why he is filling out the form and the purpose of the website itself. All of these changes could be seen in the repo.

As we usually do, we first draw a theoretical grid of our page,

A picture containing text, handwriting

Description automatically generated

This helps us divide the rows and cols for each html div element. This is how the website looks now:

A screenshot of a computer

Description automatically generated

We asked chat-gpt to remind me how can I divide my code into more components and implement lifting up states as well, here is his answer, which helped us to accomplish the wanted task.

A screenshot of a computer program

Description automatically generated with medium confidence

A screen shot of a computer

Description automatically generated with medium confidence

For the cities drop down, I have downloaded this Json file from this website:

<https://simplemaps.com/data/il-cities>

and asked chat-gpt how can I add this list of cities to my project here is the response:

A picture containing text, electronics, screenshot, software

Description automatically generated

A screenshot of a computer program

Description automatically generated with medium confidence

Same thing is done for the previous conditions input, since we want to prevent hardcoding strings and variables, thus we create a more generic way, which will be easier to add/remove conditions/cities.

Now we conclude our prototype for the first page, there might be changes afterward but for now that is it for the prototype, before we move on, let’s test if it works. Let’s fill up the fields and print the object created.

A picture containing text, screenshot, font

Description automatically generated

Yup, it works, onto the second page.

Dually, we asked chat-gpt to create a new component for the second page, We provided the required tasks, and asked him to add an info message informing the user that the table is empty when it is actually empty, if not the message disappears. Here are the results.

A screen shot of a computer program

Description automatically generated with low confidence

A screenshot of a computer screen

Description automatically generated with low confidence

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

Now it is our time to add our final touches using bootstrap. The page only consists of a table so there won’t be a complicated grid as the first page.

Now there are no more things to do since we have not implemented the back-end yet. We don’t have the data; thus we won’t see a table and be able to give it some design or fix its issues.

PART 3: Back-end

PART3.1: configuration:

We asked chat-gpt how can I create a configuration that listens to port 8080 and act as the back-end for the front-end we just wrote, which will also require us to write suitable fetches on the client side.

A screenshot of a computer program

Description automatically generated with low confidence

A screenshot of a computer program

Description automatically generated with low confidence

PART 3.2: back-end programming:

Now we asked chat-gpt to help us complete step four and five from above, firstly the User entity:  
A screenshot of a computer screen

Description automatically generated with medium confidence

A screen shot of a computer program

Description automatically generated with low confidence

We had to add all the other fields as members, and obviously create setters and getters for each one.

We then asked chat-gpt to write us a suitable userRepository and a UserService file and this was the result.

User Repository:

A screenshot of a computer program

Description automatically generated with low confidence

User Service:

A picture containing text, electronics, screenshot, software

Description automatically generated

Finally, we will ask him to write the User controller where initially it will have only two routes one for adding a new user registry and the other to get all of the users from the database.

A screenshot of a computer screen

Description automatically generated with medium confidence

According to our knowledge since our react project is running on a different domain we have to add the cross origin annotation that allows the client side to interact with the server side. Let us now go back to the client side to send the suitable HTTP post request, I asked chat-gpt to do so for me:

A picture containing text, screenshot, software, display

Description automatically generated

Again according to our knowledge, it is not recommended to have you server side port in the client-side code, thus we changed our routes to be handled using the /api route. Now, we asked chat-gpt to show us how can we run the PostgreSQL server, fetch the data and check our results.   
A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with low confidence

Since we don’t have the best knowledge in PostgreSQL, we had to ask him again how to verify the data exactly and got this answer:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

And we indeed tried this method to verify if the data is added to the database,

A black screen with white text

Description automatically generated with low confidence

As we can see all our inputs are saved to the database. Now let us add some sql constraints using the valid annotation, again we ask chat-gpt to do so.

A screen shot of a computer program

Description automatically generated with low confidence

A screen shot of a computer program

Description automatically generated with medium confidence

Now we asked chat-gpt to handle the errors using bindingResult object and the @valid annotation in the post request endpoint:

A screen shot of a computer program

Description automatically generated with medium confidence

We still have to handle the errors in our react, this task was done by us, we basically used the @valid annotation and the binding result object to return a JSON object where the key is the field name and the value is the error message, in addition, we added error handlings for each relevant field to display the messages to the user. However I also chat-gpt to provide me even more constraints using @pattern annotation that includes a suitable regex for some fields, such as landline and cellular phone numbers cannot include letters, on the other hand, first and last names cannot have letters or special characters, etc…

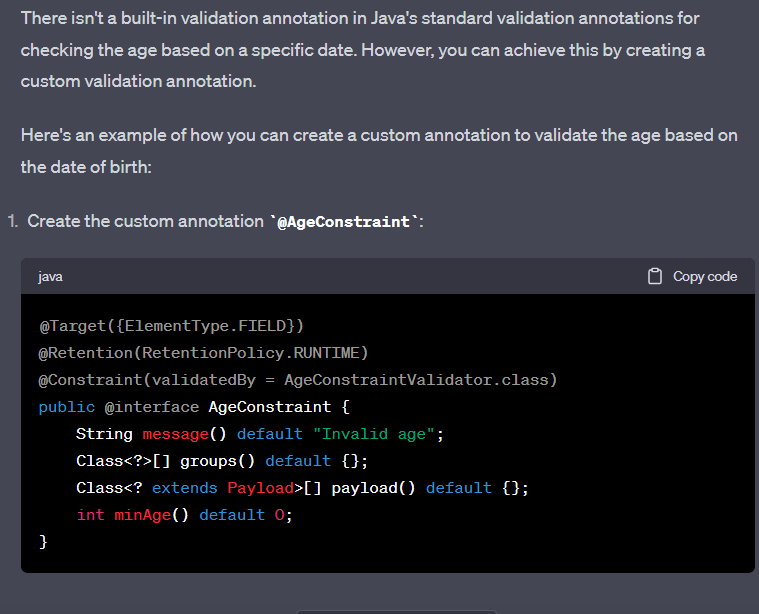
A picture containing text, screenshot, software, font

Description automatically generated

A picture containing text, screenshot, font

Description automatically generated

Another constraint we want to add, is to check for a valid date of birth, basically we want the user to be at least 16 years old to fill out the survey, we know that there is not a built-in validation annotation where it checks the required task, so we asked chat-gpt to create a custom validator for us to do so:



A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

Now back to front-end again, we ask chat-gpt to create a suitable get request fetch from the summary page to our /api/users route, and add all the necessary fields:   
A picture containing text, screenshot, software, multimedia software

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

As you can see it did not provide all of the fields, so I had to add the others by myself, now the second page looks like this

A screenshot of a computer

Description automatically generated with low confidence

We still got work to do, we need to add a search bar option that allows the user to search either by city name or by date range. Who else other than chat-gpt can help us achieve such a task? Firstly, we asked him to implement the search by city filter, we provided that we have a service, repo, and a controller class, as well as, the summary component, and how would they look after implementing the feature…   
A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A computer screen shot of a program code

Description automatically generated

A screenshot of a computer program

Description automatically generated

And it works amazingly, however I noticed that the user cant retrieve all of the data again after his first search, only by navigating to the register page and back to the summary page, which is obviously not comfortable for the user at all, thus we asked chat gpt to refactor our component so that it checks for empty queries so that if the user search on an empty query he can retrieve all of the data. This is the updates we had to undergo   
A computer screen shot of a program

Description automatically generated

A screen shot of a computer program

Description automatically generated

Now the feature works fine let’s move to the second method the user can search with which is a date range, so I asked chat gpt to do the same but with a date range.   
A screen shot of a computer

Description automatically generated

A computer screen shot of a program code

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

Now we add our own touch of simple design, again using bootstrap. We simply divided both search options into a grid where each one of them will have a col-6 className, this will make the fields next to each other, we gave them as well a border with slight color to differ the brightness. In addition, we created a new button that cancels the search, meaning it will fetch all the users again.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Now that concludes the functionality of our website, everything works, design is simple and average, added a loading gif when fetching data, and everything looks great but the code itself is a mess, now we have to go through the code, add appropriate error handlers, instead of the ones that chat-gpt provided where it simply prints the error to the console and crashes the site, instead we want to create for each page, an error page that receives the error message if any occurs, and displays it to the user. In addition, divide duplicate codes into functions, and add appropriate Java and JavaScript documentation, for our other fellow coders to read and understand what was done.

Resources: -

Note that the screenshots from chat-gpt are the main or the working code snippets it provided. For the full detailed conversation, we had with him:

<https://chat.openai.com/share/7c514bd4-9a06-4c40-80f2-1cd09f2bba72>

for bootstrap documentation that we used a lot for building and designing the form, buttons, and summary table here are the full docs:

<https://getbootstrap.com/docs/5.3/getting-started/introduction/>

for the full code, it is found in this GIT repository:   
https://github.com/anton11nahhas/website-built-by-ai