INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES HAUTS-DE-FRANCE

Boutillier Paul
HARDUIN Giovanni

SPRING

RAPPORT

Le fonctionnement de notre application
Les différentes ressources
La base de données
Répartition du travail au sein du groupe
Les difficultés rencontrées

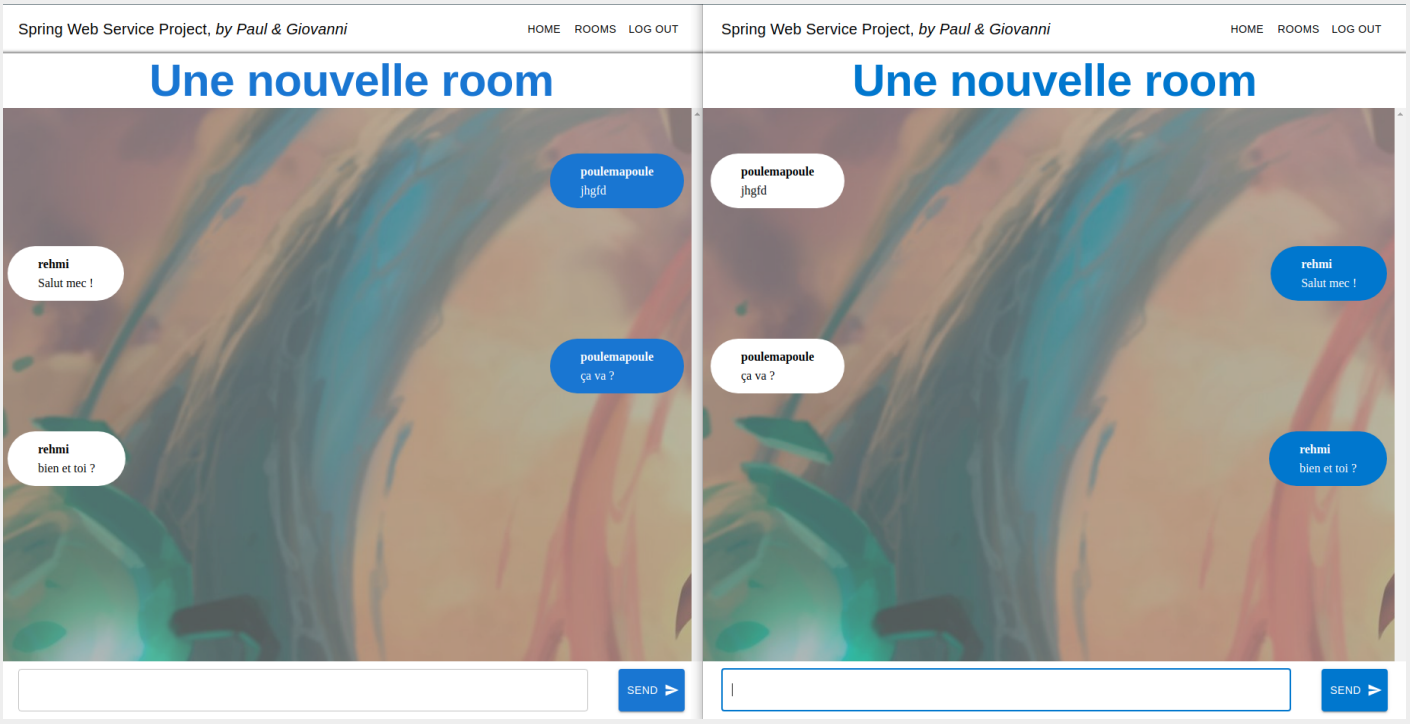
Nous avons tout donné pour ce projet et nous expliquerons au sein de ce rapport pourquoi nous méritons la note suprême :)

1 - Le fonctionnement de notre application

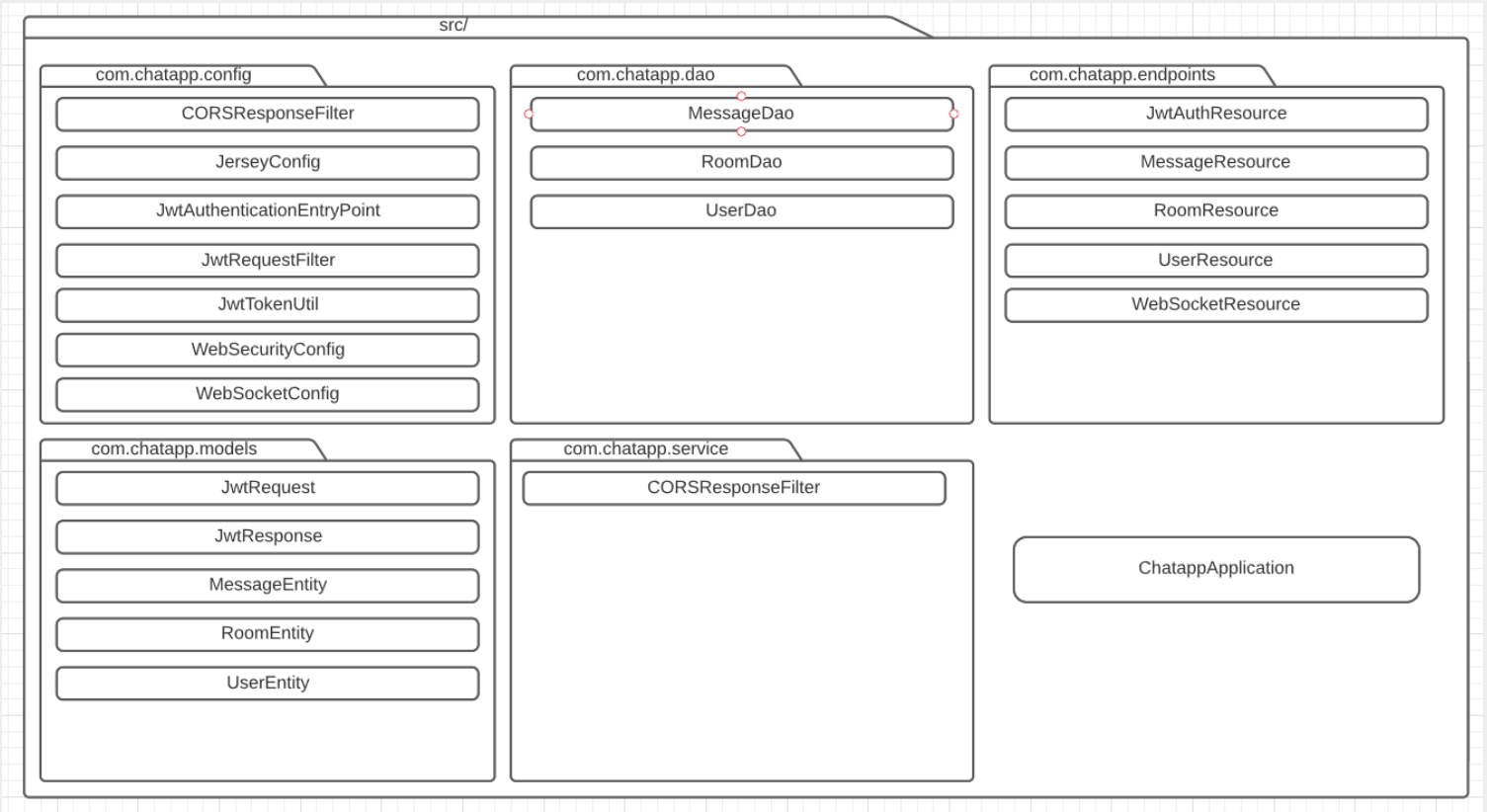
Notre application est une messagerie instantanée similaire à l'application Messenger. Un user peut s'inscrire avec un nom d'utilisateur et un mot de passe afin d'obtenir un compte, chaque user peut ensuite créer un salon afin de discuter avec ses amis mais il peut également rejoindre d'autres salons déjà créés. Notre API nous permet de faire du CRUD (Create, read, update, delete) en effet chaque user peut créer un salon, créer un message, il peut lire les salons déjà créés et lire les messages qui y sont associés. De même elle permet l'update des rooms, après avoir été créées, celles-ci sont modifiables (modifier le nom et la description) par les utilisateurs. Enfin, le delete est possible sur les rooms.

Ce système de messagerie instantanée a pu être possible grâce à l'utilisation des websockets (côté client ainsi que server), en effet l'utilisation de ceux-ci nous on permit de ne pas utiliser un rafraîchissement par intervalle de temps, qui aurait utilisé beaucoup trop de ressources, même sans aucun transfert de données.
De plus, la combinaison des websockets et d'une API REST (deux solutions différentes en informatique) a été un réel challenge pour nous et nous a permis de monter en compétence dans ce domaine.

Par ailleurs, nous avons aussi souhaité que notre application ait un système d'authentification par token afin que les requêtes soient filtrées et découvrir son implémentation par le biais de Spring Boot. Cette fonctionnalité ajoute alors un fichier WebSecurityConfig.java, qui sera très important par la suite.



2 - Les différentes ressources

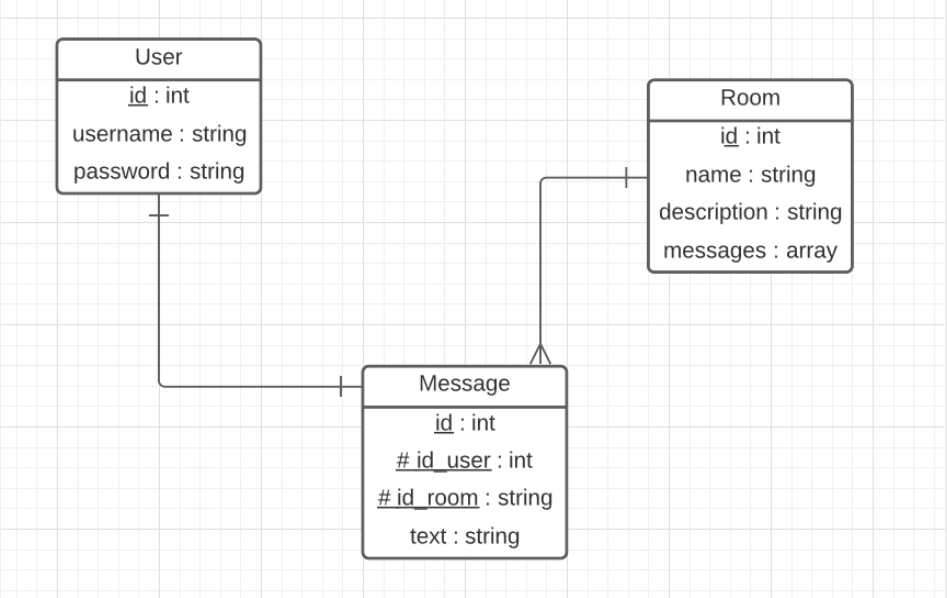


Au niveau des ressources, on peut remarquer une architecture assez complexe du serveur puisque les différentes fonctionnalités que l'on a souhaité implémenter ont nécessité une configuration particulière, mais nous y reviendrons dans les difficultés rencontrées.

Pour ce qui est des ressources réelles et des différents endpoints, voici une liste :

Auth	Users	Rooms	Messages
POST /rest/auth/register	GET /rest/users	POST /rest/rooms	POST /rest/messages
POST /rest/auth/login	GET /rest/users/:id	GET /rest/rooms	GET /rest/messages
		GET /rest/rooms/:id	GET /rest/messages/:id
			PUT /rest/messages/:id
			DELETE /rest/messages/:id

3 - La base de données



La table user est composée de :

- Un id, la clé primaire qui est un integer en auto-incrément.
- Un username qui n'est pas une clé primaire ici, nous avons traité cela dans l'algorithme du serveur lors de la création du compte.
- Un password qui est une chaîne de caractères.

La table room est composée de :

- Un id, la clé primaire qui est un integer en auto-incrément.
- Un name qui est une chaîne de caractères.
- Une description qui est une chaîne de caractères.
- Une liste de messages, obtenue grâce à la relation one to many des entités room et message.

La table message est composée de :

- Un id, la clé primaire qui est un integer en auto-incrément.
- Un id_user, clé étrangère correspondant à l'expéditeur du message.
- Un id_room, clé étrangère correspondant à la room associée au message
- Un text, qui est le contenu du message envoyé par l'utilisateur.

4 - Répartition des tâches au sein du groupe

Nous avons tous deux décidé de l'architecture de notre application et des technologies sur lesquelles nous allions travailler. Nous nous sommes fixés des deadlines afin d'avancer de manière synchrone sur le projet et en faisant des points réguliers sur l'avancement de chacun ainsi qu'en s'aidant sur les difficultées rencontrées que nous citerons ensuite.

Plus à l'aise avec le Java, Paul était l'acteur principal du développement back-end cependant nous nous sommes chacun intéressé sur ce que l'autre faisait afin de monter en compétences sur des technologies que nous ne connaissions pas.

Plus à l'aise avec le développement front-end, Giovanni a su sauté sur l'occasion afin de mettre en valeur ses compétences dans ce domaine, ne connaissant pas le React ce fut un véritable challenge à relever.

Enfin, nous avons tous les deux travaillé sur ce livrable ainsi que sur les différents diagrammes mis à votre disposition dans ce livrable.

5 - Les difficultés rencontrées

Comme vous le remarquerez ci-dessous, le nombre d'embûches sur notre chemin était plutôt important et nous espérons que vous en tiendrez compte des méthodes employées dans la notation.

- **Auth + CORS Policy**

Détails du problème : problème récurrent au sein de notre application, les erreurs de CORS Policy obtenues sur le client nous a donné pas mal de fil à retordre, surtout lorsqu'il s'agissait de l'authentification.
En effet, nos requêtes aboutissaient lors de l'authentification mais retournaient une erreur CORS lorsqu'il s'agissait d'autres endpoint.

Moyens employés afin de le résoudre : suite à de multiples recherches sur le net et de nombreuses revues de code, nous avons pu comprendre l'erreur et la traiter.

```
@Override
protected void configure(HttpSecurity httpSecurity) throws Exception {
    // We don't need CSRF for this example
    httpSecurity.cors().and() HttpSecurity
            .csrf().disable()
                .authorizeRequests() ExpressionUrlAuthorizationConfigurer<...>.ExpressionInterceptUrlRegistry
                .antMatchers( ...antPatterns: "/rest/auth/login", "/rest/auth/register").permitAll()
                .antMatchers( ...antPatterns: "/websocket-chat/**").permitAll()

                // all other requests need to be authenticated
                .anyRequest().authenticated().and().
                // make sure we use stateless session; session won't be used to store user's state.
                        exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint).and().sessionManagement()
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS);

    // Add a filter to validate the tokens with every request
    httpSecurity.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);
}
```

En effet le fichier WebSecurityConfiguration.java nécessitait l'utilisation de la méthode .cors() afin que tous les endpoints restreints par l'authentification utilisent CORS.

- **Auth + WebSocket**

Détails du problème : nous avons pu remarquer lors de notre développement que les websockets n'utilisent pas le même protocole que nos requêtes dites "http". Ainsi nous ne pouvions pas passer de token d'authentification dans l'entête de nos requêtes et nous fûmes fortement handicapé par ce problème.

Moyens employés afin de le résoudre : nous avons découvert que les websockets pouvaient utiliser un autre système d'authentification appelé certificat CRSF mais nous avons décidé de résoudre ce problème plus simplement afin de ne pas alourdir notre charge de travail. Nous avons donc retiré l'authentification sur le endpoint destiné à nos websockets, comme vous pouvez le remarquez sur l'illustration ci-dessous

```
        .csrf().disable()
            .authorizeRequests() ExpressionUrlAuthorizationConfigurer<...>.ExpressionInterceptUrlRegistry
            .antMatchers( ...antPatterns: "/rest/auth/login", "/rest/auth/register").permitAll()
            .antMatchers( ...antPatterns: "/websocket-chat/**").permitAll()
```

- **WebSocket + CORS**

Détails du problème : pour ne plus le citer, ce problème, qui fût déjà très chronophage à résoudre pour l'authentification, le fût encore plus pour l'utilisation de nos websockets. En effet, notre API pouvait désormais communiquer avec nos différents endpoints mais nos requêtes via les websockets restaient bloquées par CORS.

Moyens employés afin de le résoudre : après maintes et maintes heures passées sur stack overflow, nous avons pu trouver la solution. En effet, il s'agissait de rajouter une ligne particulière dans le fichier WebSocketConfig.java, qui est notre fichier de configuration destiné à ceux-ci.

```java
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {
    @Override
    public void registerStompEndpoints(StompEndpointRegistry stompEndpointRegistry) {
        stompEndpointRegistry.addEndpoint( ...paths: "/websocket-chat")
                .setAllowedOrigins("http://localhost:3000")
                .withSockJS();
    }

    @Override
    public void configureMessageBroker(MessageBrokerRegistry registry) {
        registry.enableSimpleBroker( ...destinationPrefixes: "/topic");
        registry.setApplicationDestinationPrefixes("/app");
    }
}
```

Vous pouvez donc remarquer l'utilisation de la méthode setAllowOrigins() qui nous a sauvé de quelques nuits blanches supplémentaires.

- **“Liste de liste de liste de l…”**

Détails du problème : suite à l’implémentation des relations “OneToMany” ainsi que “ManyToOne”, nous avons pu remarquer un problème plutôt cocasse puisque, lorsque nous souhaitions récupérer la listes des rooms, nous obtenions une liste de rooms contenant des messages contenant tous une room contenant des messages contenant tous une room contenant … Je vous laisse imaginer la suite …

Moyens employés afin de le résoudre : notre problème majeur était donc de ne pas récupérer les rooms associées aux messages sans pour autant supprimer une méthode getter de notre entité. Nous avons donc parcouru le net à la recherche d’une annotation qui aurait pu nous permettre cela.

```
@JsonIgnore
public RoomEntity getRoom() { return room; }

@JsonProperty
public void setRoom(RoomEntity room) { this.room = room; }
```

Comme vous pouvez donc le voir ci-dessus, l'annotation @JsonIgnore nous a permis de ne pas retourner les rooms associées au messages dans le json et l’annotation @JsonProperty de bien garder le setter actif, sans lequel nombre d’erreurs sont survenues.

- **Github:**

Détails du problème : à la dernière minute, il nous était impossible de récupérer ou de modifier le projet qui était sur le github de Giovanni, ce qui nous a rajouté une charge de travail supplémentaire et un stress accru.

Moyens employés afin de le résoudre : disposant d’une copie d’une ancienne version de l’application sur le pc de Paul, nous avons donc pu recréer un projet Github et copier les fonctionnalités qui manquaient dans le front-end.

6 - Conclusion

Nous ne connaissions ni Spring ni React, celà nous tenait à cœur de réaliser une application de messagerie instantanée car ce projet relevait beaucoup de problématiques et nous avons dû dans un premier temps réfléchir à l'architecture de notre application et aux technologies que nous allions utiliser. Pour nous, ce fut un excellent challenge car nous avons développé des fonctionnalités que nous n'avions jamais eu l'opportunité de développer avec des technologies et des langages que nous n'avions jamais rencontrés avant ce module. Nous sommes satisfait du résultat final même si l'accumulation de projets et de partiels ne nous a pas permis d'aller au bout de ce que l'on voulait.