



Bachelor 3 CDA (Concepteur Développeur d'Applications)

Année académique : 2024/2025

Dossier de projet

TASKLIST

Application pour gérer ses tâches



Réalisé par :

Adam Younsi

adam.younsi@laplateforme.io

Remerciement

Ce dossier marque l'aboutissement d'un parcours exigeant, mais profondément formateur, dans le cadre de ma formation en Bachelor 3 – Concepteur Développeur d'Applications.

Il représente non seulement la synthèse de compétences techniques acquises au fil du temps, mais aussi l'illustration concrète de ma capacité à mener un projet de bout en bout, avec rigueur, autonomie et persévérance.

Je tiens à remercier chaleureusement mes formateurs, mes camarades de promotion, ainsi que les intervenants professionnels qui ont su me guider, me conseiller et parfois me challenger avec exigence.

Leurs retours constructifs ont fortement contribué à faire évoluer ce projet et m'ont permis d'atteindre un niveau de maturité technique et organisationnelle que je n'aurais pas imaginé en début de parcours.

Ce projet a également été une aventure personnelle : il m'a permis de découvrir mes limites, de les repousser, mais surtout de confirmer mon appétence pour l'ingénierie logicielle, la conception d'applications et les problématiques concrètes liées au développement logiciel moderne.

Table des matières

1. Introduction.....	4
2. Listes des compétences visées.....	5
a. Technologies principale utilisée.....	5
3. Cahier des charges.....	7
a. Objectif du projet.....	7
b. Utilisateur cible.....	7
c. Fonctionnalités principales.....	8
d. Contraintes techniques et fonctionnelle.....	8
e. Méthodologie adoptée.....	9
4. Gestion de projet.....	11
a. Outils utilisés.....	11
b. Suivi et adaptation.....	11
5. Conception.....	13
a. Analyse des besoins.....	13
b. Maquette wireframe et figma.....	13
c. Vue d'ensemble figma avec liaison.....	23
6. Architecture logicielle.....	24
a. Programmation orienté objet.....	24
b. Application des principes solid.....	26
c. Organisation des packages – architecture en couche.....	27
d. Sécurité – Authentification, hashing et token.....	28
e. Middleware – filtre personnalisé.....	29
f. Rest et séparation des responsabilité.....	29
g. Définition et concept clé.....	29
7. Développement backend.....	33
a. Technologie principale.....	33
b. Contrôleur et service.....	33
c. Authentification, JWT et sécurité des routes.....	33
d. Sécurité côté backend.....	34
e. Test backend.....	34
8. Cahier des recettes planification des tests.....	35
9. Développement frontend.....	36
a. Technologie et structure.....	36
b. Structure des composant.....	36
c. Appel api.....	36
d. Test frontend.....	37
e. Test fonctionnel avec Cypress.....	39
10. Base de données.....	40
a. MCD et MLD modélisés.....	40
b. Base relationnelles MySQL+Docker.....	42
c. Tables principales.....	42

d. Backup, restauration.....	43
e. Jeu d'essai utilisée.....	43
11. CI/CD et déploiement.....	44
a. Introduction aux outils CI/CD.....	44
b. Conteneurisation avec docker.....	45
c. Github Actions et pipeline CI/CD.....	47
12. Veille technologique.....	49
a. Objectif de la veille.....	49
b. Veille sur la sécurité.....	49
c. Veille sur les tests automatisé.....	49
d. Découverte d'outils professionnelle.....	50
e. Source et plateforme utilisées.....	50
13. Problèmes rencontré et solutions.....	51
a. Problème 1 : authentification JWT.....	51
b. Problème 2 : installation de jest.....	51
c. Problème 3 : Configuration NGINX dans docker.....	52
d. Problème 4 : installation de Cypress.....	53
14. Sybthèse/Conclusion.....	53
a. Bilan projet.....	53
b. Satisfaction personnelle.....	53

Introduction

Je m'appelle **Younsi Adam**, j'ai 23 ans et je suis actuellement étudiant en **Bachelor 3** dans la spécialité **Concepteur et Développeur d'Applications (CDA)**.

Ce dossier constitue l'aboutissement de mon année de formation et vise à valider le titre professionnel RNCP 37873, reconnu au niveau 6.

Passionné depuis toujours par les technologies du web, la programmation et les défis techniques, j'ai choisi de réaliser ce projet de manière **individuelle** afin de me confronter pleinement aux réalités d'un cycle de développement complet, en m'appuyant sur les bonnes pratiques du métier.

Le projet que je présente ici est une **application web de gestion de tâches** (tasklist), développée avec des technologies modernes et respectueuses des standards de sécurité et de performance.

L'objectif de cette application est simple mais représentatif : permettre à un utilisateur de créer, gérer, modifier et suivre l'état de ses tâches quotidiennes dans une interface claire, responsive et sécurisée.

Ce projet m'a permis de mettre en œuvre les compétences acquises dans les trois blocs du référentiel CDA :

- Le **développement d'interfaces utilisateur et de composants métiers**
- La **conception d'une architecture logicielle multicouche**
- Et la **préparation au déploiement et à l'intégration continue** dans une démarche DevOps

Ce dossier vise à retracer la réalisation complète de ce projet : de l'expression du besoin à la mise en production, en passant par les choix techniques, les développements effectués, les obstacles rencontrés et les solutions mises en œuvre.

Liste des compétences visées

Dans le cadre de la préparation au titre RNCP 37873 – Concepteur Développeur d'Applications, mon projet m'a permis de mobiliser les compétences attendues dans les trois blocs de certification.

Tout au long de ce projet, j'ai adopté une approche professionnelle, en simulant les conditions réelles d'un développement applicatif en entreprise : analyse des besoins, conception technique, implémentation fonctionnelle, tests, conteneurisation, et automatisation du déploiement.

Chaque phase a été l'occasion d'explorer en profondeur les outils et pratiques du métier, en lien direct avec les compétences du référentiel.

Technologies principales utilisées

- **Backend** : Java, Spring Boot, Maven
- **Frontend** : React, TypeScript, CSS, HTML
- **Base de données** : MySQL
- **DevOps / CI/CD** : Docker, Docker Compose, GitHub Actions
- **Outils** : Trello, GitHub, Figma, Postman, MySQL workbench, Cypress, VSCode

L'ensemble de ces technologies a été sélectionné pour répondre aux contraintes modernes de développement, de sécurité, de maintenabilité et de déploiement d'une application web professionnelle.

Le tableau ci-dessous synthétise ces compétences, les technologies associées ainsi que le niveau de maîtrise évalué.

Bloc RNCP	Compétences visées	Projet concerné	Niveau atteint*
Bloc 1	Installer et configurer l'environnement de travail	TaskList	Acquis
Bloc 1	Développer des interfaces utilisateur	TaskList (React)	Acquis
Bloc 1	Développer des composants métier (Spring Boot)	TaskList	Acquis
Bloc 1	Contribuer à la gestion de projet informatique	Trello, GitHub	Acquis
Bloc 2	Analyser les besoins et maquetter une application	Wireframes Figma	Acquis
Bloc 2	Définir l'architecture logicielle d'une application multicouche	Spring / React	Acquis
Bloc 2	Concevoir et mettre en place une base de données relationnelle	MySQL	Acquis
Bloc 2	Développer des composants d'accès aux données SQL	Spring Data JPA	Acquis
Bloc 3	Préparer et exécuter les plans de tests d'une application	Postman, Jest, Cypress	Acquis
Bloc 3	Préparer et documenter le déploiement d'une application	Docker, CI/CD	Acquis
Bloc 3	Contribuer à la mise en production (CI/CD, DevOps)	GitHub Actions	Acquis

Cahier des charges

Objectif du projet

L'application **TaskList** a été conçue comme un outil personnel d'organisation permettant de centraliser la gestion des tâches quotidiennes à travers un système de **listes personnalisées**. L'objectif est de proposer une interface simple, rapide et moderne pour :

- Créer et organiser ses tâches
- Structurer ses journées ou ses projets par liste
- Marquer les tâches comme effectuées de façon visuelle
- Travailler dans un environnement agréable, responsive et sécurisé

Ce projet s'inscrit dans une **logique professionnelle**, avec une architecture backend robuste (Spring Boot), un frontend moderne (React), une base de données structurée, et une mise en production conteneurisée.

Il a été développé de manière autonome pour reproduire le **cycle complet de réalisation d'une application web**, du besoin utilisateur jusqu'au déploiement et à la documentation.

Utilisateur cible

L'utilisateur type de l'application est une personne souhaitant :

- Créer plusieurs listes de tâches (par thème, par projet, etc.)
- Ajouter, afficher et supprimer des tâches dans chaque liste
- Visualiser facilement les tâches effectuées
- Personnaliser l'apparence et l'utilisation de l'interface (dark mode, responsive)
- Accéder à son espace personnel en toute sécurité

Il peut s'agir :

- d'un étudiant qui organise ses cours et révisions
- d'un professionnel qui souhaite suivre ses tâches par projet
- ou de toute personne recherchant une solution simple pour rester productif

Fonctionnalités principales

Voici les fonctionnalités actuellement disponibles dans l'application :

Gestion des listes

- Création d'une ou plusieurs listes personnalisées
- Affichage des listes dans une interface claire
- Renommage possible de chaque liste
- Suppression d'une liste

Gestion des tâches (par liste)

- Création de tâches associées à une liste (obligatoire)
- Affichage des tâches par liste
- Marquage visuel d'une tâche terminée (ligne traversant le texte)
- Suppression d'une tâche terminée ou non

Gestion du profil

- Authentification sécurisée (inscription / connexion avec JWT)
- Affichage personnalisé avec avatar et email utilisateur
- Accès à un tableau de bord après connexion

Interface utilisateur

- Affichage clair et responsive (ordinateur et mobile)
- Activation du **mode sombre (dark mode)** pour le confort visuel

Contraintes techniques et fonctionnelles

Le projet respecte les bonnes pratiques de sécurité et d'architecture logicielle, notamment :

Sécurité

- Authentification sécurisée via Spring Security et JWT
- Vérification des permissions pour chaque action utilisateur

- Validation des champs côté frontend et backend

Architecture

- Séparation des responsabilités : Controller, Service, Repository
- Utilisation de DTO pour protéger la couche API
- Tests sur les endpoints (via Postman) et organisation par modules

Base de données

- Chaque utilisateur dispose de ses propres listes et tâches
- Modélisation relationnelle avec MySQL
- Données sécurisées et liées via les relations

Interface utilisateur

- Application responsive en React
- Navigation fluide et mise à jour dynamique des données
- Expérience utilisateur pensée pour la simplicité

DevOps

- Application conteneurisée via Docker
- Utilisation de docker-compose pour lier backend, frontend et BDD
- Intégration continue avec GitHub Actions

Méthodologie adoptée : Kanban avec Trello

L'organisation du projet s'est faite via un tableau Trello structuré en Kanban.

Chaque carte représente une tâche ou un lot fonctionnel à développer.

Le tableau comprend les colonnes suivantes : Backlog, À faire, En cours, Terminé, Terminé Backend, Terminé Frontend.

Chaque carte contient :

- Un titre clair
- Une date limite
- Des étiquettes techniques (Frontend, Backend, CI/CD...)
- Et parfois une checklist pour les tâches complexes

Cela m'a permis de suivre l'évolution du projet de façon structurée et de prioriser intelligemment les développements.

Illustration – Tableau Trello

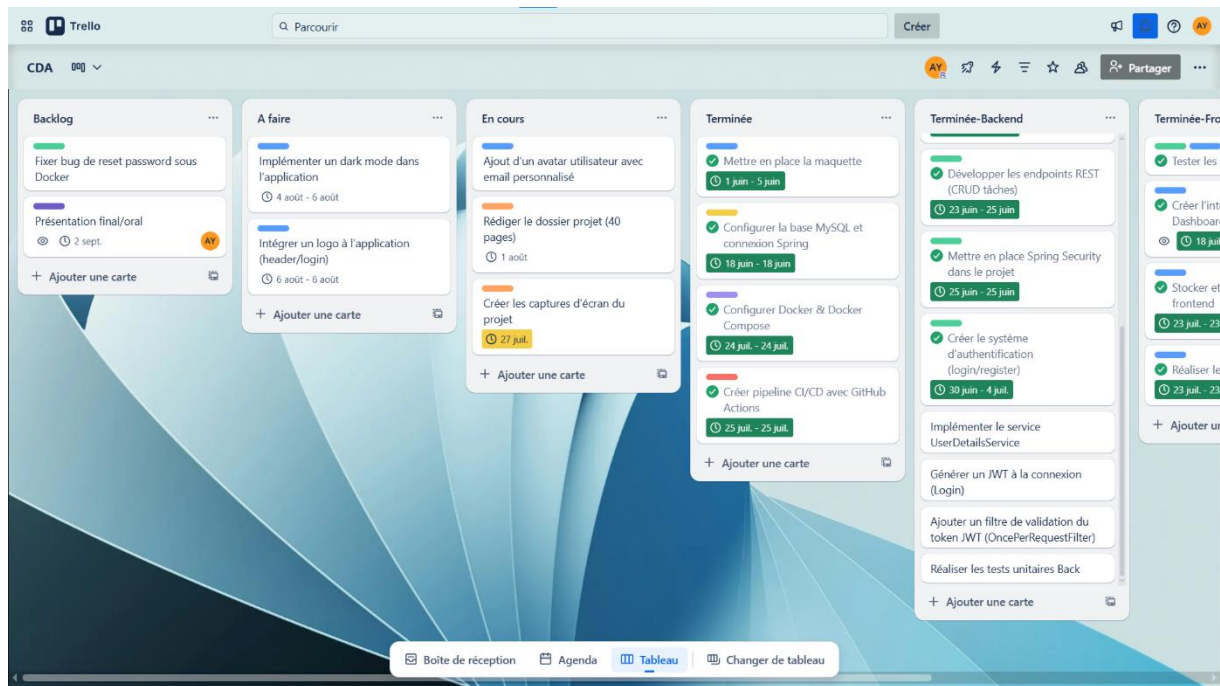


Figure 1 : Organisation globale du projet dans Trello partie 1

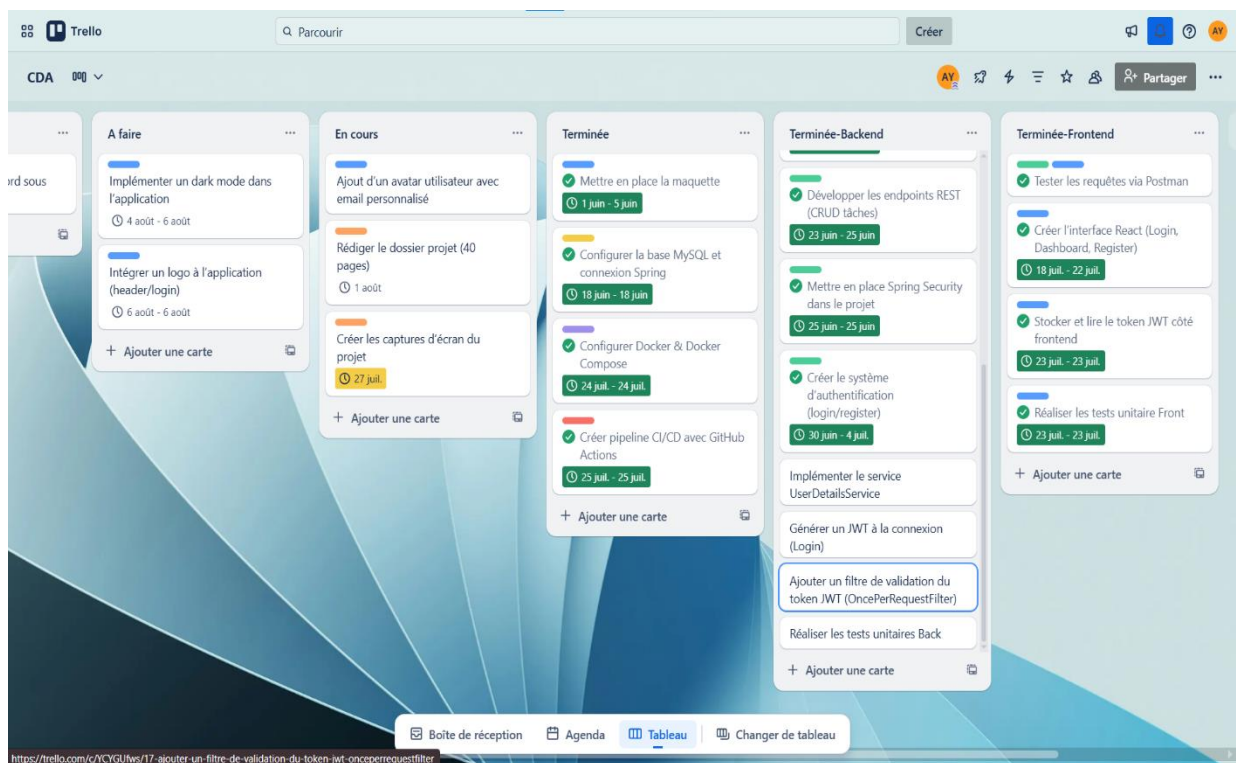


Figure 2 : Organisation globale du projet dans Trello partie 2

Gestion de projet

Outils utilisés

Le projet **Tasklist** a été développé selon une approche professionnelle en utilisant les outils suivants :

- **Git** : versionnage complet du code source (backend et frontend)
- **GitHub** : hébergement du dépôt et mise en place de l'intégration continue
- **GitHub Actions** : automatisation du processus CI/CD
- **Trello** : gestion visuelle du projet via une approche Kanban
- **Figma** : réalisation des wireframes et de la maquette
- **Jest** : outils de tests unitaires frontend
- **Cypress** : outils de tests fonctionnel frontend
- **Postman** : outils de tests des requêtes HTTP des Endpoint
- **spring-boot-starter-test** : Dépendance de tests unitaires backend
- **MySQL Workbench** : interface visuelle pour faciliter la manipulation des données
- **Docker** : conteneurisation de l'application
- **VSCode** : IDE pour rédaction du code

Suivi et adaptation

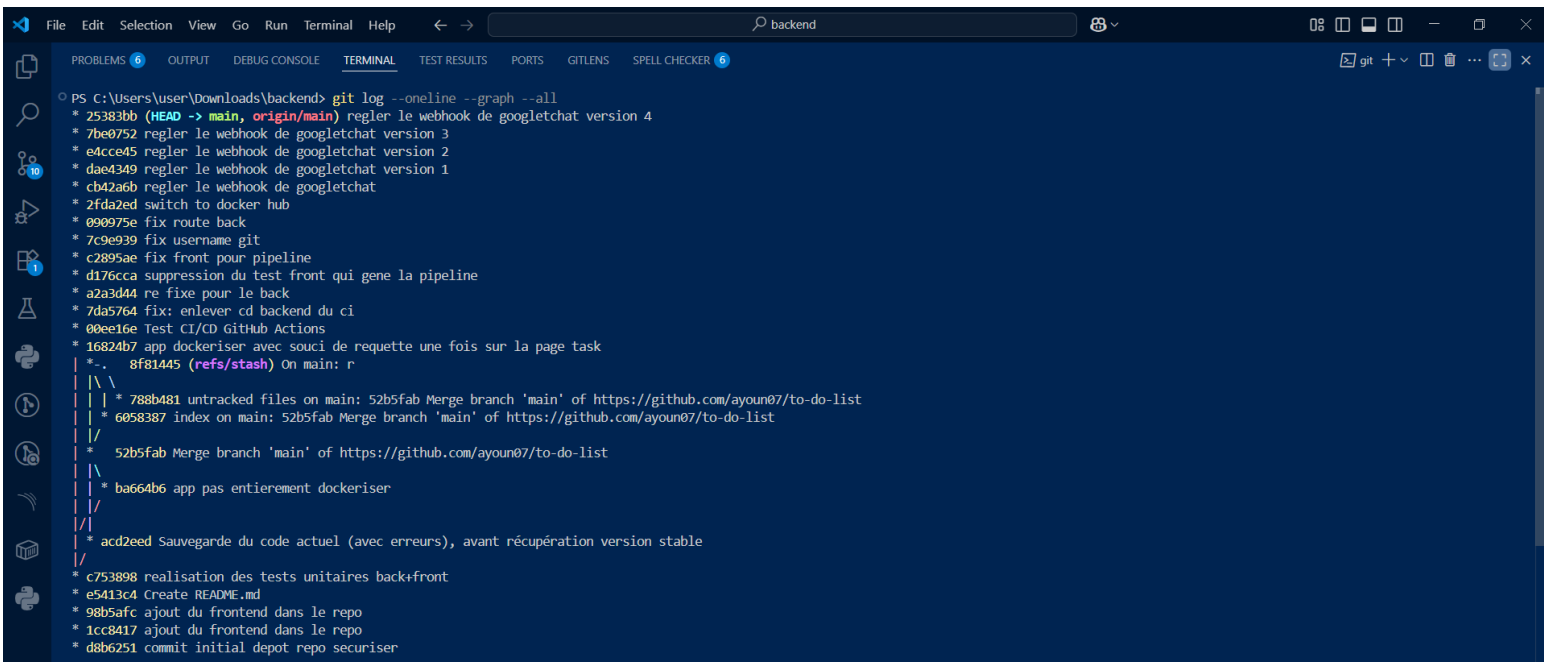
Le suivi du projet a été assuré à l'aide de **Git** pour la gestion des versions et **Trello** pour l'organisation des tâches.

Chaque évolution du code a été **committée avec soin**, comme le montre l'historique Git ci-dessous.

Ce processus m'a permis :

- D'assurer la **traçabilité des développements**
- De revenir facilement à une version stable

- De documenter l'évolution du projet par étapes claires
- De détecter et corriger rapidement les erreurs ou conflits



```

PS C:\Users\user\Downloads\backend> git log --oneline --graph --all
* 25383bb (HEAD -> main, origin/main) regler le webhook de googletchat version 4
* 7be0752 regler le webhook de googletchat version 3
* e4cce45 regler le webhook de googletchat version 2
* dae4349 regler le webhook de googletchat version 1
* cb42a6b regler le webhook de googletchat
* 2fda2ed switch to docker hub
* 090975e fix route back
* 7c9e939 fix username git
* c2895ae fix front pour pipeline
* d176cca suppression du test front qui gene la pipeline
* a2a3d44 re fixe pour le back
* 7da5764 fix: enlever cd backend du ci
* 00ee16e Test CI/CD GitHub Actions
* 16824b7 app dockeriser avec souci de requette une fois sur la page task
| *.. 8f81445 (refs/stash) On main: r
| \ \
| | | * 788b481 untracked files on main: 52b5fab Merge branch 'main' of https://github.com/ayoun07/to-do-list
| | | * 6058387 index on main: 52b5fab Merge branch 'main' of https://github.com/ayoun07/to-do-list
| | /
| | * 52b5fab Merge branch 'main' of https://github.com/ayoun07/to-do-list
| | \
| | * ba664b6 app pas entierement dockeriser
| | /
| | /
| | * acd2eed Sauvegarde du code actuel (avec erreurs), avant récupération version stable
| /
* c753898 realisation des tests unitaires back+front
* e5413c4 Create README.md
* 98b5afc ajout du frontend dans le repo
* 1cc8417 ajout du frontend dans le repo
* d8b6251 commit initial depot repo securiser
  
```

Figure 3 : Historique Git du projet Tasklist (commande git log --oneline --graph --all)

Conception

Analyse des besoins

User Stories (Histoires Utilisateur)

- En tant qu'utilisateur, je souhaite m'inscrire pour pouvoir accéder à l'application.
- En tant qu'utilisateur, je veux pouvoir me connecter de manière sécurisée avec un mot de passe chiffré.
- En tant qu'utilisateur connecté, je veux pouvoir créer une ou plusieurs listes de tâches.
- En tant qu'utilisateur connecté, je veux ajouter, supprimer ou terminer des tâches dans une liste.
- En tant qu'utilisateur, je veux recevoir un e-mail de confirmation lors de l'inscription ou de la réinitialisation du mot de passe.

Cas d'utilisation (Use Cases)

Acteur	Cas d'utilisation	Description
Utilisateur	S'inscrire	Formulaire avec validation, création de compte, e-mail de confirmation.
Utilisateur	Se connecter	Authentification avec JWT et accès au dashboard.
Utilisateur	Créer une liste de tâches	Interface simple pour nommer et enregistrer une liste.
Utilisateur	Gérer ses tâches	Ajouter, supprimer ou terminer une tâche.
Utilisateur	Réinitialiser son mot de passe	Envoi d'un e-mail avec lien temporaire.

Maquettes : Wireframes et Figma

Wireframes (maquettes filaires)

Les wireframes illustrent les écrans clés de l'application et permettent de valider les parcours utilisateur avant d'entamer le design graphique final.

Chaque écran a été extrait individuellement pour en faciliter la lisibilité et l'explication.

Login Page

Cette page permet à l'utilisateur de se connecter avec un identifiant et un mot de passe. Un lien permet d'accéder à la page d'inscription. Un second lien offre la possibilité de réinitialiser son mot de passe.

The image shows a login form titled "Connexion" with a lock icon. It contains two input fields: "Nom d'utilisateur" and "Mot de passe". Below these is a button labeled "Se connecter". At the bottom, there are two links: "Pas encore de compte ? S'inscrire" and "Mot de passe oublié ?".

Figure 4 : LoginPage

Register Page

Cette page contient un formulaire d'inscription. L'utilisateur doit remplir :

- Nom d'utilisateur
- Adresse email
- Mot de passe ainsi que la confirmation du mot de passe

Le mot de passe doit être conforme aux critères de sécurité. Une validation dynamique prévient si les critères ne sont pas respectés. Si c'est le cas le bouton S'inscrire s'active et un mail contenant un lien d'activation est envoyé au mail renseigné par l'utilisateur.

Inscription

Nom d'utilisateur

Adresse email

Mot de passe

Confirmer le mot de passe

Le mot de passe doit contenir :

- ✓ Minimum 8 caractères
- ✓ Une majuscule
- ✓ Une minuscule
- ✓ Un chiffre
- ✓ Un caractère spécial
- ✓ Confirmation du mot de passe

S'inscrire

Déjà inscrit ? Se connecter

Figure 5 : RegisterPage

Forgot Password Page

Cet écran propose un champ unique permettant de saisir une adresse e-mail. Lors de la soumission, un e-mail est envoyé à l'utilisateur contenant un lien de réinitialisation du mot de passe.

Ce lien est composé de deux parties :

- Une requête HTTP construite à partir d'un **endpoint** défini dans le fichier AuthController.
- Un **jeton (token)** généré aléatoirement, à usage unique, avec une durée de validité limitée. Ce jeton est temporairement enregistré dans la base de données et est automatiquement supprimé une fois utilisé par l'utilisateur pour réinitialiser son mot de passe.

Enfin, un lien est également proposé pour permettre à l'utilisateur de revenir à la page de connexion.

Mot de passe oublié 🔒

Entrez votre email

Envoyer

[Retour à la connexion](#)

Figure 6 : Forget-passwordPage

Reset Password Page

Cette page permet à l'utilisateur de définir un nouveau mot de passe, via un lien reçu par email. Un bouton permet de valider et un lien redirige vers la connexion

Réinitialiser le mot de passe 🔑

Nouveau mot de passe

Réinitialiser

[Retour à la connexion](#)

Figure 7 : Reset-PasswordPage

Task Page (Tableau de bord)

La page principale de l'application présente une interface organisée en deux zones. Sur la barre latérale, l'utilisateur peut visualiser l'ensemble de ses listes de tâches existantes, ainsi qu'ajouter de nouvelles listes. Chaque liste affichée dispose de deux boutons positionnés à sa gauche : le premier permet de renommer la liste, le second de la supprimer, ainsi que toutes les tâches qui y sont associées.

Dans la zone principale, un champ de saisie permet d'ajouter de nouvelles tâches à la liste sélectionnée. Lorsqu'une tâche est cliquée, une ligne de type "barré" apparaît sur le texte, indiquant qu'elle a été réalisée. Un bouton spécifique permet ensuite de supprimer les tâches terminées de manière claire et efficace.

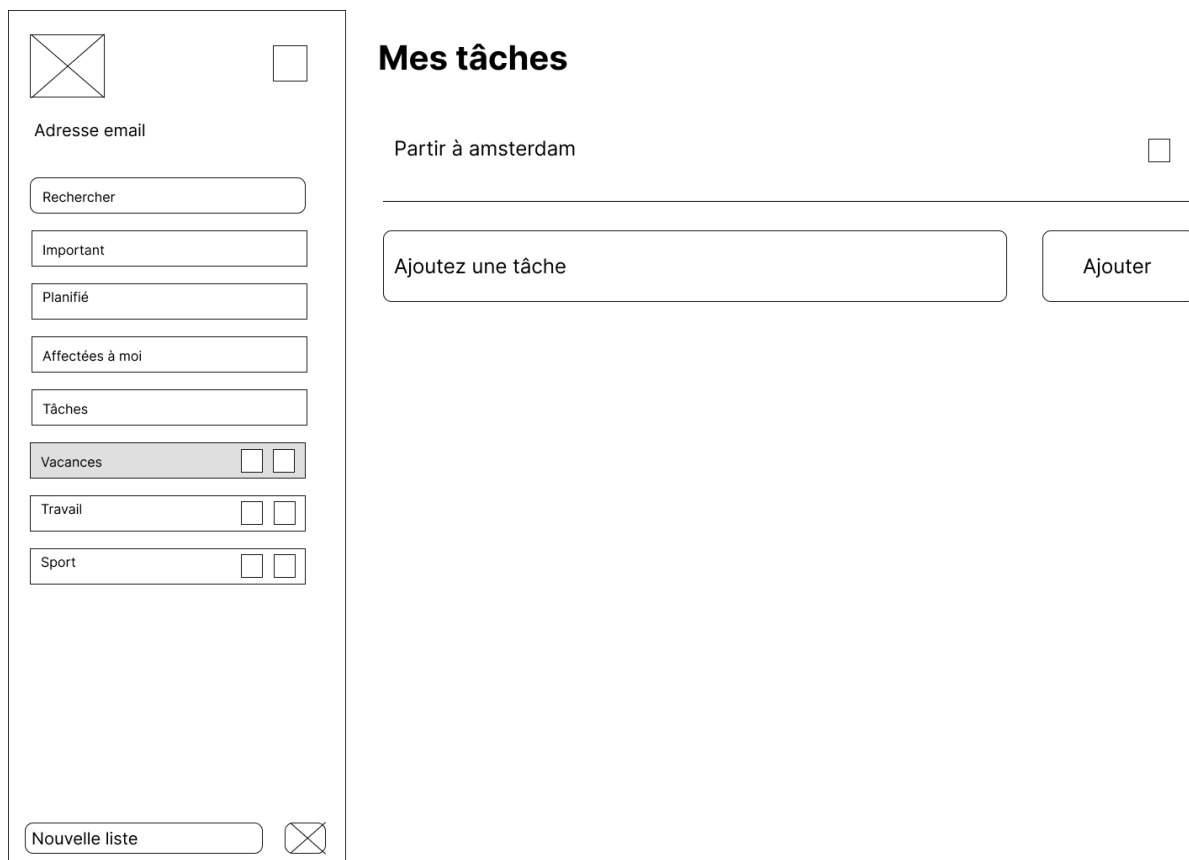


Figure 8 : TaskPage

Chaque wireframe est accompagné de flèches illustrant les liens entre les pages, renforçant la compréhension du parcours utilisateur.

L'image suivante regroupe tous les wireframes du projet avec les flèches de navigation. Cette vue synthétique permet de comprendre le **parcours utilisateur global** ainsi que les **transitions prévues entre chaque écran**. Elle est très utile pour vérifier la logique de navigation avant l'intégration technique.

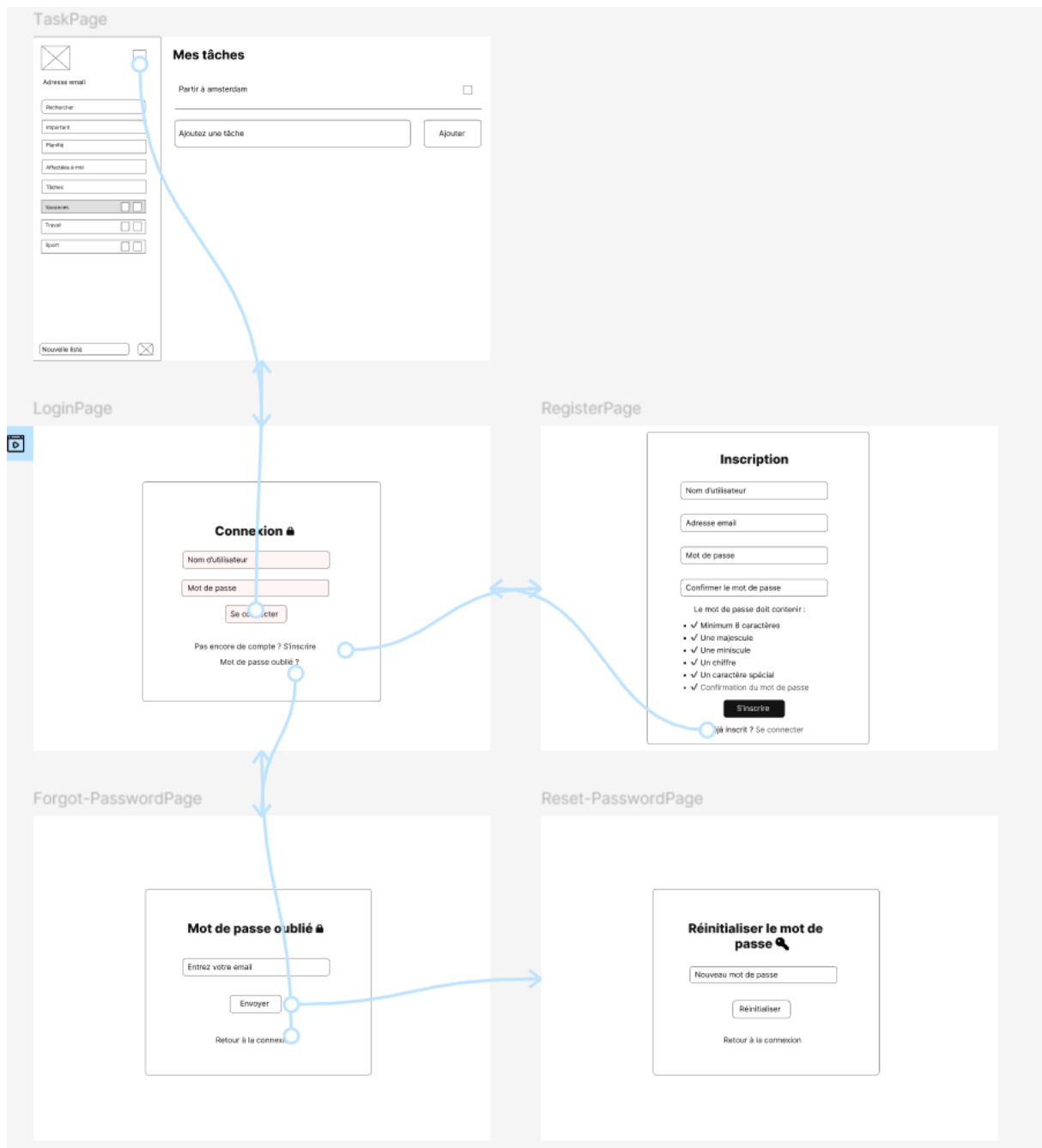


Figure 9 :Vue d'ensemble des wireframes

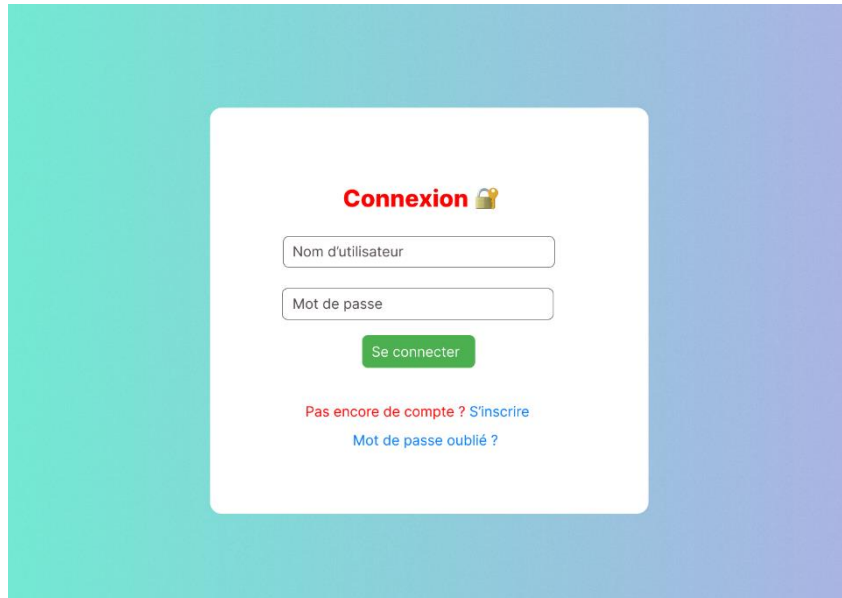
Maquette Figma (Design final)

Après validation des wireframes, une maquette graphique a été créée à l'aide de Figma, en intégrant une palette de couleurs douces (dégradé bleu/vert), une typographie lisible, et des composants UI modernes. Cette maquette vise à illustrer l'apparence réelle de l'application.

Les écrans suivants ont été conçus dans leur version finale :

Login Page

Interface centrée, minimaliste avec appel à l'action clair. Le bouton vert attire l'attention. Lien d'inscription et réinitialisation accessibles.

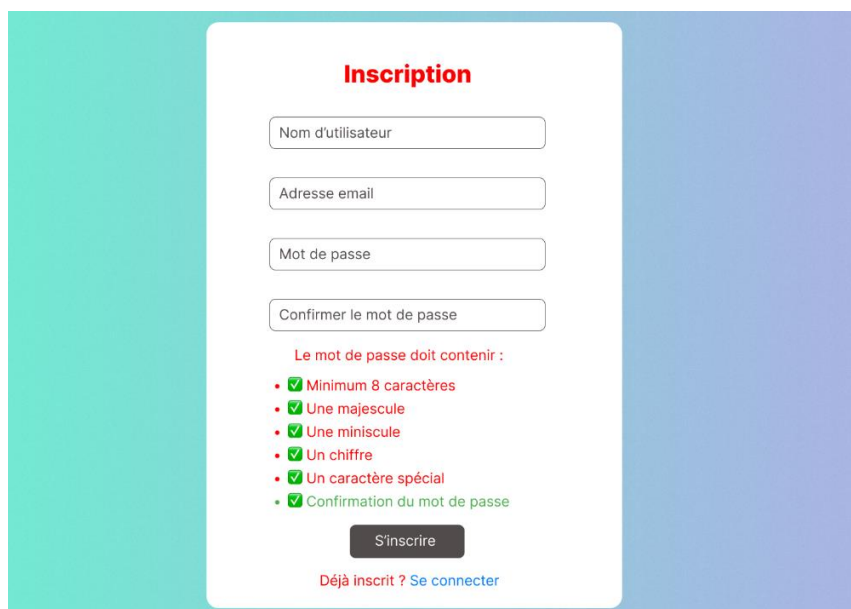


The image shows a login page mockup. It features a white central card on a blue-to-teal gradient background. The card has a red title 'Connexion' with a key icon. Below the title are two input fields: 'Nom d'utilisateur' and 'Mot de passe'. A green 'Se connecter' button is positioned below the password field. At the bottom of the card, there are two links: 'Pas encore de compte ? S'inscrire' and 'Mot de passe oublié ?'.

Figure 10 : LoginPage Maquette

Register Page

Formulaire complet avec retour d'erreurs visuel (ex : mot de passe invalide) et checklist dynamique de validation des critères de sécurité.

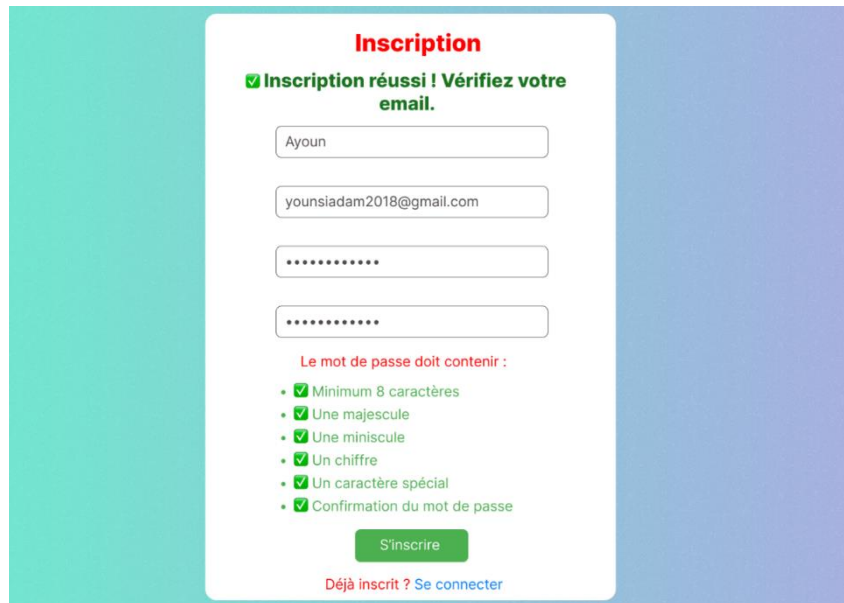


The image shows a registration page mockup. It features a white central card on a blue-to-teal gradient background. The card has a red title 'Inscription'. Below the title are four input fields: 'Nom d'utilisateur', 'Adresse email', 'Mot de passe', and 'Confirmer le mot de passe'. Below the password fields is a checklist titled 'Le mot de passe doit contenir :'. The checklist items are: 'Minimum 8 caractères', 'Une majuscule', 'Une minuscule', 'Un chiffre', 'Un caractère spécial', and 'Confirmation du mot de passe'. Each item has a green checkmark. Below the checklist is a dark grey 'S'inscrire' button. At the bottom of the card, there is a link: 'Déjà inscrit ? Se connecter'.

Figure 11 : ReisterPage Maquette

Use Case Inscription Réussie

Affichage d'un message de succès une fois le formulaire correctement complété et l'e-mail validé.

Maquette d'interface utilisateur pour l'inscription réussie. Le formulaire est centré sur un fond dégradé. Le titre 'Inscription' est en rouge. Un message de succès 'Inscription réussi ! Vérifiez votre email.' est affiché en vert. Les champs de saisie contiennent : 'Ayoun', 'younsiadam2018@gmail.com', et deux champs de mot de passe masqués par des points. Une liste de critères de validation du mot de passe est affichée avec des coches vertes. Un bouton 'S'inscrire' est en vert, et un lien 'Déjà inscrit ? Se connecter' est en rouge.

Inscription

✓ Inscription réussi ! Vérifiez votre email.

Ayoun

younsiadam2018@gmail.com

.....

.....

Le mot de passe doit contenir :

- ✓ Minimum 8 caractères
- ✓ Une majuscule
- ✓ Une minuscule
- ✓ Un chiffre
- ✓ Un caractère spécial
- ✓ Confirmation du mot de passe

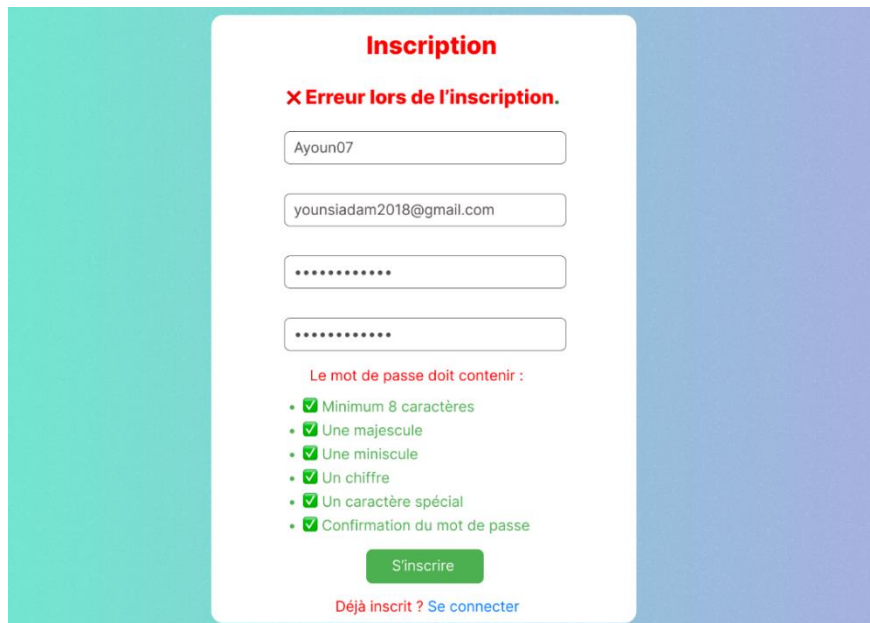
S'inscrire

Déjà inscrit ? [Se connecter](#)

Figure 12 : UseCase Register Réussie Maquette

Use Case Erreur d'inscription

En cas d'utilisation d'une adresse e-mail déjà enregistrée, une alerte s'affiche en rouge en haut du formulaire.

Maquette d'interface utilisateur pour l'inscription avec erreur. Le formulaire est centré sur un fond dégradé. Le titre 'Inscription' est en rouge. Un message d'erreur 'Erreur lors de l'inscription.' est affiché en rouge. Les champs de saisie contiennent : 'Ayoun07', 'younsiadam2018@gmail.com', et deux champs de mot de passe masqués par des points. Une liste de critères de validation du mot de passe est affichée avec des coches vertes. Un bouton 'S'inscrire' est en vert, et un lien 'Déjà inscrit ? Se connecter' est en rouge.

Inscription

✗ Erreur lors de l'inscription.

Ayoun07

younsiadam2018@gmail.com

.....

.....

Le mot de passe doit contenir :

- ✓ Minimum 8 caractères
- ✓ Une majuscule
- ✓ Une minuscule
- ✓ Un chiffre
- ✓ Un caractère spécial
- ✓ Confirmation du mot de passe

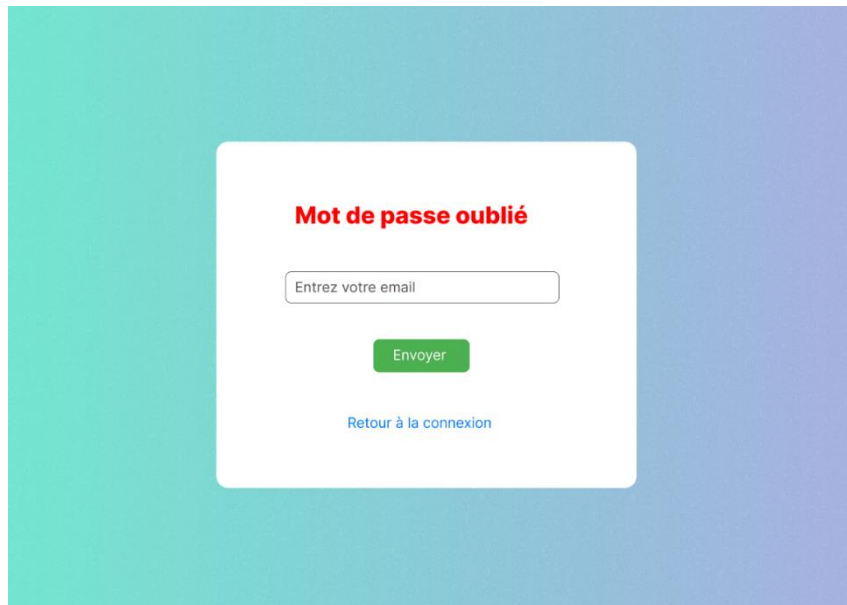
S'inscrire

Déjà inscrit ? [Se connecter](#)

Figure 13 : UseCase Register Erreur Maquette

Forgot Password Page

Interface claire avec message de confirmation après envoi.

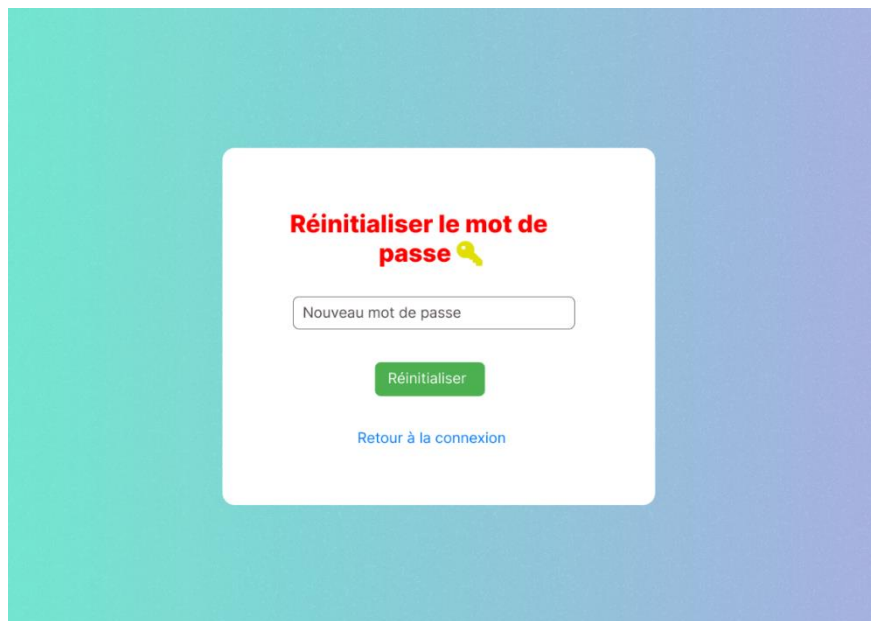


The image shows a 'Forgot Password' page mockup. It features a white rounded rectangle centered on a teal-to-blue gradient background. Inside the rectangle, the title 'Mot de passe oublié' is written in red. Below it is a text input field with the placeholder 'Entrez votre email'. Under the input field is a green button labeled 'Envoyer'. At the bottom of the rectangle is a blue link that says 'Retour à la connexion'.

Figure 14 : ForgotPasswordPage Maquette

Reset Password Page

Formulaire simplifié pour définir un nouveau mot de passe.



The image shows a 'Reset Password' page mockup. It features a white rounded rectangle centered on a teal-to-blue gradient background. Inside the rectangle, the title 'Réinitialiser le mot de passe' is written in red, followed by a yellow key icon. Below it is a text input field with the placeholder 'Nouveau mot de passe'. Under the input field is a green button labeled 'Réinitialiser'. At the bottom of the rectangle is a blue link that says 'Retour à la connexion'.

Figure 15 : ResetPasswordPage Maquette

Task Page (Tableau de bord)

Interface utilisateur avec barre latérale, icônes, et filtrage interactif. Arrière-plan bleu et retour visuel sur l'état des tâches.

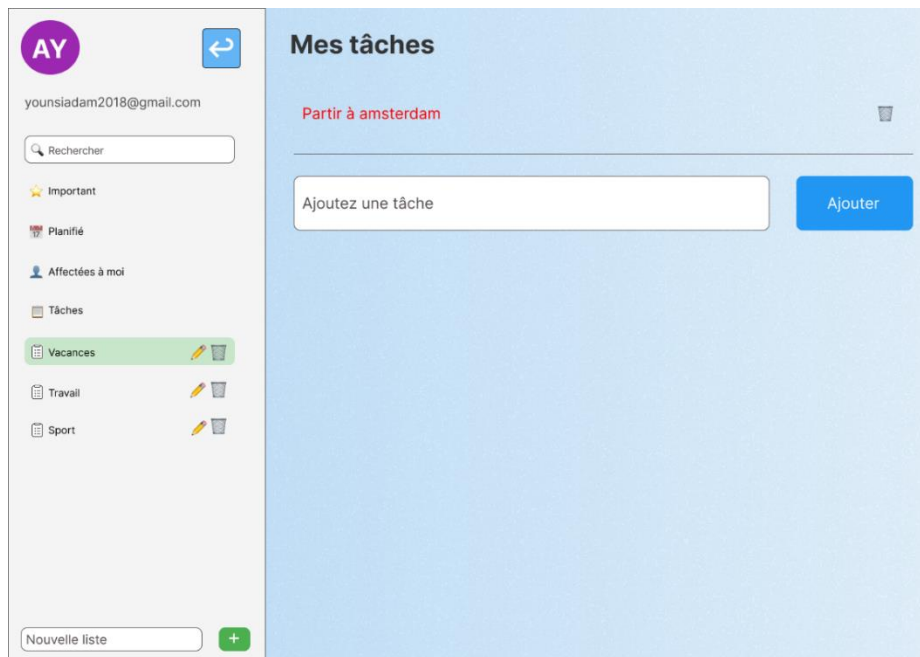


Figure 16 : TaskPage Maquette

Responsive Design

Une maquette mobile de la page de connexion a été également réalisée pour démontrer l'adaptabilité de l'interface sur smartphones. La disposition verticale est conservée, et l'ergonomie reste optimale sur petits écrans.



Figure 17 : Responsive Maquette

Vue d'ensemble Figma avec liaisons

Une dernière capture globale regroupe tous les écrans et leurs connexions sous forme de flux. Cette vue met en lumière les transitions d'un écran à un autre selon les actions de l'utilisateur :

- Inscription → Confirmation ou Erreur
- Connexion → Tableau de bord
- Mot de passe oublié → Réinitialisation

Cette approche visuelle complète renforce la lisibilité du parcours utilisateur et démontre le souci d'ergonomie et d'accessibilité dans la conception de l'application.

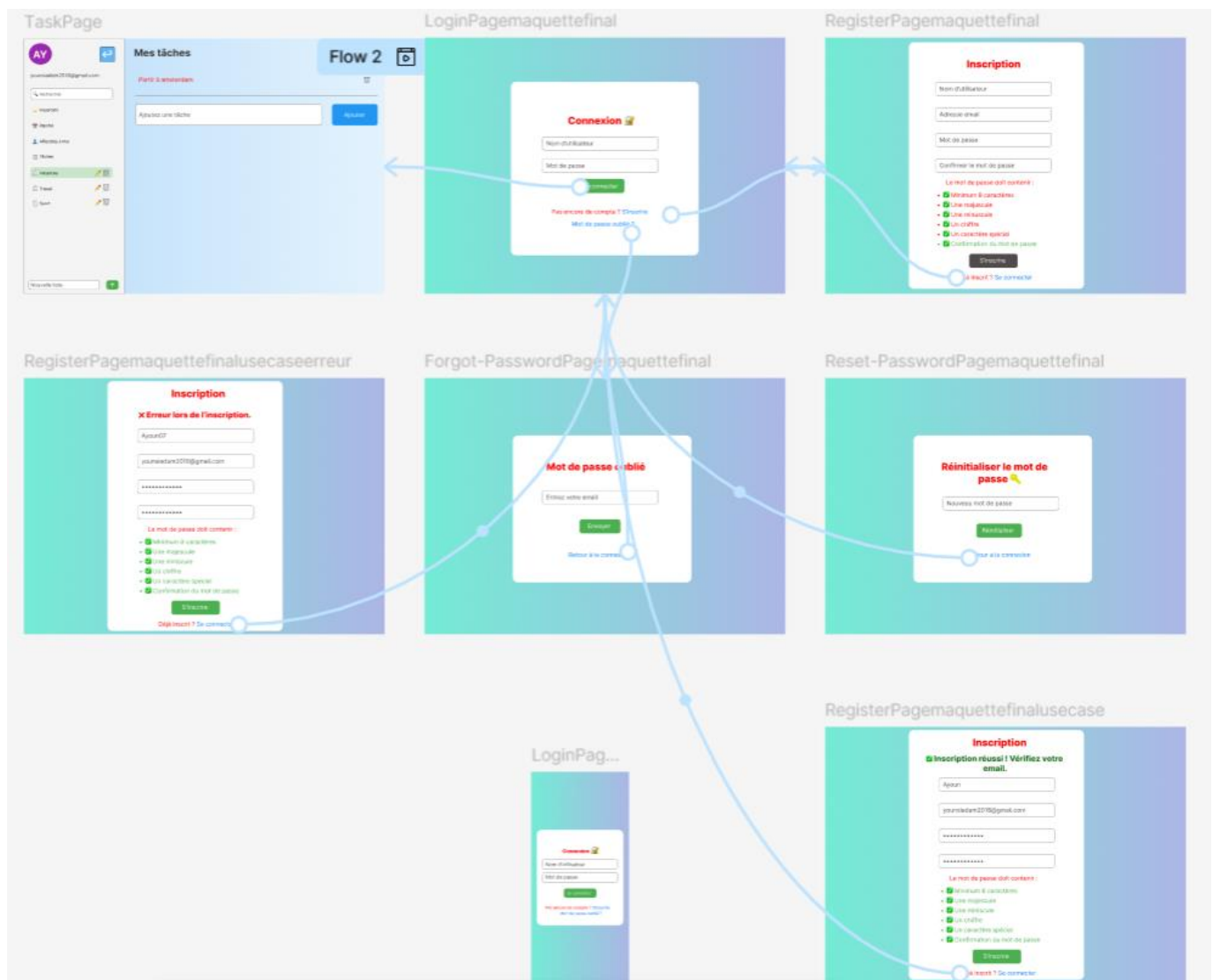


Figure 20 : Vue d'ensemble des maquettes

Architecture logicielle

Programmation Orientée Objet (POO)

L'ensemble du backend du projet a été conçu en Java, en respectant les principes fondamentaux de la Programmation Orientée Objet (POO). Chaque entité métier – notamment l'utilisateur, la liste et la tâche – est modélisée sous forme de classe, ce qui permet une instantiation claire et structurée.

Les grands principes de la POO sont rigoureusement appliqués :

- **Encapsulation** : les attributs sont privés (private) et accédés via des accesseurs (getters) et des mutateurs (setters), garantissant ainsi la protection des données.

```
1 package com.todo.backend.model;
2
3 import jakarta.persistence.*;
4 import java.util.UUID;
5
6
7
8 @Entity
9 @Table(name = "users")
10 public class User {
11
12     @Id
13     @GeneratedValue(strategy = GenerationType.IDENTITY)
14     private Long id;
15
16     @Column(unique = true)
17     private String username;
18
19
20     private String password;
21
22     @Column(unique = true)
23     private String email;
24
25     @Column(name = "reset_token")
26     private String resetToken;
27
28     private boolean enabled = false;
29
30     private String confirmationToken = UUID.randomUUID().toString();
31
32     // Getters & Setters
33
34     public Long getId() {
35         return this.id;
36     }
```

Figure 21 : Encapsulation des données de la tables users avec getters et setters

- **Héritage** : les services implémentent des interfaces (comme UserDetailsService) pour générer des comportements spécifiques tout en bénéficiant d'un contrat standard.
- **Abstraction** : la logique est séparée entre les couches (modèle, service, contrôleur), facilitant la compréhension et la maintenance du code.
- **Polymorphisme** : les services sont injectés via des interfaces (à l'aide de l'annotation @Autowired), permettant la substitution et les tests unitaires faciles.

Application des principes SOLID

Les principes SOLID, piliers de la conception logicielle, sont appliqués de manière cohérente dans l'ensemble du projet :

- **S (Single Responsibility)** : chaque classe remplit une fonction précise (ex : TaskController se limite à la gestion des endpoints des tâches).

```

17 @RestController
18 @RequestMapping("/api/tasks")
19 @CrossOrigin
20 public class TaskController {
21
22     private final TaskRepository taskRepo;
23     private final UserRepository userRepo;
24     private final ListRepository listRepo;
25
26     // Injection des dépendances via le constructeur
27     public TaskController(TaskRepository taskRepo, UserRepository userRepo, ListRepository listRepo) {
28         this.taskRepo = taskRepo;
29         this.userRepo = userRepo;
30         this.listRepo = listRepo;
31     }
32
33     // Récupère les tâches d'une liste donnée, seulement si elle appartient à l'utilisateur connecté
34     @GetMapping("/list/{listId}")
35     public List<Task> getTasksByList(@PathVariable Long listId, Authentication auth) {
36         String username = auth.getName();
37         User user = userRepo.findByUsername(username).orElseThrow();
38         ListEntity list = listRepo.findById(listId).orElseThrow();
39
40         if (!list.getUser().equals(user)) {
41             throw new ResponseStatusException(HttpStatus.FORBIDDEN, reason: "Accès interdit.");
42         }
43
44         return taskRepo.findByListAndUser(list, user);
45     }

```

Figure 22 : Contrôle uniquement la couche API

- **O (Open/Closed)** : les services peuvent être étendus sans être modifiés, grâce à l'utilisation d'interfaces.

- **L (Liskov Substitution)** : le CustomUserDetailsService respecte pleinement le contrat de l'interface UserDetailsService de Spring Security.

```

11 @Service
12 public class CustomUserDetailsService implements UserDetailsService {
13
14     @Autowired
15     private UserRepository userRepository;
16
17     @Override
18     public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

```

Figure 23 : Respect de Liskov

- **I (Interface Segregation)** : les interfaces JPA sont spécifiques et ne forcent pas l'implémentation de méthodes inutiles.
- **D (Dependency Inversion)** : l'architecture repose sur l'injection de dépendances, favorisant un code faiblement couplé.

Organisation des packages – Architecture en couches

Le projet suit une architecture MVC évoluée avec une organisation par fonctionnalité. Chaque couche est clairement délimitée :

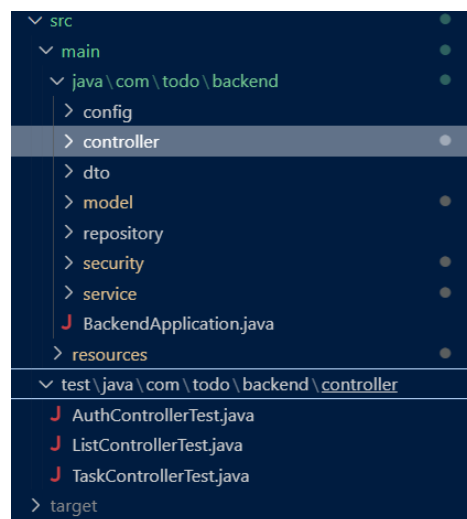


Figure 24 : Organisation propre par couche

- **Controller** : AuthController, TaskController, ListController reçoivent les requêtes HTTP et orchestrent les appels aux services.

- **Service** : EmailService, CustomUserDetailsService contiennent la logique métier et le traitement des données.
- **Model** : User, Task, ListEntity représentent les entités persistées via JPA.
- **Repository** : UserRepository, TaskRepository, ListRepository assurent l'accès aux données.
- **DTO** : LoginRequest, RegisterRequest protègent les échanges de données entre frontend et backend.
- **Security** : JwtUtil, JwtAuthFilter, WebSecurityConfig assurent l'authentification et la sécurité.
- **Middleware/Config** : CorsConfig, JwtAuthFilter gèrent les requêtes HTTP entrantes et la configuration CORS.

Sécurité – Authentification, Hashing et Token

L'authentification est de type **stateless** grâce à l'utilisation de **JSON Web Tokens (JWT)**. Aucun état n'est conservé sur le serveur entre deux requêtes.

Le mot de passe de chaque utilisateur est hashé avec l'algorithme **BCrypt**, reconnu pour sa robustesse et ses délais adaptatifs. Cela est conforme aux recommandations RGPD et OWASP :

```

62  @Bean
63  public PasswordEncoder passwordEncoder() {
64      return new BCryptPasswordEncoder();
65  }

```

Figure 25 : Illustration du hashing sécurisé des mots de passe.

Ainsi, lors de l'inscription, le mot de passe est hashé avant d'être enregistré en base de données. Lors de la connexion, il est comparé au hash existant, sans jamais stocker le mot de passe en clair.

Un lien de confirmation par email est également prévu pour valider l'adresse de l'utilisateur, ainsi qu'un mécanisme de réinitialisation du mot de passe via lien sécurisé.

Middleware – Filtres personnalisés

Un composant de type **middleware** a été développé sous forme de filtre HTTP `JwtAuthFilter`. Ce filtre :

- Intercepte chaque requête HTTP entrante.
- Vérifie la présence et la validité du token JWT.

```
24 | // Génère un token JWT en y mettant le nom d'utilisateur comme "subject"
25 | public String generateToken(UserDetails userDetails) {
26 |     return Jwts.builder()
27 |         .setSubject(userDetails.getUsername())
28 |         .setIssuedAt(new Date())
29 |         .setExpiration(new Date(System.currentTimeMillis() + EXPIRATION_TIME))
30 |         .signWith(key, SignatureAlgorithm.HS256)
31 |         .compact();
32 | }
```

Figure 26 : Token signé, expiration

- Rejette ou accepte l'accès à la ressource.
- Injecte l'utilisateur dans le contexte de Spring Security en cas de succès.

Ce fonctionnement est équivalent à un middleware dans d'autres frameworks comme Laravel ou Express.js.

REST et séparation des responsabilités

Le backend repose sur une architecture **RESTful** découpée. Il n'y a pas de gestion de vues HTML : la partie **frontend React** joue le rôle de la vue (View), interrogeant uniquement des APIs JSON via HTTP.

Définitions et concepts clés

Qu'est-ce qu'un token ?

Un **token** est une chaîne de caractères générée pour identifier de façon sécurisée un utilisateur. Il est souvent utilisé dans les applications web pour authentifier les utilisateurs de manière stateless. Le **JWT** (JSON Web Token) encode des informations (comme l'identifiant de l'utilisateur) de manière signée mais non chiffrée.

Sécurité du mot de passe

Il est impératif de ne jamais stocker de mot de passe en clair. Le hashage avec **BCrypt** assure que même en cas de fuite de base de données, les mots de passe restent difficilement exploitables. Un bon mot de passe hashé est salé (ajout d'une chaîne aléatoire) et lent à calculer pour décourager les attaques brutales.

Lien de validation / mot de passe oublié

Lorsqu'un utilisateur s'inscrit, un **email de confirmation** est envoyé avec un lien d'activation. En cas de mot de passe oublié, un lien temporaire de réinitialisation est envoyé, expirant après un délai court.

Requêtes via Postman

Les requêtes Postman permettent de tester les APIs REST du **backend** sans passer par l'interface frontend. Elles sont essentielles pour vérifier le comportement des routes (POST, GET, PUT, DELETE), les statuts HTTP, et la gestion des erreurs.

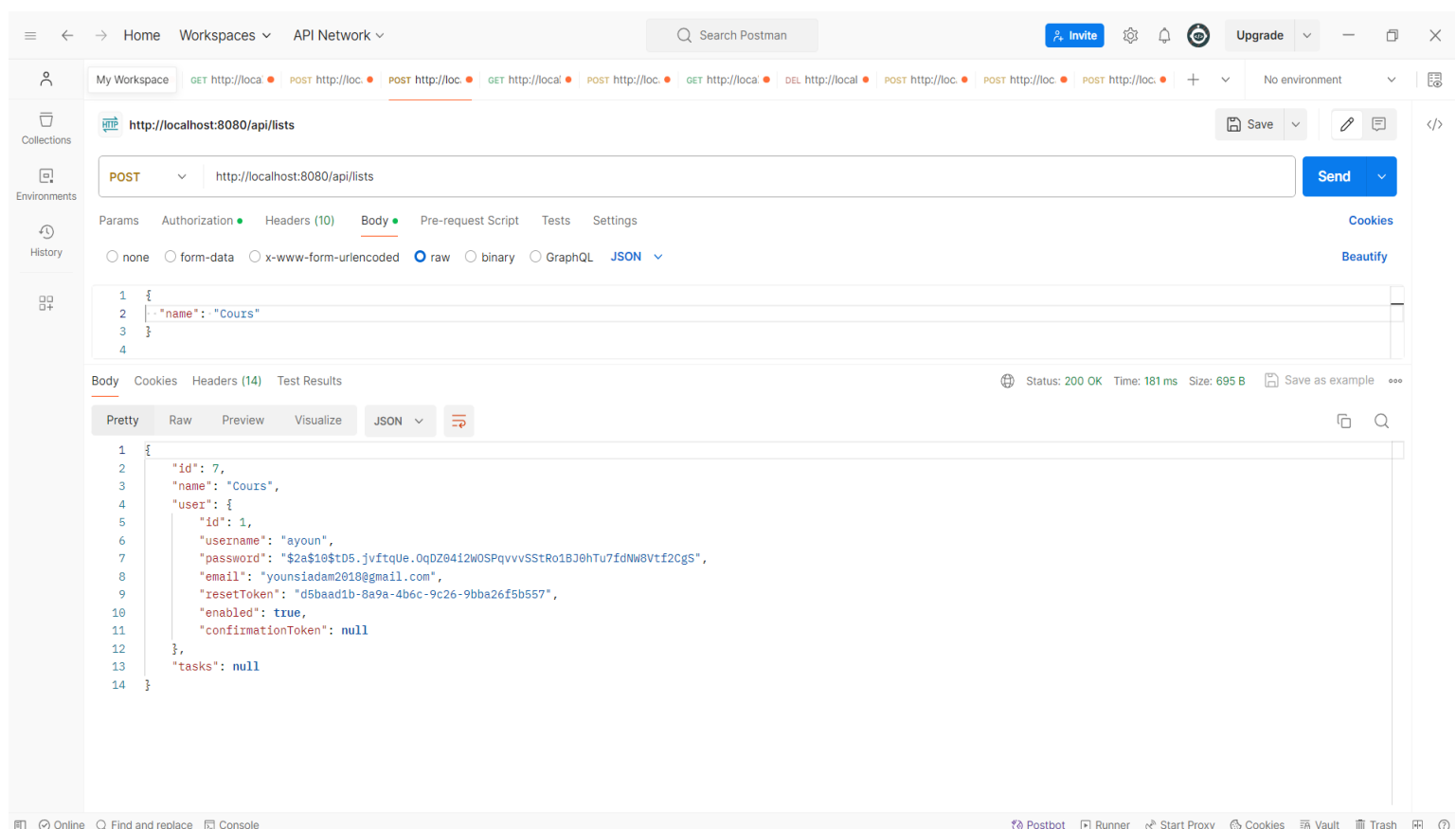


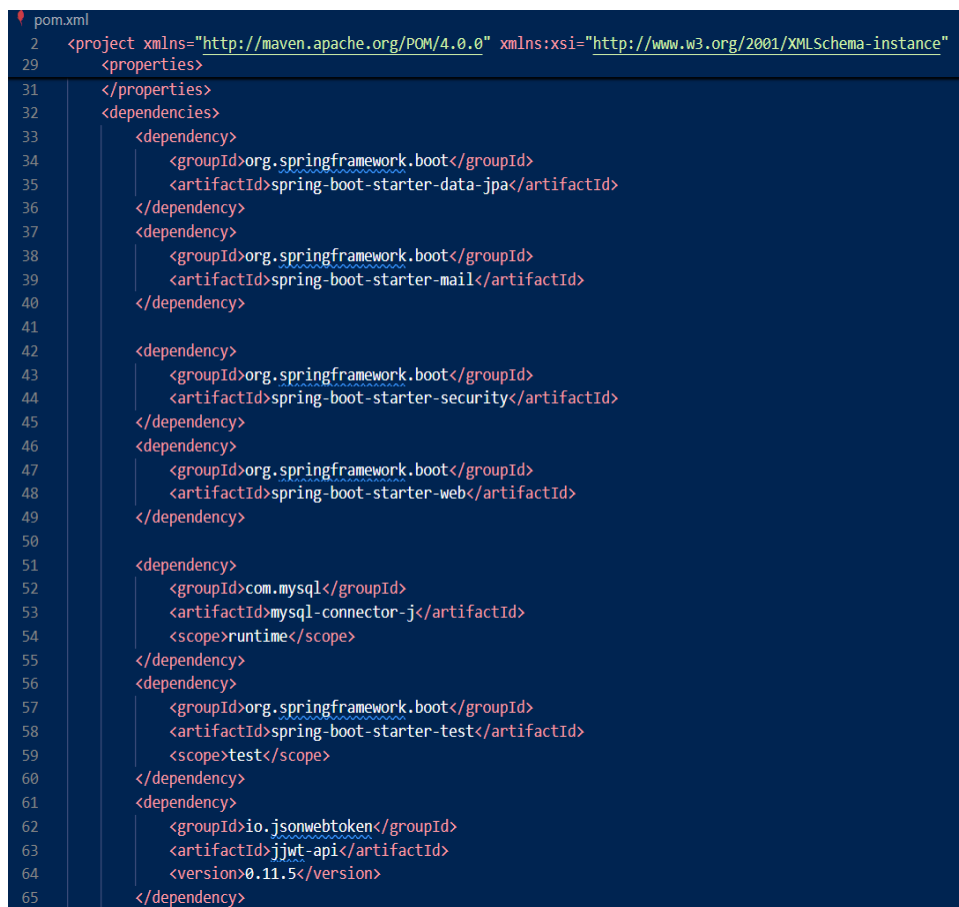
Figure 27 : Requêtes API sécurisé avec le token JWT dans Authorization Bearer token

À quoi sert Maven ?

Maven est un outil de gestion de projet et de dépendances. Il permet de compiler, tester et emballer une application Java. Il gère les bibliothèques via un fichier central : pom.xml.

Qu'est-ce que le fichier pom.xml ?

Il s'agit du fichier de configuration principal de Maven. Il définit les dépendances (Spring Boot, MySQL, JWT...), les plugins de build, les versions, les profils de déploiement et les configurations globales du projet.



```
1 pom.xml
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
29 </properties>
31 </properties>
32 <dependencies>
33   <dependency>
34     <groupId>org.springframework.boot</groupId>
35     <artifactId>spring-boot-starter-data-jpa</artifactId>
36   </dependency>
37   <dependency>
38     <groupId>org.springframework.boot</groupId>
39     <artifactId>spring-boot-starter-mail</artifactId>
40   </dependency>
41   <dependency>
42     <groupId>org.springframework.boot</groupId>
43     <artifactId>spring-boot-starter-security</artifactId>
44   </dependency>
45   <dependency>
46     <groupId>org.springframework.boot</groupId>
47     <artifactId>spring-boot-starter-web</artifactId>
48   </dependency>
49   <dependency>
50     <groupId>com.mysql</groupId>
51     <artifactId>mysql-connector-j</artifactId>
52     <scope>runtime</scope>
53   </dependency>
54   <dependency>
55     <groupId>org.springframework.boot</groupId>
56     <artifactId>spring-boot-starter-test</artifactId>
57     <scope>test</scope>
58   </dependency>
59   <dependency>
60     <groupId>io.jsonwebtoken</groupId>
61     <artifactId>jjwt-api</artifactId>
62     <version>0.11.5</version>
63   </dependency>
64 </dependencies>
```

Figure 28 : Gestion des dépendances

Qu'est-ce qu'un framework ?

Un **framework** est un ensemble de bibliothèques et de conventions destiné à faciliter le développement d'applications. Spring Boot, par exemple, fournit une structure préétablie, des annotations prêtes à l'emploi et une gestion automatique des composants (beans, services, etc.).

Qu'est-ce qu'un JWT ?

Le **JWT (JSON Web Token)** est un standard d'authentification sous forme de token. Il contient trois parties encodées en base64 :

1. Header (type + algorithme de signature)
2. Payload (données comme username, date d'expiration...)
3. Signature (validation de l'intégrité du message)

Ce token est signé avec une clé secrète et transmis dans chaque requête (via l'en-tête Authorization) pour permettre l'accès aux routes protégées.

Développement Backend

Technologies principales

Le backend a été développé en **Java 17** avec le framework **Spring Boot**, en respectant le modèle MVC. L'architecture et les bonnes pratiques (POO, SOLID, REST, stateless) ont été détaillées dans la section précédente.

Dans cette section, nous allons approfondir **le fonctionnement opérationnel** des différents composants, notamment les **contrôleurs**, les **services métier**, la **gestion de la sécurité (authentification/autorisation)** et les **tests backend**.

Contrôleurs et services

Les contrôleurs exposent les endpoints REST de l'application. Ils reçoivent les requêtes HTTP, valident les données, puis délèguent la logique métier aux services.

Exemples :

- **AuthController** : gère l'inscription, la connexion, la réinitialisation du mot de passe.
- **TaskController** : permet de créer, afficher ou supprimer des tâches.
- **ListController** : gère les listes de tâches associées à chaque utilisateur.

Chaque contrôleur est lié à un service métier (**CustomUserDetailsService**, **EmailService...**), ce qui garantit une séparation claire des responsabilités.

Authentification : JWT et sécurité des routes

Le système d'authentification est **stateless**. Il repose sur des **JSON Web Tokens (JWT)**, générés lors de la connexion et utilisés ensuite pour sécuriser toutes les routes de l'API.

Fonctionnement :

- Lors de la connexion, l'utilisateur reçoit un token signé avec une clé secrète.
- Ce token est ensuite envoyé dans chaque requête via l'en-tête **Authorization: Bearer <token>**.
- Le filtre personnalisé **JwtAuthFilter** vérifie le token à chaque requête entrante.
- Si le token est valide, l'utilisateur est injecté dans le contexte de sécurité Spring (**SecurityContextHolder**).

Toutes les routes sensibles (accès aux tâches, modifications, etc.) sont protégées via **Spring Security**.

Sécurité côté backend

- **Validation des entrées** : Les DTO (RegisterRequest, LoginRequest) sont validés via des annotations @NotNull, @Email, etc.
- **Hashage des mots de passe** : Utilisation de l'algorithme **BCrypt** (via BCryptPasswordEncoder).
- **Protection des routes** : Les endpoints REST sont restreints aux utilisateurs authentifiés, grâce à la configuration dans WebSecurityConfig.
- **Expiration des tokens** : Le JWT possède une durée de vie limitée (10h) pour renforcer la sécurité.

Tests backend

Les tests backend ont été menés en deux temps :

- **Tests unitaires automatisés**
- **Tests fonctionnels via Postman**

Tests unitaires Spring Boot

Des tests unitaires ont été écrits pour vérifier le bon fonctionnement des contrôleurs REST principaux.

Ces tests utilisent le concept de **mocking**, grâce à l'utilisation de l'annotation @MockBean pour simuler le comportement des services sans devoir réellement interroger la base de données ou les couches internes.

Qu'est-ce que le mocking ?

Le **mocking** consiste à créer de fausses implémentations de classes ou de services dans le but d'isoler le code à tester. Cela permet de simuler le comportement d'un service externe, comme une base de données ou un envoi d'e-mail, et ainsi de tester unitairement une méthode ou un composant spécifique.

Dans ce projet, le mocking a été utilisé pour injecter des réponses personnalisées dans les services de type UserService, TaskService, etc., et valider les résultats attendu des contrôleurs.

Cahier des recettes

(planification des tests)

Tests Backend

Test n°	Nom du test	Contrôleur testé	Objectif du test	Résultat attendu	Statut
B1	testLoginSuccess	AuthController	Vérifier que le login retourne un token valide	HTTP 200 + JWT dans la réponse	Réussi
B2	testLoginFailInvalidUser	AuthController	Vérifier que le login échoue avec mauvais user	HTTP 401 ou message d'erreur	Réussi
B3	testRegisterUser	AuthController	Vérifier l'inscription d'un nouvel utilisateur	HTTP 201 ou message de succès	Réussi
B4	testGetUserLists	ListController	Récupérer les listes de l'utilisateur connecté	Liste JSON des listes	Réussi
B5	testCreateList	ListController	Créer une nouvelle liste	HTTP 201 + liste enregistrée	Réussi
B6	testGetUserTasks	TaskController	Récupérer les tâches d'une liste donnée	Liste JSON de tâches	Réussi
B7	testAddTask	TaskController	Ajouter une nouvelle tâche	HTTP 201 + tâche créée	Réussi

Ces tests ont été réalisés dans les fichiers AuthControllerTest.java, ListControllerTest.java, et TaskControllerTest.java. Chaque test simule une requête REST avec les données nécessaires, puis vérifie la réponse (statut HTTP, contenu JSON).

Tests via Postman

Les tests manuels ont été complétés avec **Postman** pour valider :

- Le comportement des routes API (statuts, retours JSON)
- L'intégration du token JWT dans les headers
- La protection des routes sensibles

Développement Frontend

Technologies et structure

Le frontend a été développé en **React** avec **TypeScript**, dans une architecture composant modulaire. Le typage fort a permis de réduire les erreurs en temps de compilation et de faciliter le débogage durant le développement.

L'application repose sur un routage par composant et sur une logique claire de gestion de l'état et des appels API. Les styles sont gérés de manière séparée dans des fichiers CSS propres à chaque composant.

Structure des composants

La structure fonctionnelle se compose des fichiers suivants :

- App.tsx : composant racine, point d'entrée de l'application avec le routing et le layout global.
- Login.tsx : composant de connexion avec formulaire.
- Register.tsx : création de compte utilisateur.
- ForgotPassword.tsx / ResetPassword.tsx : gestion du mot de passe.
- TaskPage.tsx : tableau de bord principal avec les listes et tâches.
- Sidebar.tsx, MainContent.tsx, Task.tsx : composants UI réutilisables.

Les styles CSS sont séparés (.css liés à chaque composant).

Appels API

Les appels à l'API backend sont centralisés dans api.ts, ce qui permet une bonne maintenabilité du code.

Exemple d'appel :

```
4 | const BASE_URL = process.env.REACT_APP_API_URL; // depuis .env
5 |
6 | const axiosInstance = axios.create({
7 |   baseURL: BASE_URL,
8 |   headers: {
9 |     "Content-Type": "application/json",
10 |   },
11 | });
12 | // You, 2 weeks ago • ajout du frontend dans le repo ...
13 | axiosInstance.interceptors.request.use((config) => {
14 |   const token = localStorage.getItem("token");
15 |   if (token && config.headers) {
16 |     config.headers.Authorization = `Bearer ${token}`;
17 |   }
18 |   return config;
19 | });
```

Figure 29 : Appel API avec token

Chaque requête inclut le token JWT depuis le localStorage ou le contexte utilisateur.

Gestion de l'état et des données

La gestion de l'état repose sur les hooks **useState**, **useEffect**, et les **props** passées entre composants.

- useState permet de stocker les données (formulaires, listes, tâches...)
- useEffect est utilisé pour charger les données dès le montage du composant
- Les props servent à transmettre les données ou callbacks vers les composants enfants

Extrait typique :

```
20  const [tasks, setTasks] = useState<Task[]>([]);
```

Responsive design

L'interface est totalement adaptée aux différents formats d'écran. Une maquette mobile a été conçue et validée.

- Les composants comme Sidebar ou TaskPage s'adaptent à la largeur de l'écran.
- Le design mobile simplifie la navigation (colonnes empilées, bouton burger, etc.)
- Utilisation de **media queries** dans le CSS pour adapter les éléments graphiques.

Respect de la maquette Figma

La maquette graphique Figma validée en amont a été fidèlement reproduite :

- Respect de la palette de couleurs et des espacements
- Mêmes comportements sur les messages d'erreur et confirmations
- Composants visuellement identiques à ceux du prototype Figma

Tests frontend

Tests unitaires avec Jest

Un test unitaire est présent dans `__tests__/Login.test.tsx` pour vérifier le bon rendu du formulaire de connexion :

Cahier des recettes

(planification des tests)

Tests Frontend

Test n°	Nom du test	Composant testé	Objectif du test	Résultat attendu	Statut
F1	affiche le formulaire login	Login.tsx	Vérifier que les champs login et mot de passe s'affichent	Inputs et bouton visibles	Réussi
F2	affiche erreur si login invalide	Login.tsx	Simuler un login invalide et vérifier le message d'erreur	Texte 'Identifiants invalides.' visible	Réussi
F3	stocke token si login réussi	Login.tsx	Vérifier que le token est enregistré dans localStorage	localStorage contient 'token'	Réussi

Ces tests permettent de valider l’affichage, la validation et le comportement du formulaire.

Test fonctionnel avec Cypress

Un test Cypress a été ajouté dans `cypress/e2e/createLogin.cy.js` pour valider tout le parcours utilisateur :

- Connexion utilisateur
- Navigation vers le dashboard
- Création d'une nouvelle liste de tâches
- Vérification de l'affichage en réel

Cypress permet d'émuler le comportement de l'utilisateur en conditions réelles.

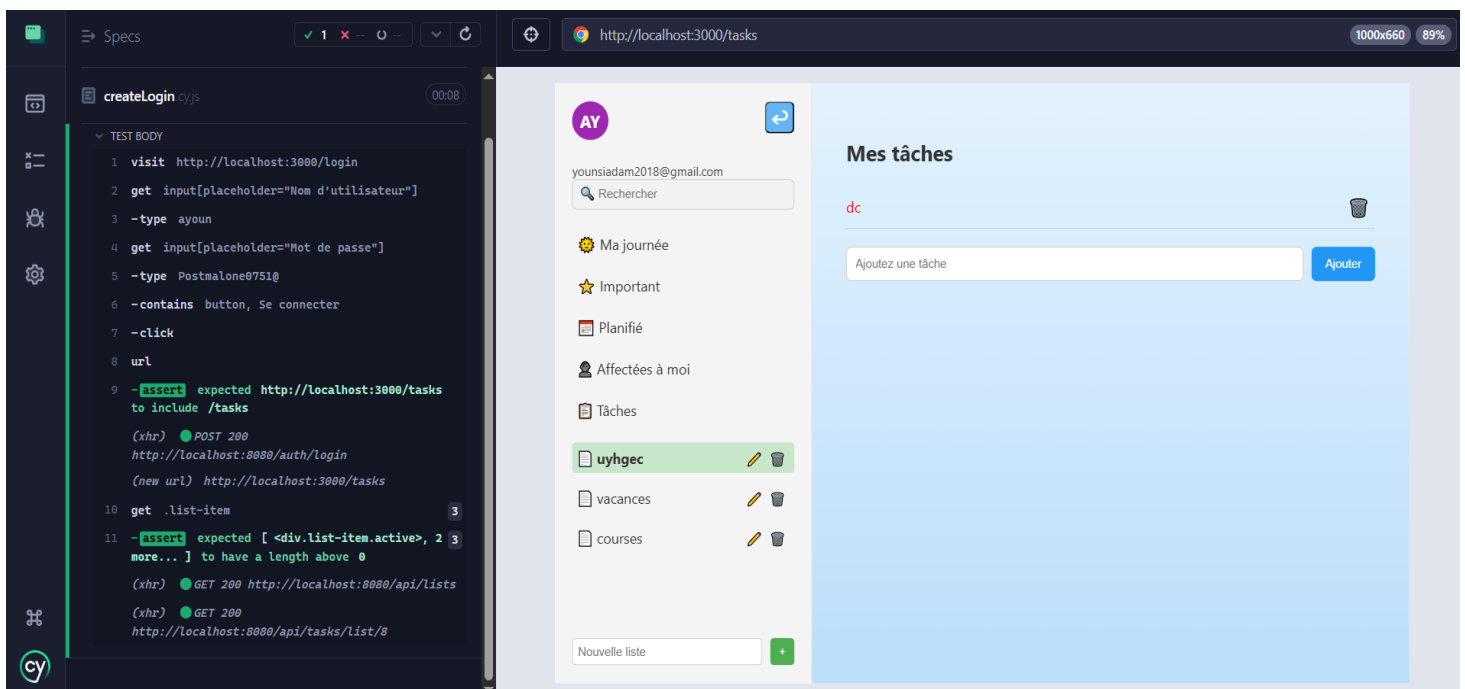


Figure 30 : `createLogin.cy.js` avec cypress

Base de données

MCD et MLD modélisés

La base de données de l'application repose sur une architecture **relationnelle** et a été intégralement modélisée en suivant la méthodologie classique :

- MCD (Modèle Conceptuel de Données)
- MLD (Modèle Logique de Données)
- MPD (Modèle Physique de Données)

Le **MCD** a été construit avec l'outil diagrams.net (draw.io) en mettant en évidence :

- Les **entités** métier : users, list_entity, tasks
- Les **relations métiers** : owns, contains, creates
- Les **cardinalités** et les **attributs** sans typage technique

Le **MLD** et le **MPD** ont été générés via **MySQL Workbench** à partir de la structure réelle de la base.

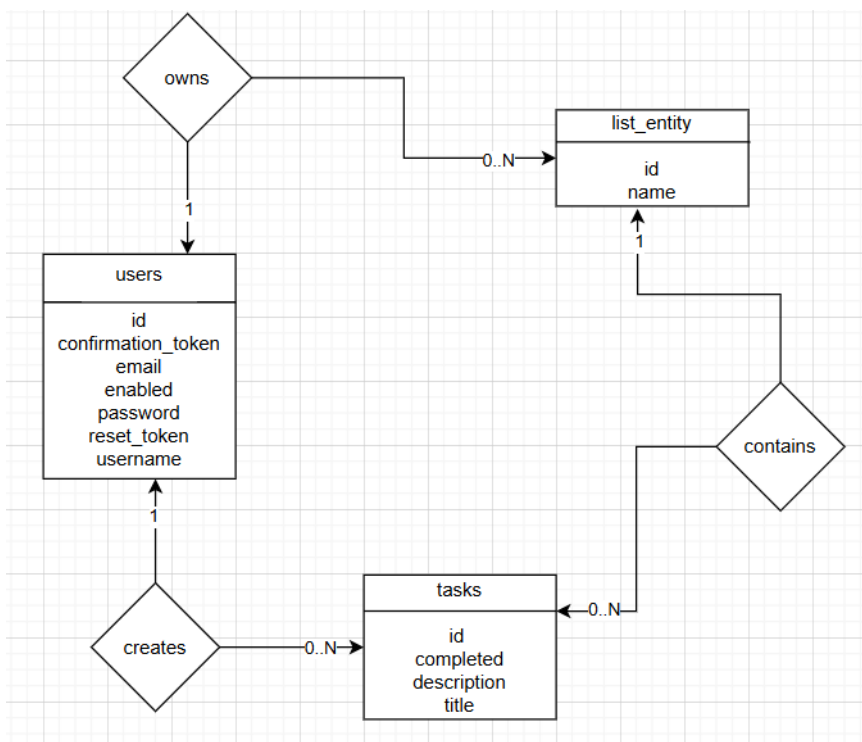


Figure 31 : Diagramme MCD

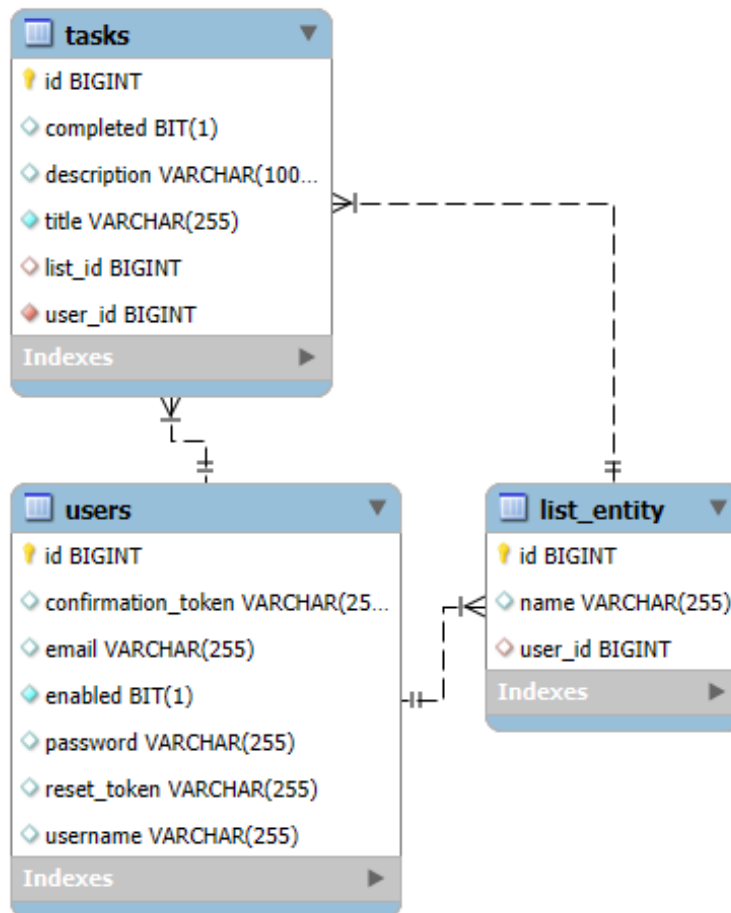


Figure 32 : Diagramme MLD

```

21  USE `tododb` ;
22
23  -----
24  -- Table `tododb`.`users`
25  -----
26  CREATE TABLE IF NOT EXISTS `tododb`.`users` (
27    `id` BIGINT NOT NULL AUTO_INCREMENT,
28    `confirmation_token` VARCHAR(255) NULL DEFAULT NULL,
29    `email` VARCHAR(255) NULL DEFAULT NULL,
30    `enabled` BIT(1) NOT NULL,
31    `password` VARCHAR(255) NULL DEFAULT NULL,
32    `reset_token` VARCHAR(255) NULL DEFAULT NULL,
33    `username` VARCHAR(255) NULL DEFAULT NULL,
34    PRIMARY KEY (`id`),
35    UNIQUE INDEX `UK6dotkott2kjsp8vw4d0m25fb7` (`email` ASC) VISIBLE,
36    UNIQUE INDEX `UKr43af9ap4edm43mmtg01oddj6` (`username` ASC) VISIBLE)
37  ENGINE = InnoDB
38  AUTO_INCREMENT = 4
39  DEFAULT CHARACTER SET = utf8mb4
40  COLLATE = utf8mb4_0900_ai_ci;
  
```

Figure 33 : Extrait du MPD

Base relationnelle MySQL + Docker

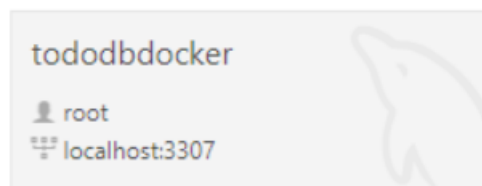
La base de données fonctionne sous **MySQL**, déployée en conteneur via **Docker**. Elle est exposée en **port externe 3307**, différent du port MySQL par défaut (3306), ce qui permet de l'isoler proprement des services locaux.

Le conteneur est lancé avec une commande type :

```
3  ∨ services:
    ∟ ∟ Run Service
4  ∨ db:
5      image: mysql:8.0
6      container_name: mysql-db
7      restart: always
8  ∨ environment:
9      MYSQL_ROOT_PASSWORD: Postmalone0751@
10     MYSQL_DATABASE: tododb
11 ∨ ports:
12     - "3307:3306" # Externe:Interne
13 ∨ volumes:
14     - db-data:/var/lib/mysql
15 ∨ networks:
16     - app-network
```

Figure 34 : DB sous docker

Cela permet à MySQL Workbench de se connecter via localhost:3307 pour la gestion graphique.



Tables principales :

- users : stocke les données utilisateurs (email, mot de passe, tokens...)
- list_entity : représente les listes créées par chaque utilisateur
- tasks : représente les tâches, liées à une liste et à un utilisateur

Les relations entre les tables sont conformes aux bonnes pratiques de modélisation, avec **clés étrangères explicites** et **intégrité référentielle assurée**.

Backup / restauration

Une sauvegarde complète de la base a été générée via **l'export SQL** dans MySQL Workbench (.sql).

Ce fichier peut être utilisé pour :

- Restaurer intégralement la structure de la base (MPD)
- Réinjecter les données de test (jeu d'essai)

La procédure standard consiste à :

- Lancer MySQL Workbench
- Créer une nouvelle base (tasklist par exemple)
- Aller dans File > Run SQL Script
- Importer le fichier .sql

En quelques secondes, toutes les tables, clés, et relations sont reconstruites.

Jeu d'essai utilisé

Un jeu d'essai a été inséré pour simuler un comportement réaliste de l'application :

- 2 utilisateurs fictifs (comptes de test)
- Plusieurs listes (vacances, travail, sport...)
- Des tâches associées à chaque liste, avec statuts (completed, description, etc.)

Ces données ont permis :

- De tester la cohérence des jointures
- De valider les endpoints backend (via Postman)
- D'afficher des listes dynamiques côté frontend

CI/CD & Déploiement

Introduction aux outils CI/CD

Qu'est-ce que Docker ?

Docker est une plateforme de conteneurisation qui permet d'exécuter des applications dans un environnement isolé appelé **conteneur**. Un conteneur inclut l'application, ses dépendances, ses librairies et sa configuration, garantissant un fonctionnement identique peu importe la machine hôte.

Qu'est-ce qu'un Dockerfile ?

Un **Dockerfile** est un fichier texte contenant les instructions nécessaires pour construire une image Docker. Il décrit :

- L'image de base (ex: node, openjdk)
- Les fichiers à copier
- Les dépendances à installer (npm install, mvn install)
- La commande à exécuter au démarrage du conteneur (CMD, ENTRYPOINT)

Qu'est-ce que docker-compose ?

Docker Compose est un outil qui permet de définir et de lancer plusieurs conteneurs Docker à partir d'un seul fichier docker-compose.yml. On y retrouve souvent des services comme :

- Une base de données (MySQL, PostgreSQL)
- Une API backend (Java, Node...)
- Un frontend (React, Angular)

Chaque service est indépendant mais peut communiquer avec les autres.

Qu'est-ce que GitHub Actions ?

GitHub Actions est un outil d'intégration et de déploiement continu (CI/CD) fourni par GitHub. Il permet d'automatiser les actions à effectuer lors d'un push de code, telles que :

- La compilation
- Les tests
- La construction d'images Docker
- Voir le déploiement automatique

Qu'est-ce qu'une pipeline CI/CD ?

Une **pipeline CI/CD** est une série d'étapes d'automatisation qui garantit que le code poussé est :

- **Correct** (via tests)
- **Propre** (via linter)
- **Compilable et exécutable**
- **Potentiellement déployable automatiquement**

Dans mon projet, la pipeline CI/CD est exécutée à chaque **push sur la branche main**. En cas de succès ou d'échec, une **notification est envoyée sur un groupe Google Chat** via un webhook intégré à la fin du job.

Conteneurisation avec Docker

Le projet est déployé sous forme de conteneurs Docker pour garantir portabilité, homogénéité et facilité de déploiement.

Backend : Dockerfile

```

1  FROM maven:3.9.6-eclipse-temurin-17 AS builder
2
3  WORKDIR /app
4
5  COPY pom.xml .
6  COPY src ./src
7
8  RUN mvn clean package -DskipTests
9
10 FROM eclipse-temurin:17-jdk-alpine
11
12 WORKDIR /app
13
14 COPY --from=builder /app/target/*.jar app.jar
15
16 EXPOSE 8080
17
18 ENTRYPOINT ["java", "-jar", "app.jar"]

```

Figure 35 : Dockerfile backend

Frontend : Dockerfile

```
1 FROM node:18 AS build
2 WORKDIR /app
3
4 COPY package*.json ./
5 RUN npm install --omit=dev
6
7 COPY . .
8 RUN npm run build
9
10 FROM nginx:alpine
11 RUN rm -rf /etc/nginx/conf.d/default.conf
12 COPY nginx.conf /etc/nginx/conf.d
13 COPY --from=build /app/build /usr/share/nginx/html
14 EXPOSE 80
15 CMD ["nginx", "-g", "daemon off;"]
```

Figure 36 : Dockerfile frontend

Extrait docker-compose.yml

```
1 version: '3.8'
2
3 > Run All Services
4 services:
5   > Run Service
6   db:
7     image: mysql:8.0
8     container_name: mysql-db
9     restart: always
10    environment:
11      MYSQL_ROOT_PASSWORD: Postmalone0751@
12      MYSQL_DATABASE: tododb
13    ports:
14      - "3307:3306" # Externe:Interne
15    volumes:
16      - db-data:/var/lib/mysql
17    networks:
18      - app-network
19
20    You, last week • app dockeriser avec
21
22   > Run Service
23   backend:
24     build:
25       context: .
26       dockerfile: Dockerfile
27     container_name: spring-backend
28     restart: always
29     environment:
```

Figure 37 : Dockercompose.yml

GitHub Actions : pipeline CI/CD

ci.yml extrait

```
1  name: ci
2
3  on:
4    push:
5      branches:
6        - main
7
8  jobs:
9    build-and-test:
10     runs-on: ubuntu-latest
11
12     services:
13       mysql:
14         image: mysql:8.0
15         env:
16           MYSQL_ROOT_PASSWORD: ${ secrets.MYSQL_ROOT_PASSWORD }
17           MYSQL_DATABASE: tododb
18         ports:
19           - 3306:3306
20         options: >-
21           --health-cmd="mysqladmin ping --silent"
22           --health-interval=10s
23           --health-timeout=5s
24           --health-retries=5
25
26     env:
27       DB_URL: ${ secrets.DB_URL }
28       DB_USERNAME: ${ secrets.DB_USERNAME }
29       DB_PASSWORD: ${ secrets.DB_PASSWORD }
30       EMAIL_USERNAME: ${ secrets.EMAIL_USERNAME }
31       EMAIL_PASSWORD: ${ secrets.EMAIL_PASSWORD }
```

Figure 38 : Fichier d'intégration

Qualité de code & linting

- **ESLint** pour React + TypeScript (npm run lint)
- **Maven** avec plugins de formatage Java
- **Validation CI** à chaque push

Suivi des logs et notifications

- GitHub Actions gère tous les logs : compilation, erreurs, tests, etc.

- Notification finale sur Google Chat : succès ou erreur (webhook configuré)

Déploiement manuel (Docker Compose)

Une fois la pipeline validée, le déploiement est réalisé manuellement avec :

```
s\user\Downloads\backend>
docker-compose up --build
[+] Running 12/1
```

Le frontend est accessible sur <http://localhost:3000> et le backend sur <http://localhost:8080>.

The image displays three screenshots of GitHub Actions workflow logs. The first screenshot shows the 'build-and-test' job, which succeeded last week in 1m 39s. The second screenshot shows the 'docker' job, which succeeded last week in 1m 23s. The third screenshot shows the 'notify' job, which succeeded last week in 3s. Below these screenshots is a screenshot of a Google Chat message from 'Ci/Cd notifier' dated '23 juil., 05:34'. The message states 'CI/CD terminé avec succès pour la branche main' and includes a 'Répondre' button. At the bottom, there is a text input field with the placeholder 'Envoyer un message à Ci/Cd/07/todolist' and a 'Send' button.

Figure 37 : Log pipelines sur GithubActions avec notification googletchat

Veille technologique

Objectifs de la veille

Tout au long de la réalisation du projet, une veille technologique régulière a été menée afin de mieux comprendre certaines notions techniques, d'améliorer les pratiques de développement et de résoudre des problèmes spécifiques rencontrés.

Cette démarche proactive m'a permis de rester informé des bonnes pratiques du secteur et d'intégrer de nouveaux outils professionnels à mon stack technologique.

Veille sur la sécurité : JWT

Le fonctionnement de **JWT (JSON Web Token)** a fait l'objet de nombreuses lectures, notamment sur :

- La structure du token (header.payload.signature)
- Les méthodes de validation et d'expiration
- Les risques liés au stockage local
- Les mécanismes de hachage et signature (HMAC SHA256)

Sources consultées :

- jwt.io (documentation officielle)
- StackOverflow (questions récurrentes sur les filtres Spring Security)
- Medium : articles de retours d'expérience en Java Spring Security

Veille sur les tests automatisés

Pour comprendre l'écosystème des tests en frontend et backend, une veille a été menée sur :

Cypress (tests E2E)

- Documentation officielle <https://docs.cypress.io>
- Chaînes YouTube et articles Medium
- Mise en place d'un test d'usage réel (création de liste via UI)

Jest (tests unitaires frontend)

- Site officiel jestjs.io
- Articles sur la gestion des snapshots et du mock
- StackOverflow pour la gestion des erreurs de version

Ces outils ont été intégrés à mon projet grâce à cette veille, et m'ont permis de mettre en place une stratégie de test fiable.

Découverte d'outils professionnels

Parmi les outils que j'ai découverts ou approfondis grâce à ce projet :

Outil	Utilité principale
Cypress	Tests end-to-end (interface utilisateur)
Jest	Tests unitaires (composants React)
MySQL Workbench	Conception visuelle MCD/MLD/MPD, requêtage SQL
Draw.io	Modélisation graphique (MCD notamment)

Sources et plateformes utilisées

- StackOverflow (résolutions communautaires)
- Medium (expériences de développeurs)
- Documentation officielle des frameworks :
 - [Spring.io](https://spring.io)
 - [React.dev](https://react.dev)
 - [Nodejs.org](https://nodejs.org)
 - [Docker.com](https://docker.com)
 - jwt.io
- GitHub (exploration de projets similaires open-source)
- YouTube (tutoriels ciblés sur la configuration Cypress et Jest)

Problèmes rencontrés & solutions

Tout au long du projet, plusieurs difficultés techniques ont été rencontrées, issues de différents environnements et couches de l'application : authentification, tests, conteneurisation, configuration de serveur, etc. Voici les principales problématiques rencontrées et les solutions mises en place.

Problème n°1 : Authentification JWT (backend)

Contexte : Lors de la mise en place du système de connexion via JWT, les requêtes étaient bien générées mais le token n'était pas reconnu sur les routes protégées, générant des erreurs 403.

Analyse :

- Le filtre `JwtAuthFilter` n'était pas correctement injecté dans la chaîne de filtrage Spring Security
- Problème de nom de header `Authorization` mal orthographié dans certaines requêtes

Solution mise en place :

- Réécriture du filtre `JwtAuthFilter` avec `OncePerRequestFilter`
- Injection correcte dans `SecurityFilterChain`
- Ajout de logs pour débogage

Apprentissage : Compréhension plus fine de la chaîne Spring Security et des filtres personnalisés.

Problème n°2 : Installation de Jest (tests frontend)

Contexte : Lors de l'ajout de Jest pour tester les composants React, une erreur de conflit de version empêchait npm install de réussir.

Analyse :

- Conflit entre les versions de Jest et React Scripts
- Paquets déjà préinstallés entraient en collision avec des plugins internes

Solution mise en place :

- Suppression du node_modules et du package-lock.json
- Réinstallation avec des versions compatibles de jest, babel-jest, ts-jest
- Configuration manuelle du fichier jest.config.ts

Apprentissage : Meilleure maîtrise de la gestion des dépendances Node.js et du fonctionnement interne de Jest.

Problème n°3 : Configuration de Nginx dans Docker

Contexte : Le frontend devait être servi avec Nginx dans le conteneur Docker. Les routes React ne fonctionnaient pas en direct (erreurs 404 au rafraîchissement).

Analyse :

- React utilise un système de routing interne (SPA)
- Nginx nécessite une redirection fallback vers index.html pour toutes les routes

Solution mise en place :

- Ajout d'une configuration personnalisée dans nginx.conf :

```
location / {  
  
    try_files $uri /index.html;  
  
}
```

- Réintégration du fichier dans l'image Docker et rebuild

Apprentissage : Compréhension du fonctionnement des SPAs avec serveurs statiques et des redirections Nginx.

Problème n°4 : Installation de Cypress (tests E2E)

Contexte : L'installation de Cypress posait problème suite à un conflit de version avec Node.js.

Analyse :

- Cypress n'était pas compatible avec la version de Node installée en local (trop récente)
- Erreurs lors de `npx cypress open`

Solution mise en place :

- Mise à jour de Cypress vers une version stable
- Alignement des versions dans `package.json`
- Installation d'une version de Node compatible via NVM

Apprentissage : Importance de vérifier la compatibilité des versions entre outils JS, gestion d'environnements avec NVM

Chaque problème rencontré a été l'occasion d'approfondir des compétences techniques, de documenter les outils utilisés, et d'adopter une méthodologie de résolution efficace (lecture de logs, tests en local, recherches GitHub et StackOverflow).

Synthèse / Conclusion

Bilan du projet

Ce projet a été l'occasion de concevoir une application web complète en respectant les principes d'une architecture full-stack moderne. De la conception des maquettes à la mise en place d'une pipeline CI/CD, chaque étape a permis d'approfondir des compétences techniques et organisationnelles essentielles au métier de développeur.

L'intégration d'outils comme Docker, GitHub Actions, Cypress, Jest, Spring Boot et React a permis de construire une application robuste, modulaire, et industrialisable.

Le projet s'inscrit dans une démarche réaliste : interface utilisateur moderne, gestion d'utilisateurs sécurisée (JWT), base de données relationnelle avec MCD/MLD/MPD documentés, et architecture RESTful.

Satisfaction personnelle

Sur le plan personnel, ce projet m'a permis :

- De renforcer ma rigueur dans la conception technique et la modélisation
- De découvrir des outils professionnels de déploiement et d'intégration continue
- De dépasser des difficultés techniques concrètes (authentification, testing, configuration Docker)
- D'avoir une vision plus complète du cycle de vie d'une application logicielle

Cette réalisation représente un véritable aboutissement technique et personnel. Elle témoigne de ma capacité à mener un projet web de bout en bout, à documenter proprement mon travail, et à me préparer au monde professionnel.