Web applications, Responsive Websites, Mobile applications, IP telephony, Custom-mapping services.

testapplication.eu

Admin/client web application for collecting and viewing different log entries

Project carried out by Gaspar Clavell as part of the Web Application Development internship

April 27th, 2020

Maribor, Slovenia

MENTOR: David Fenko

INDEX

1. CONTEXT **3**

2. INVOLVED TECHNOLOGIES **6**

3. MAIN FILES OF THE APPLICATION **6**

4. MAIN SECTIONS OF THE APPLICATION **8**

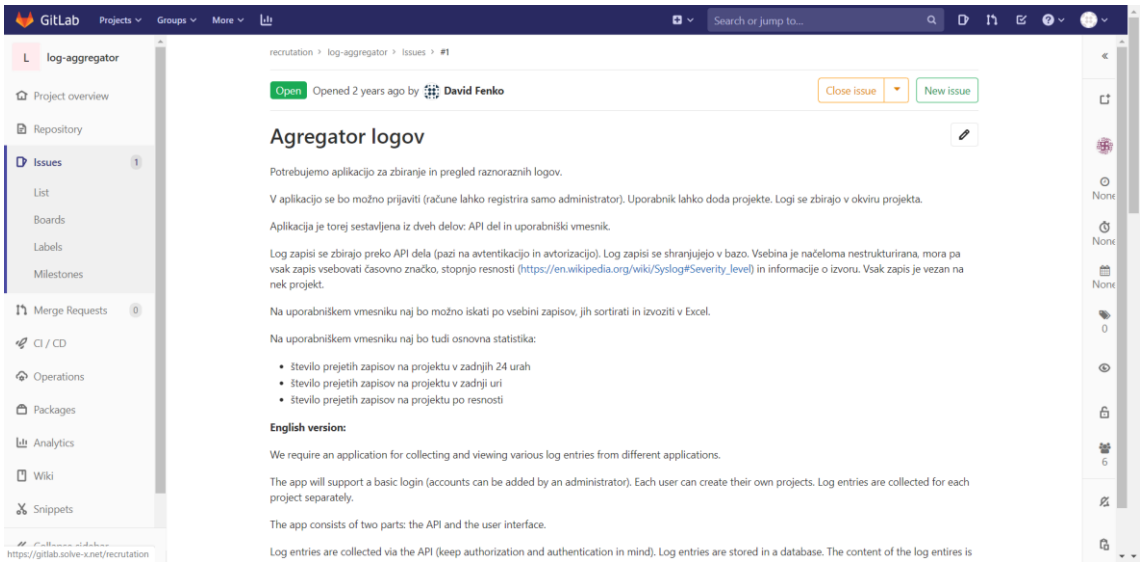
5. NEXT POSSIBLE STEPS **12**

6. USEFUL LINKS AND CREDENTIALS **13**

7. NOTES **14**

1. CONTEXT

The starting point of this project begins with the “Aggregator” exercise, proposed in early March, right at the beginning of the internship:

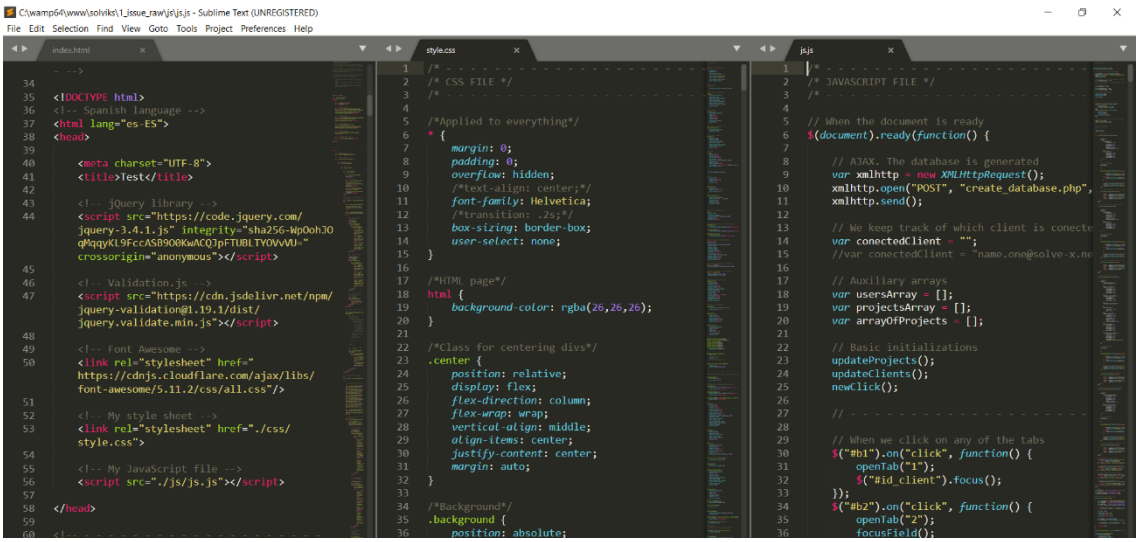


This practice focuses on some of the main features across web applications, such as:

- Admin and client log in.
- Creation and removal of client accounts on the part of the admin.
- Content generation on the part of the client.
- Displaying of both client profiles and their generated content at admin interfaces.
- Every user interaction that implies data management must be reflected in a server-side database.

The premise was simple: make it work the way you know.

1.1. First approach

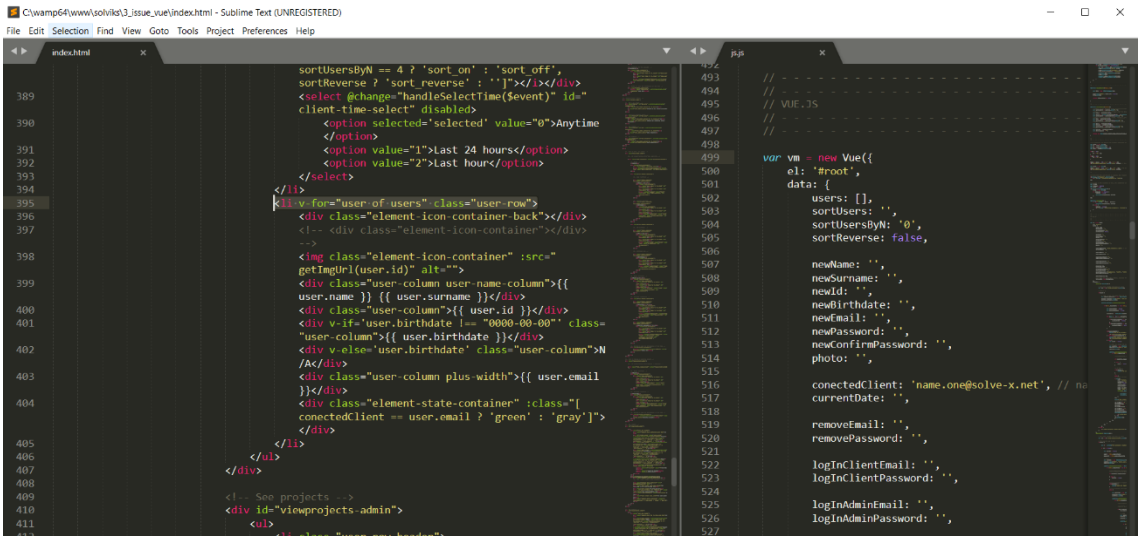


The first approach revolved around raw web development technologies: HTML, CSS, JavaScript, PHP and SQL. Using JQuery (DOM manipulation), AJAX (asynchronous data exchange with the server) and Validate.js (form validation) as additional resources.

1.2. Second approach: Vue.js CDN

Following its submission, I had the chance to take part either at backend or frontend development. Since I chose frontend, the next exercise consisted of rebuilding the application using the JavaScript framework with which Solve-X works: Vue.js.

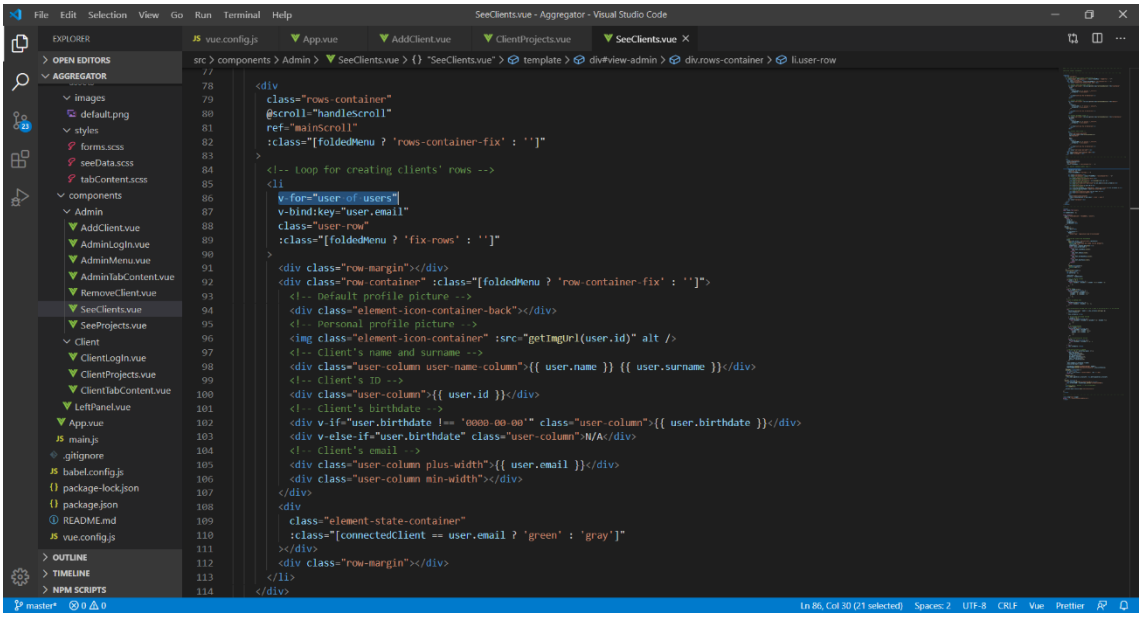
```
<!-- Vue.js -->
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```



After a couple of weeks of studying, I got working a first rebuilt application. Nevertheless, Vue.js was implemented importing it as a script (CDN), JavaScript was still in a single file and JQuery was present all over the code.

1.3. Third approach: Vue.js CLI

I reported the progress made, along with the former set of insufficiencies. So I was given green light to keep exploring the framework.



This time I intended to handle Vue.js using CLI. To do so, I installed Visual Studio Code as my IDE, allowing me to use NPM in its terminal as a quick package manager. Among the installed resources, we have:

- **Font Awesome**: icons.
- **Vuelidate**: form validation.
- **Axios**: HTTP requests.
- **FormData**: file uploading to the server.
- **Sass-loader**: SCSS file handling.

The code arrangement changed drastically, being now a hierarchical structure based on components. JQuery is gone, so the way we manipulate the DOM is entirely different, as well as the manner in which variables and methods are accessed (we must stablish communication among components). Regarding to server communication, AJAX works through Axios instead of XMLHttpRequest(), which involved the use of two servers during the production mode: one for running the Vue app (port 8080), and the other for PHP files and uploads (port 80).

In addition, and seeking to keep playing around with Vue.js features, I developed new app functionalities that strengthen the interface as well as the user experience.

After another month of work, this new rebuild was ready.

2. INVOLVED TECHNOLOGIES

- **HTML5**: texts and structure of the application.
- **CSS3**: style, minor visual effects and responsive design (Flexbox).
- **JavaScript**: behavior, variables-involving operations, local storage and AJAX.
- **Vue.js**: framework oriented to build user interfaces and single-page applications. Its component-based structure allows us to extend basic HTML elements to encapsulate reusable code, and its reactivity system provides a powerful way to dynamically create, remove and update elements depending on their data sources.
- **PHP**: interaction with the server and its database.
- **SQL**: typing, deletion, modification, and consult operations in the database.

3. MAIN FILES OF THE APPLICATION

a. VUE COMPONENTS (HTML + CSS + JS)

Admin:

- **AddClient.vue**: form for adding new clients.
- **AdminLogIn.vue**: form for logging as an admin.
- **AdminMenu.vue**: admin options.
- **AdminTabContent.vue**: admin section container.
- **RemoveClient.vue**: form for removing existing clients.
- **SeeClients.vue**: chart with clients' information.
- **SeeProjects.vue**: chart with information about clients' projects.

Client:

- **ClientLogIn.vue**: form for logging as a client.
- **ClientProjects.vue**: section where connected clients manage their projects.
- **ClientTabContent.vue**: client section container.

- **LeftPanel.vue**: left panel containing client and admin tabs.
- **App.vue**: main project file.

b. CSS/SCSS

- **form.scss**: style sheet shared by forms.
- **seeData.scss**: style sheet shared by charts.
- **tabContent.scss**: style sheet shared by client and admin containers.

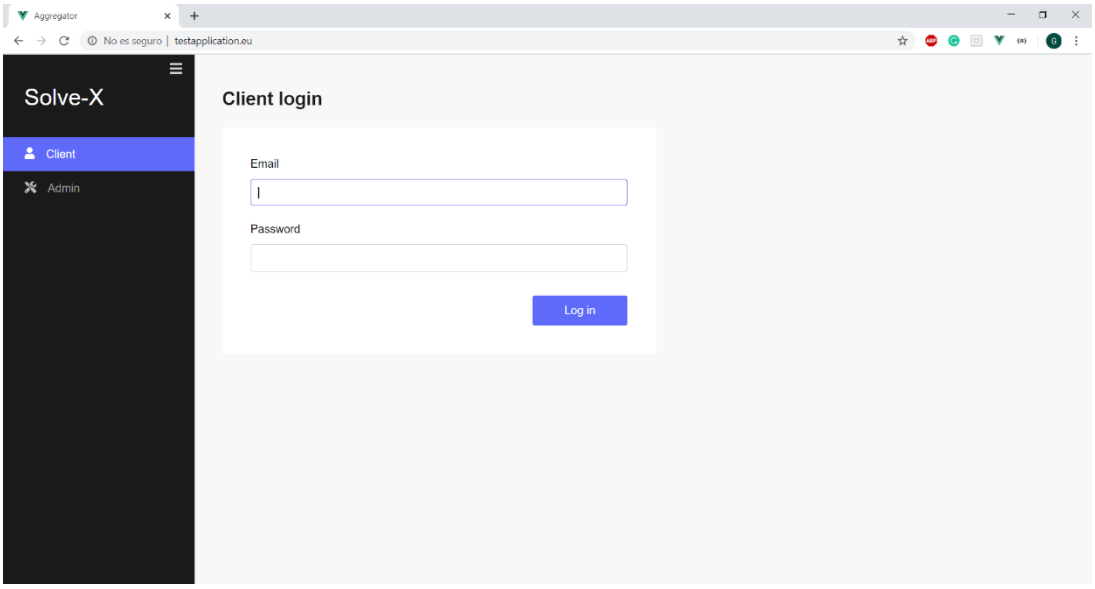
c. PHP

- **add_project.php**: uploads new projects into the database.
- **add_user.php**: uploads new clients into the database.
- **check_admin.php**: checks if an admin account exists.
- **check_user.php**: checks if a client account exists.
- **create_database.php**: creates the database and some example entries.
- **get_projects.php**: gets clients' projects.
- **get_users.php**: gets clients' information.
- **remove_projects.php**: removes a project from the database.
- **remove_user.php**: removes a client from the database.
- **remove_user_projects.php**: removes all the projects from a client.
- **rename_project.php**: renames a project.
- **update_project.php**: updates the severity level of a project.
- **upload_photo.php**: uploads a profile picture named with its client's ID.
- **upload_temp_photo.php**: uploads a temporary client profile picture.
- **variables.php**: stores common variables and methods.

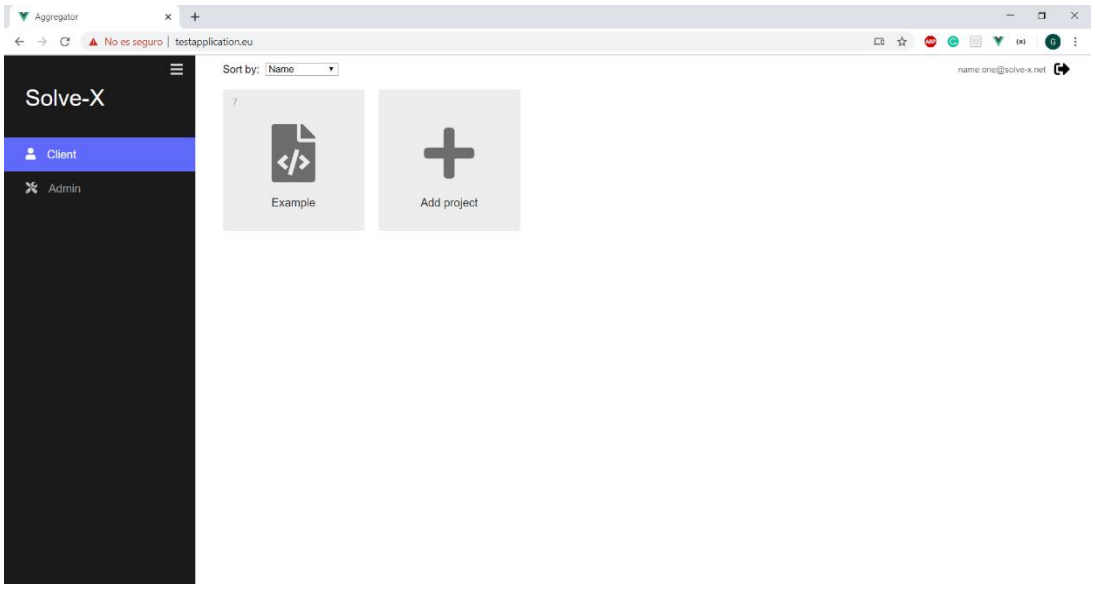
4. MAIN SECTIONS OF THE APPLICATION

Note: the database, its tables and some example entries are created (if they do not exist) every time we refresh the application. This way we can freely test renaming and deleting functionalities.

4.1. Client

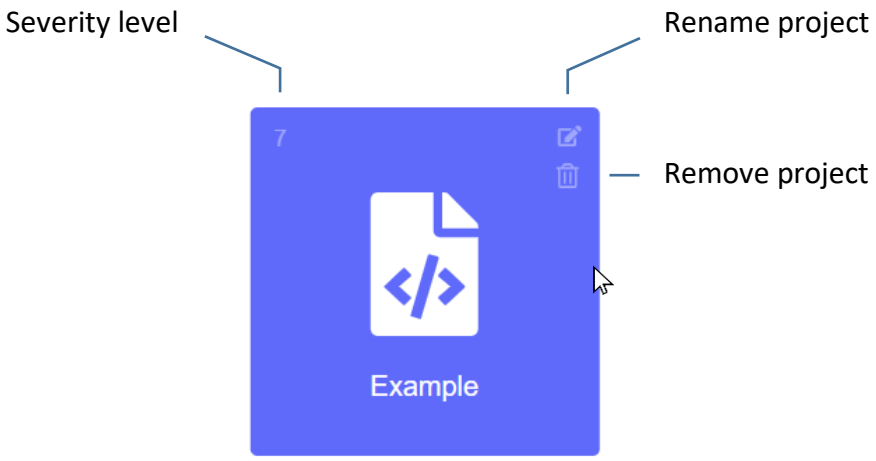


**Client log in.*

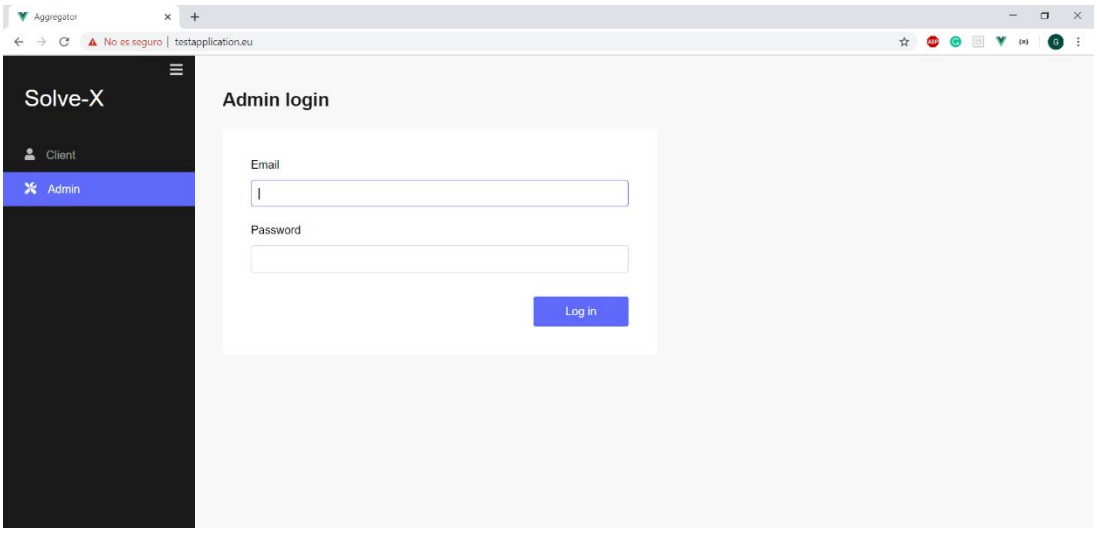


**Connected-client projects manager.*

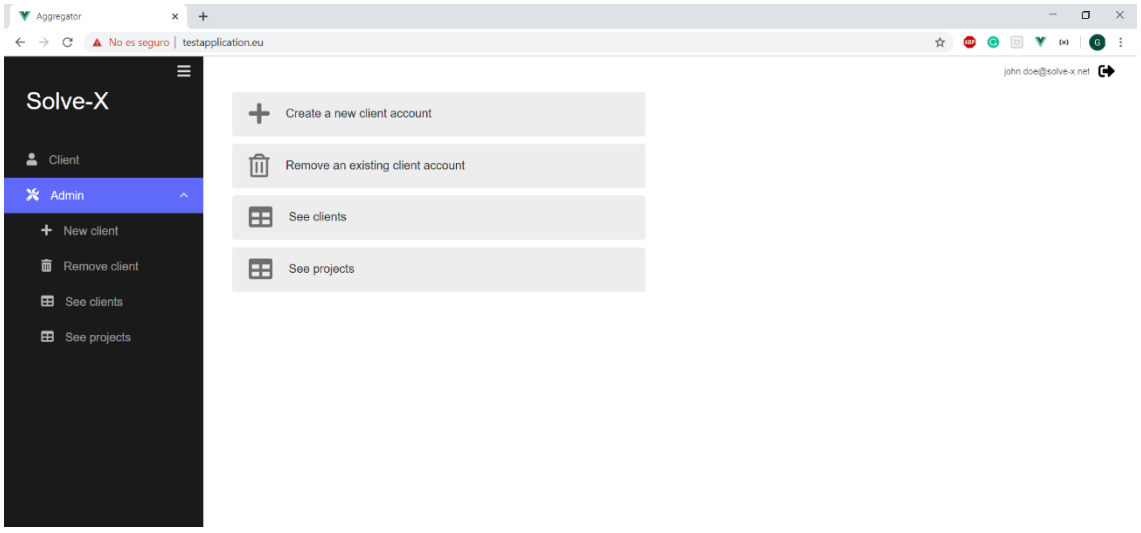
Whenever we modify any data related to a project (creation, removal, new severity-level selection or project renaming), it will be instantaneously updated into the database.



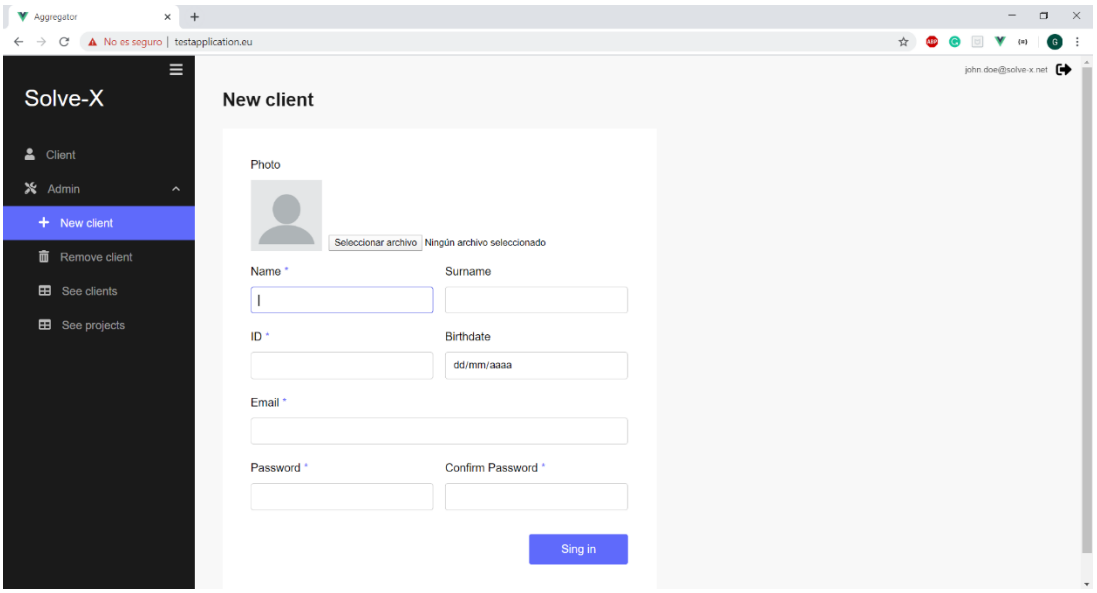
4.2. Admin



**Admin log in.*



**Admin options.*

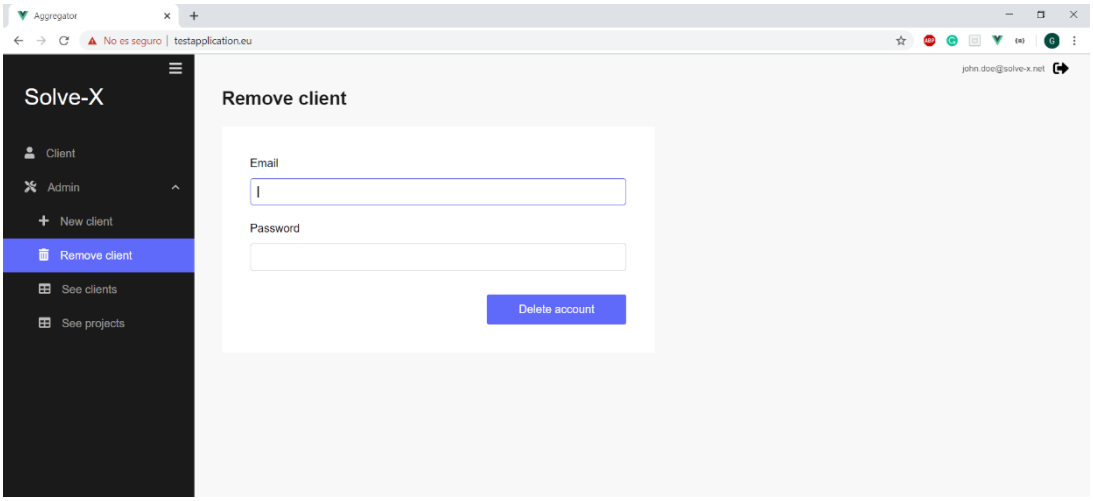


**Form for adding a new client.*

Once the validation from "new client" is complete, the application checks if the submitted ID and email are not already stored in the database; if so, the creation of the account stops. Regarding to the ID, it does not follow any national pattern; it is just composed of nine numbers, so it may be changed to any actual pattern in the future.

Every time we select a profile picture, it is temporarily kept in the server, and as soon as the form is successfully submitted, it is renamed to its client's ID and takes its definitive location.

The minimum age to register is 14 years old.



**Form for removing an existing client.*

Emphasis has been placed on every form so the corresponding text field gains focus automatically, prioritizing non-filled fields and mistaken ones.

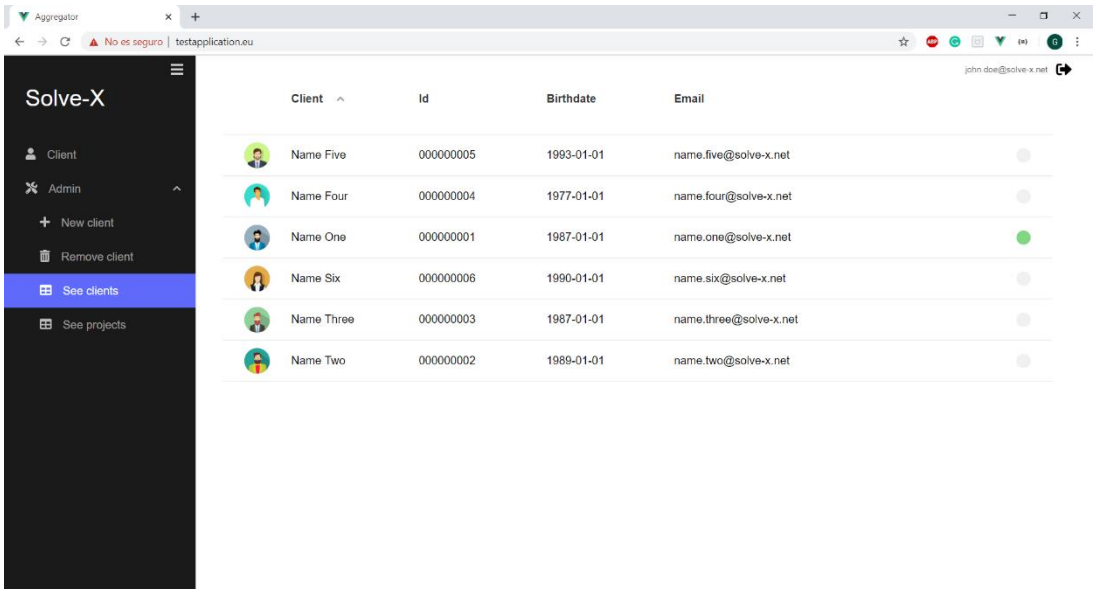


**Chart with clients' information.*

Loaded data undergoes some minor adjustments when displaying. For example: Users' names and surnames are merged in one field and Title Case is applied to them.

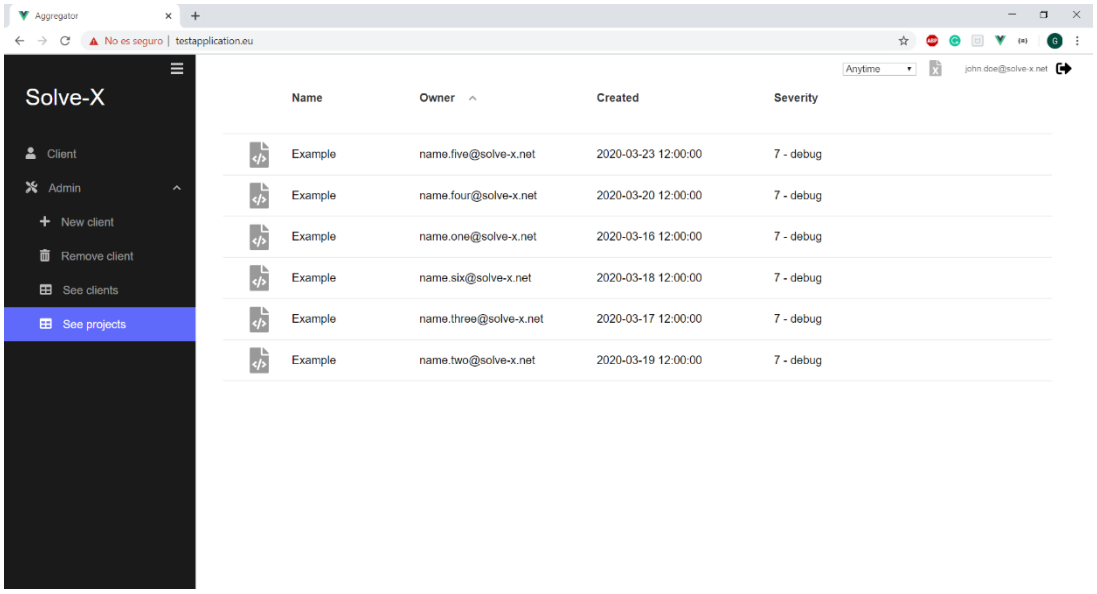


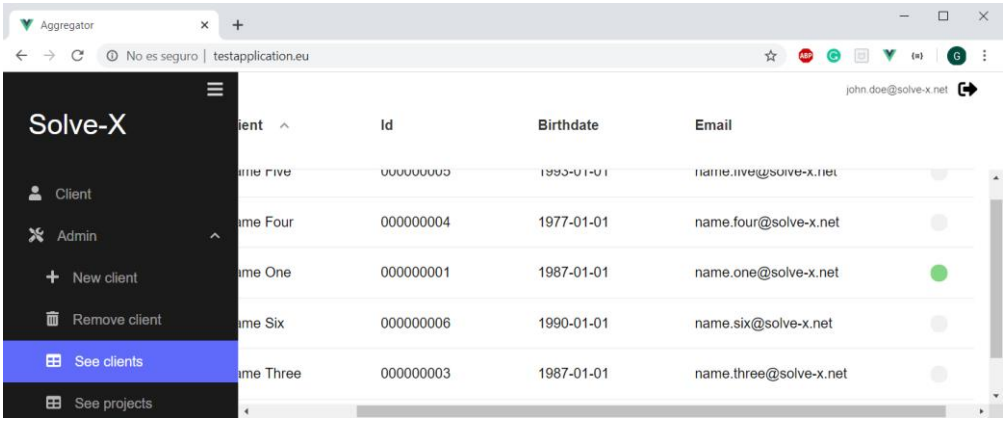
**Chart with information about clients' projects.*

"See clients" and "See projects" only load data when mounting. This way we can manipulate their display instantaneously, with no reloading involved.

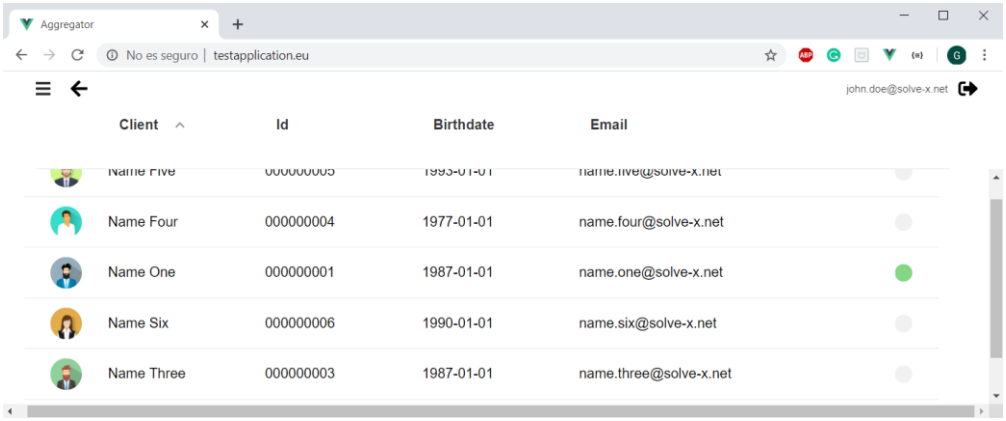
These components mimic the way Windows file explorer displays details, so that:

- We can sort alphabetically data, either ascending either descending way.
- In case we choose a non-key field and it has repeated entries, the key field will work as a second sorting criterion.

- The titles of the fields remain fixed when vertically scrolling and move along with the entries when horizontally done.



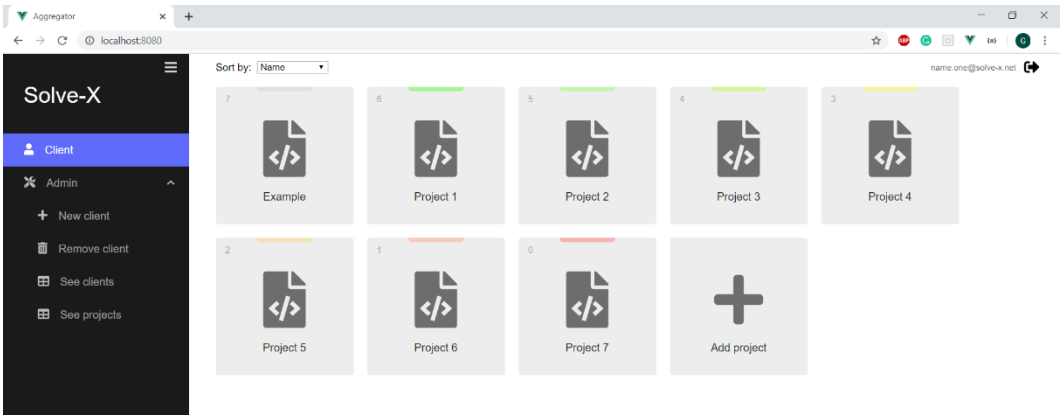
**Detail of a minimized chart. In addition to the fixed field titles, the scroll automatically resizes and relocates after folding/unfolding the left panel.*



**This is a shared property among tab components; however, charts are the trickiest and most eye-catching ones regarding that behavior.*

5. NEXT POSSIBLE STEPS

- To create a responsive design for mobile phone and tablet (Media Queries). For now, the application only displays properly in computers.
- Include TypeScript.
- To make it possible to connect multiple clients (Sockets) so that admins can see them at “See clients” component. Clients’ sessions should be set in such a way that their credentials kept stored even when closing the application (meanwhile they will appear disconnected for the admin), so they would automatically reconnect when reopening the application (until the client decides to log out).
- To add a searching bar at the top of the charts, either “See projects” either “See clients".
- To be able both to modify and save (LocalStorage) the width of the fields of the charts.
- To add two new fields to “clients” database table: “created” and “last connection”, making it possible to filter clients at “See clients” by their last connection (add a drop-down list).
- To make projects editable, so they store some sort of data with which the client can interact with.
- Implement some kind of visual feedback to indicate the severity level of the projects. Some functional tests have been carried out:



Nonetheless, the interface got too saturated when hovering projects, so I need to find another solution. In addition to that, a color scale should be adapted to be displayed at “See projects” component, next to “severity level” fields.

- Create a new admin option: “Statistics”. So that data coming from charts can be analyzed.
- Create a new client section: “My profile”. This way, a connected client could modify its personal data stored in the database (name, email etc.).

6. USEFUL LINKS AND CREDENTIALS

- **Website**

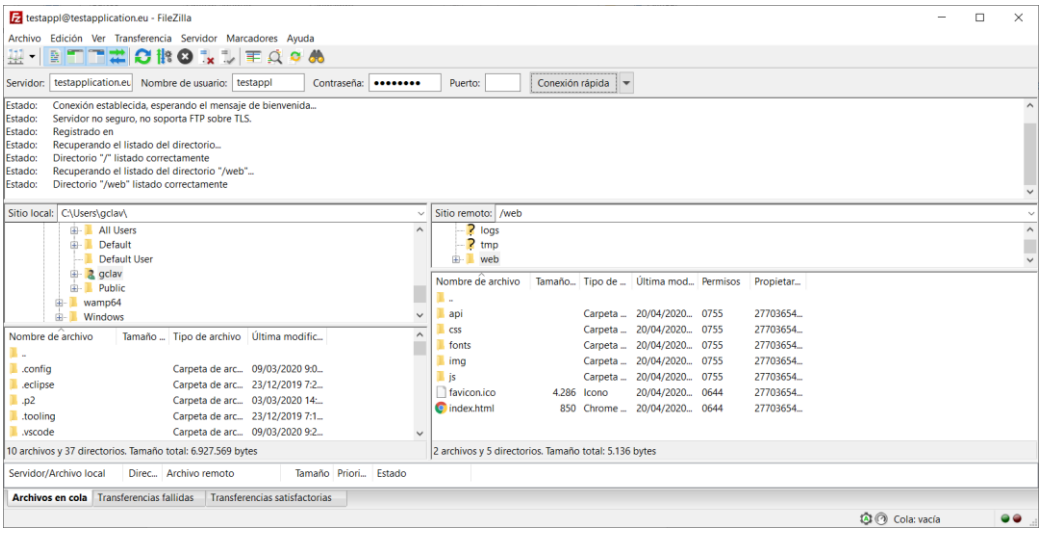
  http://testapplication.eu/

  Username: testap244
  Password: NIAGTgW4

Admin user	john.doe@solve-x.net
Client users	name.one@solve-x.net name.two@solve-x.net name.three@solve-x.net name.four@solve-x.net name.five@solve-x.net name.six@solve-x.net
Password	1234

- **File management**

  https://gestorftp.mi-alojamiento.com/?d=testapplication.eu



**Detail of FileZilla as FTP client.*

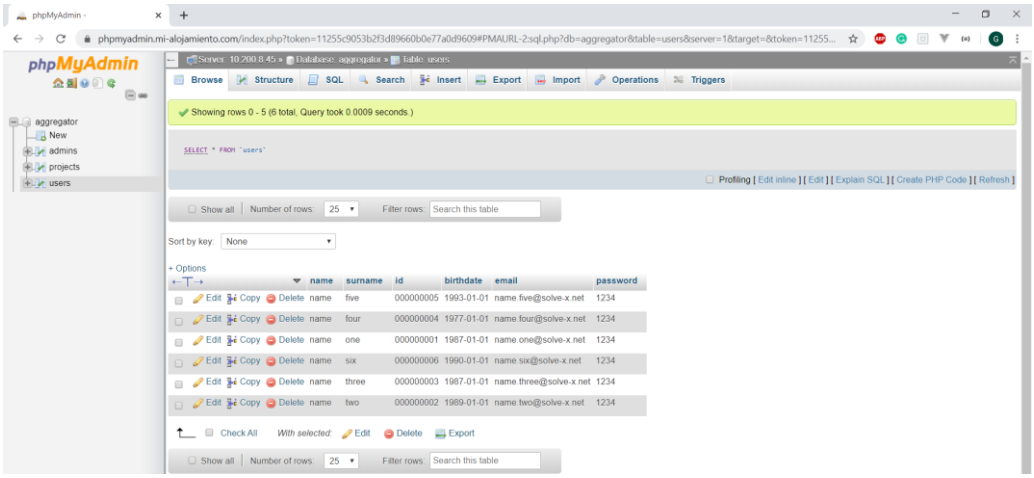
Server: ftp://testapplication.eu

FTP username: testappl

FTP password: 46jm1E33

- **Database management**

  https://phpmyadmin.mi-alojamiento.com/?d=testapplication.eu



MySQL username: mytestappl

MySQL password: pS57y7Xw

7. NOTES

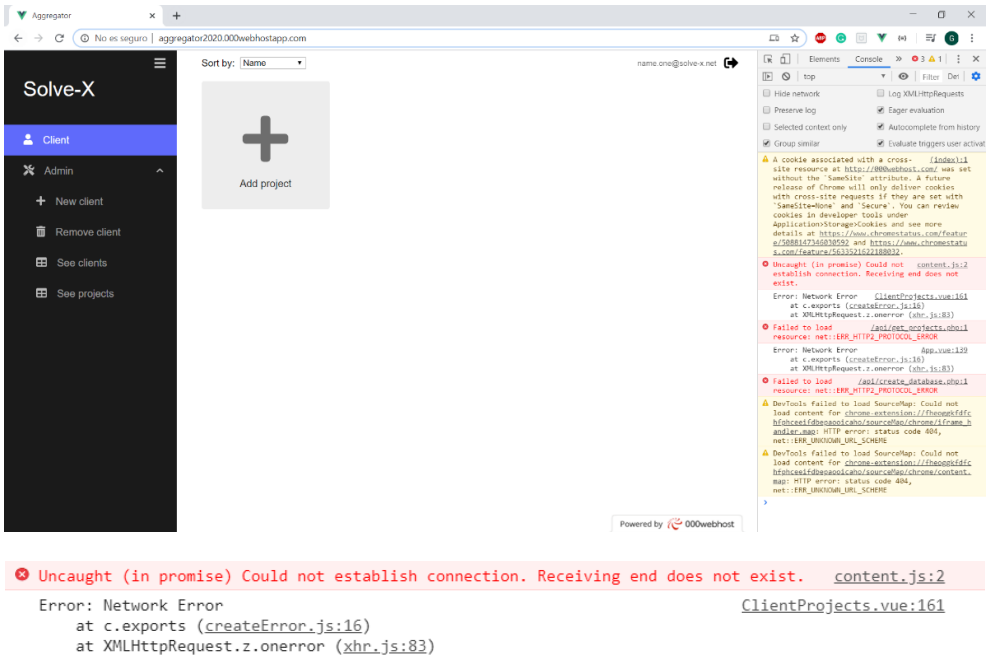
- Since the development process has been carried out on Google Chrome, it is suitable to run the application on that browser.

- There are two reasons why I ended up using a bought domain:

    1. I wanted a quick way to show the application (although connecting as localhost or using an IP address over a Wi-Fi network is great at production mode, it is a too cumbersome way to share results).

    2. The attempt I made in a free hosting server, even if it worked most of the time, it showed erratically network errors:

        http://aggregator2020.000webhostapp.com/



Residual information:

File manager: https://files.000webhost.com/
Website Name: aggregator2020
Password: aggregator

Database Manager: https://databases.000webhost.com/
Username: id13324870_root
Password: Solviks.2020

Fortunately, the hosting service linked to http://testapplication.eu/ has an indefinitely expandable free private testing period. That is why it is mandatory to introduce a username and a password.

- The creation hour registered when adding a new project (as a connected client) may be susceptible to errors when clock changes occur.