Compte rendu

# Projet Ein Code

Antoine CHATAIGNIER Idriss BENGUEZZOU
Mahmoud NASHAR Emrick PONCET

Mai 2022

# Table des matières

# 1 Introduction

Le projet de Développement Web 2 consiste à programmer en Java et en équipe de 4 étudiants une plateforme d'édition collaborative de documents permettant aux utilisateurs d'interagir en direct autour d'un document partagé.

Comme demandé dans le sujet, notre groupe est constitué de quatre étudiants de la licence informatique : M. Idriss BENGUEZZOU, M. Mahmoud NASHAR, M. Emrick PONCET et M. Antoine CHATAIGNIER.

Nous nous sommes réunis après avoir découvert les instructions et les directives du sujet pour essayer de trouver un projet intéressant. Nous avons tout de suite trouvé intéressante l'idée de concevoir un projet similaire à Google Doc, mais il nous a semblé plus pertinent de nous écarter de cette première idée à laquelle de nombreux étudiants auront certainement pensé.

C'est en travaillant sur Eclipse, non sans mal, et en regrettant de ne pouvoir développer en Java simplement sur notre éditeur préféré VSCode que nous nous sommes aperçus de l'intérêt que pourrait avoir la création d'un éditeur de code collaboratif.

C'est à partir de cette observation qu'est né EinCode, l'éditeur collaboratif des développeurs.

# 2 Organisation du groupe

Dès la mise en place de notre groupe nous avons créé un projet sur Gitlab. Nous avons décidé de travailler avec la méthode Agile. Pour ce faire, nous avons convenu un appel de groupe une fois pas semaine pour s'organiser et pour créer les différentes issues du sprint. Chaque membre s'est ensuite attribué des issues à réaliser, de ce fait il ne nous a pas été nécessaire de définir les rôles de chacun. Pour ce projet le chef d'équipe désigné par le groupe est Idriss Benguezzou.

# 3 EinCode

Avant de concevoir un programme ou un logiciel, il est nécessaire d'organiser le travail, de réfléchir à la structure de notre code, mais surtout aux besoins auxquels nous répondons avec notre nouveau produit. Il est nécessaire de focaliser notre attention sur la valeur ajoutée ainsi que sur la cible de notre produit. On pourrait imaginer que cette façon de penser est réservée aux commerciaux, mais il est tout aussi important que le développeur adopte cette vision pour comprendre les besoins auxquels il sera nécessaire de répondre pendant la phase de développement.

Durant tout le processus de développement, la philosophie était claire : concevoir un produit simple, clair et fonctionnel.

Cette philosophie s'applique à tous les aspects d'EinCode, du design au code source, en passant par la construction de la base de données.

## 3.1 Le plan du site : un site clair

### 3.1.1 La première et la plus évidente : la page doc

C'est celle qui intègre l'édition collaborative d'un document. Il existe de nombreuses options de modification du document (droits de lecture, écriture, suppression, partage...) ainsi qu'un chat. Mais pour respecter notre philosophie, nous avons décidé de ne pas rendre directement accessibles toutes ces options afin qu'un nouvel utilisateur, en entrant sur le document, ne voit que ce pourquoi il est venu : la modification.

Tous les autres éléments sont accessibles depuis des onglets, encore une fois aussi simples et clairs que possible. On notera l'utilisation réduite de la couleur primaire pour améliorer l'impact des "call-to-action", mais nous aurons l'occasion d'en reparler plus bas. Il est possible d'afficher et de masquer le chat en cliquant sur un bouton prévu à cet effet pour que ce dernier ne pollue pas l'espace de travail collaboratif.

### 3.1.2 La page "home"

La page d'accueil permet la gestion des documents de l'utilisateur. On retrouvera notamment la liste des documents accessibles à l'utilisateur sur la partie gauche (document dont il est le propriétaire et documents qui sont partagés avec lui).
On pourra aussi retrouver un bouton pour créer un nouveau document et, dessous, la liste des documents en lecture publique ou écriture publique, dans l'optique de découvrir de nouveaux documents partagés à la communauté.

### 3.1.3 La connexion

Une interface permettant de voir l'historique des fichiers auquel l'utilisateur contribue peut s'avérer intéressante, l'utilisateur aurait ainsi un accès à son "historique" des modifications, et pourrait, s'il est propriétaire du fichier, ajouter ou retirer des utilisateurs du groupe.

Le site possède déjà, avec les deux pages précédentes, l'ensemble des pages qui lui sont nécessaires. Cependant, il a été primordial d'intégrer des pages de connexion (login.jsp) et d'inscription (register.jsp) pour permettre à l'utilisateur d'accéder à son espace.

## 3.2 L'application

L'appliction EinCode possède fondamentalement les mêmes fonctionalités que le site web, ainsi les utilisateurs pourront choisir de travailler sur l'application ou de travailler le site web.

### 3.2.1 Connexion

L'application propose une première page à l'utilisateur qui va lui permettre soit de se connecter, soit de créer un compte. Ainsi une fois le compte créé et l'utilisateur connecté, ce dernier sera redirigé sur son menu perosnnel.

### 3.2.2 Un menu personnalisé

EinCode propose à ses utilisateurs une interface après la connexion permettant la gestion de ses projets. Ce menu fait office de page d'accueil pour notre utilisateur et lui permet de manipuler ses documents (création, ouverture, suppression et partage de document).
En cas d'ouverture ou de création d'un document l'utilisateur est redirigé vers le coeur de notre application.

### 3.2.3 Document

L'édition du document est scindée en deux parties, une première évidente pour le document lui-même (permettant de collaborer en temps réel avec les utilisateurs) et une seconde pour le chat attaché au document, celui-ci permet aux utilisateurs de communiquer entre eux et leurs messages sont sauvegardés dans notre base de données.

Bien entendu nous nous engagons à faire preuve d'une totale discrétion à l'égard des données de nos utilisateurs.

Finalement les options de partage, d'édition, de suppression, de chargement et de suppression des documents se retrouvent elles-aussi dans cette partie sous forme d'onglets.

## 3.3 Un design clair

Comme nous l'avons précisé plus haut, le site EinCode a pour ambition d'être le plus clair et le plus simple d'utilisation possible. Le design fait partie intégrante de l'expérience utilisateur. Nous aurions pu prendre un template HTML / CSS existant et s'approprier son style graphique, mais nous avons souhaité construire un design reconnaissable et différent des autres sites. C'est pour cela que nous avons maquetté, dessiné puis développé le style d'EinCode nous-mêmes, de Figma au CSS.

L'utilisation de la couleur primaire a été réduite aux éléments d'action, permettant à chacun de reconnaître instinctivement les éléments avec lesquels il est possible d'interagir, rendant l'expérience utilisateur fluide et confortable.
Concernant le client lourd le design se veut dans la même veine les boutons ton pastel sont facilement reconnaissable sur le fond noir, les elements d'affichages sont blancs et sautent aux yeux.

# 4 Technologies utilisées : le développement du projet

Nous entrons ici dans la partie technique du projet, il est conseillé à chaque lecteur d'ouvrir le code source du projet en parallèle de la lecture pour repérer les fichiers dont nous parlerons.

Comme demandé, ce projet a été conçu en Java, en respectant le modèle MVC (**Model - View - Controller**). Nous n'avions pas pour habitude de respecter un design-pattern pendant le développement de nos projets respectifs. Cela apporte une contrainte en termes de structure de code, mais cela évite de nombreuses erreurs et permet de clarifier le code et simplifier le code en plus de le rendre plus facilement maintenable.

## 4.1 Les packages

Pour respecter ce design pattern, nous avons créé de nombreux package Java dont vous retrouverez le contenu ci-dessous :

- com.eincode.bean : les beans du projet, représentants la **Vue** de notre modèle MVC. On retrouvera ici les classes symbolisant les objets de notre base de données,

- com.eincode.controllers : les supers contrôleurs du projet, c'est-à-dire les classes dont se servent nos contrôleurs pour manipuler la donnée. Ces supers controlleurs sont utilisés à la fois par le client léger, mais aussi pas le client lours, ce qui permet d'éviter une redondance de code, d'optimiser le nombre de nos classes, et de répercuter la modification d'un comportement à la fois sur le client lourd et sur le client léger très simplement,

- com.eincode.dao : les classes DAO (**Data Access Object**) qui permettent d'effectuer des actions sur la base de données. C'est un patron de conception qui fonctionne très bien avec l'autre patron de conception MVC.

- com.dao.helpers : les helpers contiennent les classes utiles pour l'authentification et pour la gestion de la session.

  Avant d'aller plus loin, il est nécessaire de faire une petite parenthèse sur le **helper** SessionManager. Vous pourrez remarquer que chaque méthode des super-contrôleurs demande en premier paramètre un objet de type SessionManager. Ce helper possède de nombreuses variables qu'il est nécessaire de partager d'un fichier à un autre ou d'une page à une autre. Il y a par exemple les informations du document en cours d'édition, les informations de l'utilisateurs actuels, les fichiers auxquels ce dernier a accès, les langages d'édition possibles etc...

  Le helper SessionManager est ce qui fait notre pont entre le client lourd, le client léger et les super-contrôleurs.

- com.eincode.infos : les classes d'information pourraient être assimilées à des super-beans. La classe DocInfos contient par exemple les informations d'un document en particulier, mais aussi et surtout des informations supplémentaires sur ce document comme son créateur, son langage, le contenu de ce document etc..

- com.eincode.jdbc : la gestion de la base de données
- com.eincode.servlets : les contrôleurs du client léger,
- com.eincode.websocket : la gestion des websockets, pour faire communiquer les clients avec le serveur.
- app.eincode.connexion : gestion de la base de données et du client websocket
- app.eincode.docs/lobby/menu : classes correspondantes a nos differentes fenetres
- app.eincode.utils : classes utilitaires.

## 4.2 Le client léger

Pour concevoir le client léger, nous avons utilisé des fichiers JSP, des fichiers CSS et des fichiers JS. Un des premiers intérêts du JSP est qu'il peut inclure des portions de code JSP à l'intérieur même d'un fichier JSP. Cela nous a été très utile, notamment pour gérer l'affichage des erreurs avec un seul fichier ou pour séparer le code et le rendre plus lisible.

On remarquera l'attention portée à la conception de fichiers courts, avec un seul et unique objectif. Le fichier **error.jsp** par exemple ne contient que 10 lignes, mais il est réutilisé sur chacune des pages (home.jsp, login.jsp, register.jsp, doc.jsp).

Le fichier **doc.jsp** inclut chacun des fichiers présents dans le dossier "doc", ce qui permet d'isoler le chat, les paramètres, la partie de partage et la textarea dans des fichiers à part.
On retrouve ici une logique de conception sous forme de composant, chacun étant parfaitement dissocié de son conteneur, permettant une meilleure réutilisation et une meilleur maintenabilité de chaque partie des pages.

## 4.3 Le client lourd

Concernant le client lourd la conception est principalement effectué en JAVA pur et avec la bibliothèque SWING, le but étant ici de réutiliser un maximum le code utilisé pour le client léger quand ceci était possible.

Pour ce faire nous réutilisons les helpers mis en place ainsi que le côté serveur de la partie socket, ce qui signifie que le client léger et lourd sont compatible, il est possible aux utilisateurs de collaborer même sur un type de client différent.

## 4.4 Explication détaillée de la structure et des intéractions du code

Il est temps maintenant de se poser la question de savoir comment le comportement de cette structure s'articule.

Dans un premier temps, l'utilisateur va devoir se connecter ou s'inscrire. C'est le point de départ de tout. La connexion va se réaliser grâce à la classe AuthHelpers, qui va appeler la

classe UserDao. Une fois la connexion établie (ou l'inscription), la SessionManager est initialisée (sauvegarde de l'utilisateur, des fichiers accessibles...).

Pour le client léger, à chaque action de l'utilisateur, une **servlet** va être appelée (doPost ou doGet selon la requête du client), et cette servlet va se servir des super-contrôleurs (ex : DocController) pour effectuer l'action (suppression du doc, modification des droits, sauvegarde etc...). Le super-contrôleur va appeler des classes **DAO** pour effectuer des actions sur les bases et mettre à jour la SessionManager. Enfin, le contrôleur initial, la **servlet**, va rediriger l'utilisateur avec peut-être un message d'erreur dans la réponse.
Ce fonctionnement peut sembler simple, mais il a fallu repenser le projet à plusieurs reprises pour parvenir à cette structure finale du projet.

Pour ce qui est du client lourd le fonctionnement est similaire, concernant la connexion et l'inscription cela est géré via des fichiers séparés qui permettent l'interaction avec la base de donnée.

Au moment de créer un nouveau compte, un numéro d'identification est attribué à l'utilisateur et un dossier est créé pour sauvegarder ces documents localement.

Une fois cela effectué le SessionManager est utilisé, l'interaction entre les différentes fonctionnalités et elle est gérer directement dans les diverses classes via des ActionListener dans lesquelles s'effectue le gros de la logique.

Il manque tout de même un élément dont nous n'avons pas encore parlé et qui se trouve être au cœur du projet : l'aspect collaboratif.
Pour mettre en place l'édition collaborative du code source d'un projet et permettre aux clients de communiquer avec le serveur, nous avons décidé d'utiliser des websockets.

Nous avons créé un système de **rooms** pour les clients. Chaque room correspond à un document, et les messages envoyés entre les clients et le serveur vont être de 3 types différents :

- initialChat : message reçu par le client après sa connexion dans la room, il reçoit la liste des messages précédents stockés dans la base de données
- chat : nouveau message dans le chat
- textarea : modification de la textarea représentant le document partagé.

Schéma entité - associations Diagramme UML

## 4.5 Quelques précisions nécessaires

Pour faciliter le travail de style, nous avons utilisé un framework CSS très répandu appelé **Bootstrap** (v5).

Nous avons préféré **ne pas utiliser jQuery** pour la création d'éléments dans le DOM, JS Vanilla suffisait amplement.

Nous avons eu la chance de trouver **Prism** (https ://prismjs.com/), un outil qui nous a permis de mettre en surbrillance le code à l'intérieur du textarea.
Nous souhaitions pouvoir supporter un grand nombre de langages pour le document partagé, cependant, charger tous les styles css de Prism augmentait considérablement le temps de chargement des pages. Pour pallier ce problème, nous avons rendu l'url du CDN dynamique selon

le langage du document.

En observant le code source, vous pourrez remarquer la superposition d'une div et d'une textarea dans le document. Cela est inspiré du très bon tutoriel :
https://css-tricks.com/creating-an-editable-textarea-that-supports-syntax-highlighted-code/

Cela permet à l'utilisateur d'avoir l'impression d'écrire du texte mis en couleur dans une textarea, cependant, l'utilisateur écrit dans la textarea (transparente, écriture transparent) et le texte est intégré dans une div superposée. Cela permet de conserver le curseur et la possibilité en écriture tout en intégrant un texte mis en surbrillance.

# 5 Les fonctionnalités d'EinCode

Nous avons tenu à construire un projet complet et viable. A ce titre, nous avons travaillé chaque aspect du projet en réfléchissant en permanence à l'expérience utilisateur et aux besoins de ce dernier.

Il est donc bien entendu possible de se créer un nouveau compte, mais uniquement avec une adresse e-mail, cela est vérifié lors de la validation du formulaire.

Il est possible, depuis n'importe quelle page, de se déconnecter de sa session.

Pour actualiser la liste des documents partagés avec l'utilisateur actuel, à chaque retour sur la page d'accueil, le SessionManager se charge de mettre à jour les documents disponibles en édition et en lecture.

Un document créé par l'utilisateur est bien entendu partagé avec celui-ci, mais pourtant, il n'y a pas de duplication d'affichage des documents disponibles (Vos documents et Partagés avec vous).

Il est possible de partager un document avec l'adresse e-mail d'un ami, si cette adresse e-mail n'est pas disponible, un message d'erreur s'affiche.

La liste des utilisateurs ayant accès au document est disponible dans l'onglet "Partager ce document", il est possible pour le créateur de supprimer des droits d'accès pour certains utilisateurs.

L'onglet "Paramètres" du document n'est accessible qu'aux éditeurs du document. Le créateur a en plus un encadré qui lui est propre pour gérer les droits. Le créateur est d'ailleurs le seul à pouvoir supprimer le document.

Il est possible de changer la langue du document, impactant donc la mise en surbrillance des mots du document.

Le créateur du document peut rendre ce dernier accessible à tous en lecture seule ou en écriture.

Si la lecture et l'écriture sont autorisées pour tous, et que le créateur désactive l'autorisation en lecture, un **toast** apparaît indiquant qu'il faut d'abord désactiver l'accès en écriture.

Si la lecture et l'écriture sont désactivées, et que le créateur active l'écriture pour tous, alors la lecture publique s'active automatiquement.

La date, l'heure et le nom de l'envoyeur sont affichés pour chaque message du chat sauf pour les messages de l'utilisateur actuel où le nom n'est pas inscrit (et les messages positionnés à droite).

Un utilisateur ayant uniquement le droit à la lecture sur un document ne peut pas le modifier, mais il voit les modifications en direct du document.

Le chat est conservé pour chaque document et envoyé à chaque utilisateur se connectant au document.

Les fonctionnalités “undo” et “redo” sont naturellement intégrées au textarea.

Il est possible de supprimer un document depuis l'accueil (en tant que créateur) en passant la souris sur le nom du document.

Si un document est supprimé mais que d'autres personnes travaillaient dessus, ces dernières recevront un message d'erreur et seront redirigées sur la page d'accueil.

# 6 Conclusion

Ce projet non seulement très intéressant mais aussi très enrichissant, nous avons construit, pas à pas, un projet viable et potentiellement commercialisable (ou plutôt open-sourcable).

Nous sommes partis d'un besoin existant pour imaginer et développer une réponse à ce besoin en utilisant les technologies vues en cours.

Le seul problème a été l'utilisation d'Eclipse et des serveurs Tomcat, les conflits des serveurs et l'habitude de travailler avec d'autres IDE (notamment VSCode) plus perfectionnés, agréables et intuitifs nous ont fait perdre beaucoup de temps pendant le développement du projet.

Nous avons tenté de créer un projet unique, avec un style unique. Nous avons tenté d'écrire le code le plus clair et lisible possible avec de nombreux packages, dans un souci de maintenabilité et d'évolutivité.

Cela a été une vraie découverte pratique du JSP, du modèle MVC, du patron de conception DAO, des servlets, des websockets en java et de la gestion globale d'une architecture se voulant le plus simple et efficace possible.

# 7 Annexe

Le site **d'OpenClassrooms**, très utile pour comprendre plus en détail le modèle MVC, l'utilisation des JSP, et le modèle DAO : https://openclassrooms.com/fr/courses/2434016-developpez-des-sites-web-avec-java-ee?archived-source=626954

**Figma**, pour maquetter le projet, créer un style unique : https://www.figma.com/

La documentation **Bootstrap** pour créer simplement de nouveaux composants : https://getbootstrap.com/ https ://getbootstrap.com/docs/5.1/https ://getbootstrap.com/docs/5.1/

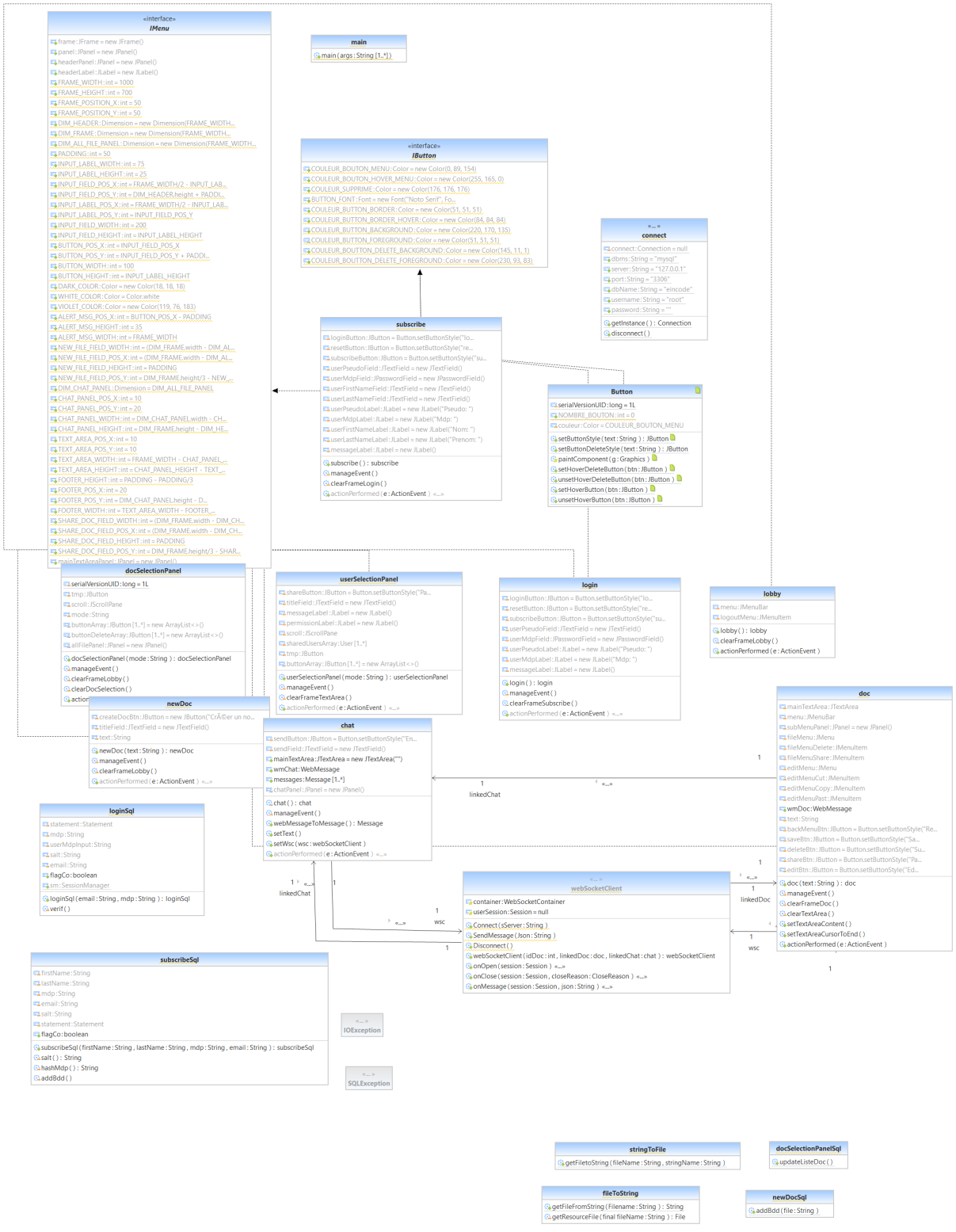
Le merveilleux site **StackOverflow** pour en savoir plus sur les problèmes rencontrés : https://stackoverflow.

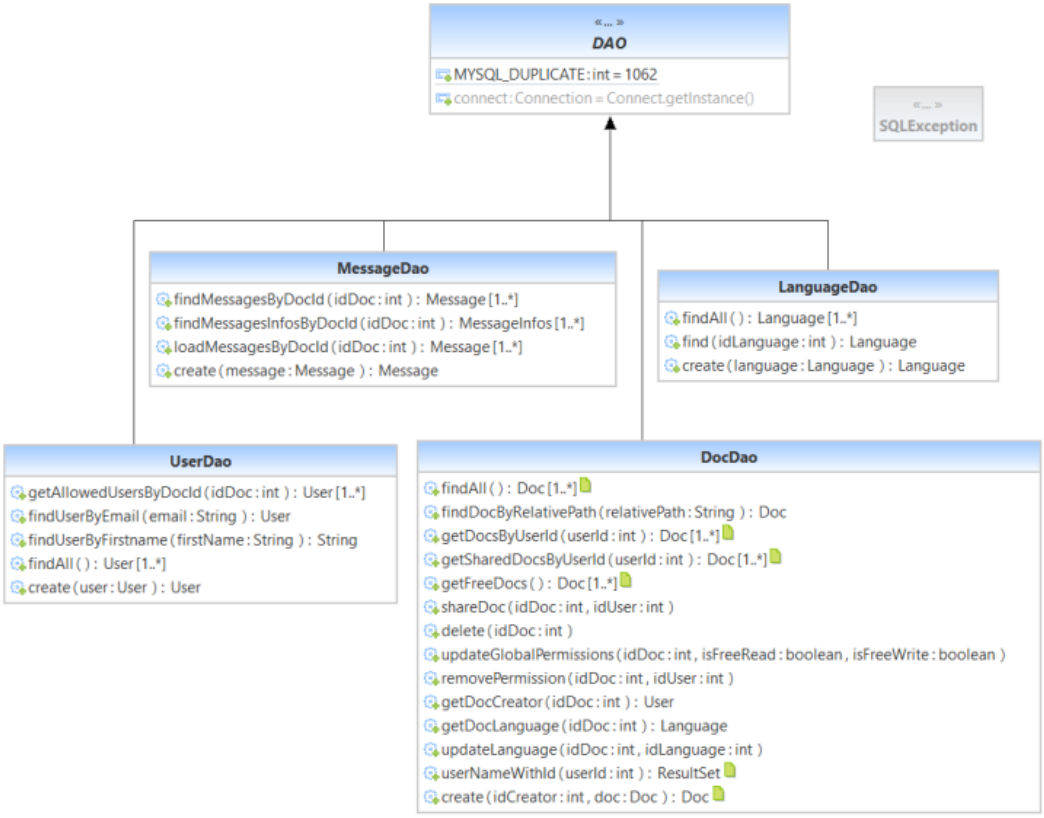
La documentation **Oracle** : https://docs.oracle.com/

**Prism** pour mettre en surbrillance le code : https://prismjs.com/

Un très bon tutoriel pour permettre **l'édition d'une textarea avec la mise en couleur grâce à Prism** : https://css-tricks.com/creating-an-editable-textarea-that-supports-syntax-highlighted-code/

Pour les combinaisons de couleurs https://sanzo-wada.dmbk.io/


«interface»
IMenu
frame : JFrame = new JFrame()
panel : JPanel = new JPanel()
headerPanel : JPanel = new JPanel()
headerLabel : JLabel = new JLabel()
FRAME_WIDTH : int = 1000
FRAME_HEIGHT : int = 700
FRAME_POSITION_X : int = 50
FRAME_POSITION_Y : int = 50
DIM_HEADER : Dimension = new Dimension(FRAME_WIDTH...
DIM_FRAME : Dimension = new Dimension(FRAME_WIDTH...
DIM_ALL_FILE_PANEL : Dimension = new Dimension(FRAME_WIDTH...
PADDING : int = 50
INPUT_LABEL_WIDTH : int = 75
INPUT_LABEL_HEIGHT : int = 25
INPUT_FIELD_POS_X : int = FRAME_WIDTH/2 - INPUT_LAB...
INPUT_FIELD_POS_Y : int = DIM_HEADER.height + PADDI...
INPUT_LABEL_POS_X : int = FRAME_WIDTH/2 - INPUT_LAB...
INPUT_LABEL_POS_Y : int = INPUT_FIELD_POS_Y
INPUT_FIELD_WIDTH : int = 200
INPUT_FIELD_HEIGHT : int = INPUT_LABEL_HEIGHT
BUTTON_POS_X : int = INPUT_FIELD_POS_X
BUTTON_POS_Y : int = INPUT_FIELD_POS_Y + PADDI...
BUTTON_WIDTH : int = 100
BUTTON_HEIGHT : int = INPUT_LABEL_HEIGHT
DARK_COLOR : Color = new Color(18, 18, 18)
WHITE_COLOR : Color = Color.white
VIOLET_COLOR : Color = new Color(119, 76, 183)
ALERT_MSG_POS_X : int = BUTTON_POS_X - PADDING
ALERT_MSG_HEIGHT : int = 35
ALERT_MSG_WIDTH : int = FRAME_WIDTH
NEW_FILE_FIELD_WIDTH : int = (DIM_FRAME.width - DIM_AL...
NEW_FILE_FIELD_POS_X : int = (DIM_FRAME.width - DIM_AL...
NEW_FILE_FIELD_HEIGHT : int = PADDING
NEW_FILE_FIELD_POS_Y : int = DIM_FRAME.height/3 - NEW_...
DIM_CHAT_PANEL : Dimension = DIM_ALL_FILE_PANEL
CHAT_PANEL_POS_X : int = 10
CHAT_PANEL_POS_Y : int = 20
CHAT_PANEL_WIDTH : int = DIM_CHAT_PANEL.width - CH...
CHAT_PANEL_HEIGHT : int = DIM_FRAME.height - DIM_HE...
TEXT_AREA_POS_X : int = 10
TEXT_AREA_POS_Y : int = 10
TEXT_AREA_WIDTH : int = FRAME_WIDTH - CHAT_PANEL_...
TEXT_AREA_HEIGHT : int = CHAT_PANEL_HEIGHT - TEXT_...
FOOTER_HEIGHT : int = PADDING - PADDING/3
FOOTER_POS_X : int = 20
FOOTER_POS_Y : int = DIM_CHAT_PANEL.height - D...
FOOTER_WIDTH : int = TEXT_AREA_WIDTH - FOOTER_...
SHARE_DOC_FIELD_WIDTH : int = (DIM_FRAME.width - DIM_CH...
SHARE_DOC_FIELD_POS_X : int = (DIM_FRAME.width - DIM_CH...
SHARE_DOC_FIELD_HEIGHT : int = PADDING
SHARE_DOC_FIELD_POS_Y : int = DIM_FRAME.height/3 - SHAR...
mainTextAreaPanel : JPanel = new JPanel()
main
main ( args : String [1..*] )
«interface»
IButton
COULEUR_BOUTON_MENU : Color = new Color(0, 89, 154)
COULEUR_BOUTON_HOVER_MENU : Color = new Color(255, 165, 0)
COULEUR_SUPPRIME : Color = new Color(176, 176, 176)
BUTTON_FONT : Font = new Font("Noto Serif", Fo...
COULEUR_BUTTON_BORDER : Color = new Color(51, 51, 51)
COULEUR_BUTTON_BORDER_HOVER : Color = new Color(84, 84, 84)
COULEUR_BUTTON_BACKGROUND : Color = new Color(220, 170, 135)
COULEUR_BUTTON_FOREGROUND : Color = new Color(51, 51, 51)
COULEUR_BOUTTON_DELETE_BACKGROUND : Color = new Color(145, 11, 1)
COULEUR_BOUTTON_DELETE_FOREGROUND : Color = new Color(230, 93, 83)
«... »
connect
connect : Connection = null
dbms : String = "mysql"
server : String = "127.0.0.1"
port : String = "3306"
dbName : String = "eincode"
username : String = "root"
password : String = ""
getInstance ( ) : Connection
disconnect ( )
subscribe
loginButton : JButton = Button.setButtonStyle("lo...
resetButton : JButton = Button.setButtonStyle("re...
subscribeButton : JButton = Button.setButtonStyle("su...
userPseudoField : JTextField = new JTextField()
userMdpField : JPasswordField = new JPasswordField()
userFirstNameField : JTextField = new JTextField()
userLastNameField : JTextField = new JTextField()
userPseudoLabel : JLabel = new JLabel("Pseudo: ")
userMdpLabel : JLabel = new JLabel("Mdp: ")
userFirstNameLabel : JLabel = new JLabel("Nom: ")
userLastNameLabel : JLabel = new JLabel("Prenom: ")
messageLabel : JLabel = new JLabel()
subscribe ( ) : subscribe
manageEvent ( )
clearFrameLogin ( )
actionPerformed ( e : ActionEvent ) «...»
Button
serialVersionUID : long = 1L
NOMBRE_BOUTON : int = 0
couleur : Color = COULEUR_BOUTON_MENU
setButtonStyle ( text : String ) : JButton
setButtonDeleteStyle ( text : String ) : JButton
paintComponent ( g : Graphics )
setHoverDeleteButton ( btn : JButton )
unsetHoverDeleteButton ( btn : JButton )
setHoverButton ( btn : JButton )
unsetHoverButton ( btn : JButton )
docSelectionPanel
serialVersionUID : long = 1L
tmp : JButton
scroll : JScrollPane
mode : String
buttonArray : JButton [1..*] = new ArrayList<>()
buttonDeleteArray : JButton [1..*] = new ArrayList<>()
allFilePanel : JPanel = new JPanel()
docSelectionPanel ( mode : String ) : docSelectionPanel
manageEvent ( )
clearFrameLobby ( )
clearDocSelection ( )
userSelectionPanel
shareButton : JButton = Button.setButtonStyle("Pa...
titleField : JTextField = new JTextField()
messageLabel : JLabel = new JLabel()
permissionLabel : JLabel = new JLabel()
scroll : JScrollPane
sharedUsersArray : User [1..*]
tmp : JButton
buttonArray : JButton [1..*] = new ArrayList<>()
userSelectionPanel ( mode : String ) : userSelectionPanel
manageEvent ( )
clearFrameTextArea ( )
actionPerformed ( e : ActionEvent ) «...»
login
loginButton : JButton = Button.setButtonStyle("lo...
resetButton : JButton = Button.setButtonStyle("re...
subscribeButton : JButton = Button.setButtonStyle("su...
userPseudoField : JTextField = new JTextField()
userMdpField : JPasswordField = new JPasswordField()
userPseudoLabel : JLabel = new JLabel("Pseudo: ")
userMdpLabel : JLabel = new JLabel("Mdp: ")
messageLabel : JLabel = new JLabel()
login ( ) : login
manageEvent ( )
clearFrameSubscribe ( )
actionPerformed ( e : ActionEvent ) «...»
lobby
menu : JMenuBar
logoutMenu : JMenuItem
lobby ( ) : lobby
clearFrameLobby ( )
actionPerformed ( e : ActionEvent )
newDoc
createDocBtn : JButton = new JButton("CrÃ©er un no...
titleField : JTextField = new JTextField()
text : String
newDoc ( text : String ) : newDoc
manageEvent ( )
clearFrameLobby ( )
actionPerformed ( e : ActionEvent ) «...»
chat
sendButton : JButton = Button.setButtonStyle("En...
sendField : JTextField = new JTextField()
mainTextArea : JTextArea = new JTextArea("")
wmChat : WebMessage
messages : Message [1..*]
chatPanel : JPanel = new JPanel()
chat ( ) : chat
manageEvent ( )
webMessageToMessage ( ) : Message
setText ( )
setWsc ( wsc : webSocketClient )
actionPerformed ( e : ActionEvent ) «...»
doc
mainTextArea : JTextArea
menu : JMenuBar
subMenuPanel : JPanel = new JPanel()
fileMenu : JMenu
fileMenuDelete : JMenuItem
fileMenuShare : JMenuItem
editMenu : JMenu
editMenuCut : JMenuItem
editMenuCopy : JMenuItem
editMenuPast : JMenuItem
wmDoc : WebMessage
text : String
backMenuBtn : JButton = Button.setButtonStyle("Re...
saveBtn : JButton = Button.setButtonStyle("Sa...
deleteBtn : JButton = Button.setButtonStyle("Su...
shareBtn : JButton = Button.setButtonStyle("Pa...
editBtn : JButton = Button.setButtonStyle("Ed...
doc ( text : String ) : doc
manageEvent ( )
clearFrameDoc ( )
clearTextArea ( )
setTextAreaContent ( )
setTextAreaCursorToEnd ( )
actionPerformed ( e : ActionEvent )
linkedChat
loginSql
statement : Statement
mdp : String
userMdpInput : String
salt : String
email : String
flagCo : boolean
sm : SessionManager
loginSql ( email : String , mdp : String ) : loginSql
verif ( )
«... »
webSocketClient
container : WebSocketContainer
userSession : Session = null
Connect ( sServer : String )
SendMessage ( Json : String )
Disconnect ( )
webSocketClient ( idDoc : int , linkedDoc : doc , linkedChat : chat ) : webSocketClient
onOpen ( session : Session ) «...»
onClose ( session : Session , closeReason : CloseReason ) «...»
onMessage ( session : Session , json : String ) «...»
linkedDoc
wsc
subscribeSql
firstName : String
lastName : String
mdp : String
email : String
salt : String
statement : Statement
flagCo : boolean
subscribeSql ( firstName : String , lastName : String , mdp : String , email : String ) : subscribeSql
salt ( ) : String
hashMdp ( ) : String
addBdd ( )
«... »
IOException
«... »
SQLException
stringToFile
getFileToString ( fileName : String , stringName : String )
docSelectionPanelSql
updateListeDoc ( )
fileToString
getFileFromString ( Filename : String ) : String
getResourceFile ( final fileName : String ) : File
newDocSql
addBdd ( file : String )


«...»
DAO
MYSQL_DUPLICATE : int = 1062
connect : Connection = Connect.getInstance()
«...»
SQLException
MessageDao
findMessagesByDocId ( idDoc : int ) : Message [1..*]
findMessagesInfosByDocId ( idDoc : int ) : MessageInfos [1..*]
loadMessagesByDocId ( idDoc : int ) : Message [1..*]
create ( message : Message ) : Message
LanguageDao
findAll ( ) : Language [1..*]
find ( idLanguage : int ) : Language
create ( language : Language ) : Language
UserDao
getAllowedUsersByDocId ( idDoc : int ) : User [1..*]
findUserByEmail ( email : String ) : User
findUserByFirstname ( firstName : String ) : String
findAll ( ) : User [1..*]
create ( user : User ) : User
DocDao
findAll ( ) : Doc [1..*]
findDocByRelativePath ( relativePath : String ) : Doc
getDocsByUserId ( userId : int ) : Doc [1..*]
getSharedDocsByUserId ( userId : int ) : Doc [1..*]
getFreeDocs ( ) : Doc [1..*]
shareDoc ( idDoc : int , idUser : int )
delete ( idDoc : int )
updateGlobalPermissions ( idDoc : int , isFreeRead : boolean , isFreeWrite : boolean )
removePermission ( idDoc : int , idUser : int )
getDocCreator ( idDoc : int ) : User
getDocLanguage ( idDoc : int ) : Language
updateLanguage ( idDoc : int , idLanguage : int )
userNameWithId ( userId : int ) : ResultSet
create ( idCreator : int , doc : Doc ) : Doc