Summary

Currently, there is no active website showing the Bioinformatics study computers' activity. Previously a website existed that showed which computers where online or offline, but since a while it stopped working and its replacement could have some extra features. That's why our team of student developers where tasked to provide exactly that. The advantage of such website is giving a quick overview what computers can be logged into using either ssh or in the classroom by students and teachers alike, without having to manually find out by trial and error.

To realise this a web.xml is needed with requests, which are links that inquire data e.g. computer workload from a server. Then these queries are fetched by the XmlWebListener and a list of these request is passed to the RequestListener. The RequestListener then starts making requests using the MakeRequest class. Once, those are made the extracted data is assigned to a structured Workstation class. Finally, we convert that structure to JSON type objects.

Then from the back-end these objects can be fetched by the middleware which is AJAX in JavaScript. This data is then connected to a configure JSON file, which is used for creating a computer grid or classroom map that can be inserted into the front-end; the HTML pages.

By following this methodology, the website results in two main featured pages. A front page with all the computers showing their online status, but also a button which shows more information like the workload or processor temperature. And, a second page with classroom maps based on the actual classrooms in the bioinformatics space at the ILST institute. Here the online status is also viewable, but more simple. Additionally, the website works both on computer as on phone.

Ultimately, a working and flexible website is created. In the future it would make a good addition if a feature is added that shows what classrooms are occupied based on the class schedule.

Materials & methods

Materials

This project is located in github. In the directory *java/nl/bioinf/shbreekers* is all the Java (17.0.8) code. And, in *webapp* all the JavaScript (1.5), HTML5, CSS3, Bootstrap (5.3.2) and Thymeleaf (3.0.15).

First of all, the contents of *plugins{}* and *dependencies* from the build.gradle file are important for compiling the project properly and having the right dependencies; these are needed for being able to use software's like Thymeleaf or making server requests.

Secondly, the project is structured with a back-end, middleware; translates back-end data to front-end, and a front-end. The back-end is Java. Here all requests are made to the server for extracting computer status data. Apache's *httpclient:4.5.14* is needed for making the requests. Next, *gson:2.10.1* is used to convert the extracted server data to JSON objects; structured objects with keys and values. In order to use servlets, *javax.servlet-api:4.0.0* is needed. These previously mentioned dependencies are also included in build.gradle.

In the middleware JavaScript is used for fetching the JSON objects that hold the data and implement that data into objects which are then passed to the front-end.

The front-end includes HTML, CSS, Bootstrap and Thymeleaf. JavaScripts are called in the front-end by adding the scripts at the bottom of HTML files. See the following example:

```
<script type="application/javascript" src="js/createWorkStationCards.js"></script>
```

Bootstrap is added in a similar manner to template.html (the template file) in the following example:

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet"
```

Ultimately all the styling is managed through JavaScript, CSS and Bootstrap; a framework for styling HTML objects more flexibly. And, the templating is managed through *thymeleaf:3.0.15.RELEASE* which is included in build.gradle.

Existing Methods

The status data will be read from a web server through AJAX (Asynchronous JavaScript And XML) calls in JS (JavaScript). AJAX is a technique that allows the user to access a webserver from the webpage.

This project is layered with a back-end powered by AJAX calls. AJAX calls in JS (JavaScript) and enables reading pc status data from the web server; W3Schools.

For styling Bootstrap enables editing of div objects more flexible than css. Meaningful keywords can be added to the class of an object to change its behaviour. For example adding "w-100" in the class of a div, causes that div to widen fully. Although, using css simpler appearance changes are managed.

Furthermore, for templating which reduces duplicated code Thymeleaf connects a template page with the functional pages through the use of fragments. These fragments will contain only basic objects like a header, navigation bar or footer.

Applied Methods

Back-end

Structures are built using JSON (config.json) to configure mapping of rooms and its computers, and web.xml holds the queries links that are needed to query (ask) the desired data from the server.

Config.json is divided in rooms by name. Each room holds a collection of pc names, that are compared to the names fetched from the server. Additionally, a classroom matrix is made to save the actual computer locations.

In the back-end "XmlWebListener" manages passing the queries from web.xml to the "RequestListener". These query links are then passed to "MakeRequest". MakeRequest sends the actual request for each query and the body of each response. At last RequestListener then saves each response as a "Workstation" object where all data is divided into variables. Then these object are converted to JSON objects using Gson().toJson.

Middleware

Request.js has async functions that handles responses to fetch JSON data from requestListener and the map configuration from config.json. Here all JSON data is sent to the front-end where html objects assigned to that data are updated based on the newly fetched information every two seconds.

The data is divided over three div objects that are created and filled by the logic of three javascript files; a file that creates a room map (CreateMapOfRooms.js), a file that creates suggestions what pc's have the lightest workload (createSuggestionCards.js) and a file that creates workstation cards (CreateWorkStationCards.js).

Each workstation card is a block that colours red of green based on the status of the computer. For more statistics of a computer the "Show status" button can be pressed and a model is created with several parameters and a load graph. The graph shows the work load in percentages of the last ten minutes; i.e. every minute a load is visualized as a point and a line connects all the dots over the ten-minute range.

The module "chart.js" is used for the visualization and is added in "index.html" with the same logic as normal JS scripts. The actual graph is created in "request.js" in the "createGraph" function.

Above the workstation cards a plain is filled with computer suggestions. A suggestion card is clickable and will start a scroll event to the workstation card. The scroll event moves towards the calculated position of the card and will end when the card is in the middle of the screen. Then the card's background colour blinks in orange for three seconds, so the user knows where to look at.

All workstation cards and their suggestions are inserted into index.html in a fluid div container called innerdiv. The advantage of fluid is more flexible positioning against the page size, also when the div is filled the contents move along more swiftly. Also, the workstation cards can be filtered by checking a checkbox by room name. The checkboxes are also made using bootstrap.

The logic for making cards appears on-click or disappear is managed in main.js. Furthermore, in main.js is the logic for retrieving the pc names from the config file, inserting the created workstation cards into the "innerdiv" grid and the call is made to update the page content. When scrolling page down a button appears with a scroll page up event if the user wants to return to the top of the page. This event is managed through a 'click' eventListener in main.js.

Front-end

In each page the lay-out of the functional div objects like "innerdiv", "mapdiv" or "menubar" is handled through bootstrap. The basic website lay-out and bootstrap connection, is handled in template.html using Thymeleaf. By adding *th:fragments* to the template file, blocks of objects are created that can be added to other pages. This can be done either with *th:replace* or *th:insert* which depends on, if you want to change to contents of objects on a page or only want to add objects to a page.

Next the user can navigate through the "menubar" in template.html to the room map, home or about page. The linking is managed with Thymeleaf links to other pages. A link consists of the urlPattern of the servlet* that loads the corresponding page. For example, the servlet of the about page has an urlpattern "/about". The appearance of the nav is managed in sidebar.css. *servlets serve for getting, posting data from and to webpages, and processing the webpages.

In Map.html two logics are applied: a button menu to change the classroom and a fluid div container called mapdiv where the created map cards are inserted. By extracting the classroom matrix from config.json map card objects are created for each index that holds a computer name. Additionally, for indices with no computer name empty map cards are made. By applying these logics a classroom map is created from a configuration matrix.

Two images were added to the template page, one favicon and the Hanze icon as image inside the menubar. Note that an icon and image are not the same type of objects; one is and one is .

In order to make the webpage responsive to window size for both desktop and phone. Two solutions are introduced. First by making the div containers "innerdiv" and "mapdiv" container-fluid. This means the contents and the objects shape changes more flexible. For "innerdiv" this works for all window sizes. Although this isn't a solution for the map due its unique sizes. Using breakpoints the viewport is changed for the map, making it decrease in size along with a smaller window. This is works solely on CSS code in "viewport.css".