ORION

Justification des choix techniques

Projet MDD

Auteur : Didier Barriere Doleac
Version 0.0.1

VERSION MVP (Minimum Viable Product)

Aperçu / Synthèse

Architecture logicielle (client-serveur) :

*L'application **Angular (client)** communique avec l'application **Spring Boot (serveur)** pour échanger des données et effectuer des opérations.*

***Avantages :** Centralisation des données et des services, facilité de gestion et de sécurité, scalabilité.*

***Inconvénients :** Dépendance au serveur, surcharge potentielle du serveur, complexité de mise en œuvre.*

Paradigmes de programmation :

Programmation Réactive pour le front-end :
Le front-end (client) utilise Angular avec divers modules Angular Material pour une interface utilisateur riche et réactive grâce à la librairie RxJs.
Les services et intercepteurs Angular gèrent les sessions et la sécurité via JWT.

Programmation Orienté Objet pour le back-end :
Le côté API (serveur) utilise Spring Boot pour gérer les requêtes HTTP, la sécurité, et les opérations de persistance avec JPA et Hibernate pour l'ORM.

Paradigme ORM (Object-Relational Mapping) :
ORM permet de manipuler les données de la base comme s'il s'agissait d'objets dans le code, ce qui facilite le développement en évitant d'écrire du code SQL répétitif et complexe.

La base de données MySQL est utilisée pour le stockage des données, et des bibliothèques comme Jackson et MapStruct facilitent la manipulation des données et la conversion des objets.

Cette architecture permet une séparation claire des responsabilités entre le front-end et le back-end, facilitant ainsi la maintenance et l'évolution du projet.

Utilisation du framework Spring Security et de JSON Web Token (JWT)

Utilisation du design pattern Strategy pour encapsuler la logique d'authentification et d'autorisation via JWT dans des composants distincts : Ces composants

sont configurés et intégrés dans le framework Spring Security.

Gestion des tokens et de la sécurité

Dans le fichier `application.properties`, les paramètres de sécurité suivants peuvent être configurés :
- `oc.app.jwtSecret` : la clé secrète utilisée pour signer les tokens JWT.
- `oc.app.jwtExpirationMs` : la durée de validité des tokens JWT en millisecondes.

Pour cette version MVP, la durée de validité des tokens est définie par défaut à 24 heures (en millisecondes), mais cette durée ne sera pas utilisée pour la persistance des tokens entre les sessions.

Design patterns :

*Le **pattern Inversion of Control (IoC)**, implémenté via Spring Boot, centralise la création et la gestion des objets, tandis que le **pattern Dependency Injection** permet d'injecter des dépendances de manière flexible. Le **pattern Repository,** standardise l'accès aux données, et le **pattern Decorator** enrichit les fonctionnalités de sécurité, comme la protection des routes/API. Les **patterns Unit of Work et Identity Map** optimisent la gestion des transactions et des sessions. Le **pattern Model-View-Controller (MVC)** clarifie la séparation des responsabilités dans l'application. Le **pattern Builder**, réduit le code superflu, tandis que le **pattern Adapter** facilite le mappage objet-objet. Le **pattern Strategy,** via JWT, gère l'authentification, et le **pattern Factory** assure la connexion à MySQL.*

Choix techniques

BACK-END (API)

Frameworks :

Choix 1 (Spring Boot Starter Data JPA)

choix technique	lien vers le site / la documentation / une ressource	but du choix
Spring Boot Starter Data JPA	Getting Started | Accessing Data with JPA (spring.io)	*Implémenter des couches de persistance en utilisant JPA (Java Persistence API).*

Justification du choix technique :

Capacité à simplifier la configuration de la persistance des données, en intégrant automatiquement les dépendances nécessaires et en gérant les versions. Génération automatique de requêtes et réduit le besoin de code SQL personnalisé. La gestion automatique des transactions minimise les erreurs et garantit la cohérence des données. L'abstraction du repository simplifie l'accès aux données. Il supporte également la pagination, le tri, et améliore les performances grâce au cache.

Pros	***Cons***
• *Intégration transparente avec Spring Boot.* • *Réduction du code boilerplate.* • *Facilité d'utilisation des repositories.*	• *Surcouche supplémentaire.* • *Limitations des méthodes de requête générées.*

Choix 2 (Spring Boot Starter Web)

choix technique	lien vers le site / la documentation / une ressource	but du choix
Spring Boot Starter Web	Maven Repository: org.springframework.boot » spring-boot-starter-web (mvnrepository.com)	*Démarrer rapidement le développement web avec Spring.*

Justification du choix technique :
Capacité à simplifier le développement d'applications web et RESTful en Java. En encapsulant automatiquement les configurations et les dépendances nécessaires, il permet de démarrer rapidement sans se soucier des détails de configuration sous-jacents. Ce starter inclut Tomcat comme serveur web embarqué par défaut, facilitant le déploiement et la mise en service des applications. Support intégral pour la création de services web RESTful grâce à Spring MVC. Sérialisation et la désérialisation JSON automatiques avec Jackson, la validation des données, et la gestion des erreurs, améliorant ainsi la robustesse et la qualité du code.

Pros	***Cons***
• *Configuration simplifiée : Démarrage rapide avec des configurations par défaut.* • *Intégration facile : Fonctionne bien avec d'autres composants Spring.* • *Communauté active : Support et documentation abondants.*	• *Abstraction élevée : Moins de contrôle sur les détails.* • *Dépendances lourdes : Peut augmenter la taille de l'application.*

Choix 3 (Spring Boot Starter Security)

choix technique	lien vers le site / la documentation / une ressource	but du choix
Spring Boot Starter Security	Getting Spring Security :: Spring Security	*En utilisant Spring Security, sécuriser l'application contre les menaces communes.*

Justification du choix technique :
Ce starter simplifie l'intégration de mécanismes d'authentification et d'autorisation, protégeant ainsi les applications contre les menaces courantes. Il gère de manière transparente la sécurité des sessions, la gestion des mots de passe, et la sécurisation des requêtes HTTP avec peu de configuration requise. Conforme aux normes de sécurité.

Pros	***Cons***
• *Configuration simplifiée : Intégration rapide avec des configurations par défaut.* • *Sécurité robuste : Prend en charge les meilleures pratiques de sécurité.* • *Extensibilité : Facile à personnaliser et à étendre.*	• *Courbe d'apprentissage : Peut être complexe* • *Dépendances lourdes : Peut augmenter la taille de l'application.*

Choix 4 (Hibernate Core)

choix technique	lien vers le site / la documentation / une ressource	but du choix
Hibernate Core	Getting started - Hibernate ORM	*faciliter la cartographie des objets Java aux tables de base de données et vice versa.*

Justification du choix technique :
En tant que framework ORM (Object-Relational Mapping), il facilite la cartographie des objets Java aux tables de bases de données, permettant ainsi une manipulation des données de manière plus intuitive et orientée objet. De plus, il offre des fonctionnalités avancées telles que le caching, la gestion des transactions et l'optimisation des requêtes, contribuant à améliorer les performances de l'application.

Pros	Cons
• *ORM puissant : Simplifie la gestion des bases de données.* • *Support HQL : Langage de requête orienté objet.* • *Cache intégré : Améliore les performances.*	• *Courbe d'apprentissage : peu être complexe* • *Configuration lourde : Peut nécessiter beaucoup de configurations.* • *Performance : Peut être moins performant pour des requêtes très complexes.*

Librairies:

Choix 5 (MySQL Connector/J)

choix technique	lien vers le site / la documentation / une ressource	but du choix
MySQL Connector/J	Maven Repository: mysql » mysql-connector-java (mvnrepository.com)	*connecter l'API a la base de données MySQL, interaction fluide entre l'application et la base de données.*

Justification du choix technique :
En tant que driver JDBC officiel, il assure une compatibilité optimale et des performances élevées pour les applications Java communiquant avec MySQL. Il offre une solution fiable, performante et facile à utiliser pour intégrer MySQL dans des applications Java.

Pros	Cons
• *Compatibilité : Fonctionne bien avec MySQL.* • *Performance : Optimisé pour des performances élevées.* • *Documentation : Bien documenté et supporté.*	• *Dépendance spécifique : Limité à MySQL.* • *Complexité : Peut être complexe à configurer* • *Mises à jour : Nécessite des mises à jour régulières pour rester sécurisé.*

Choix 6 (Java JWT)

choix technique	lien vers le site / la documentation / une ressource	but du choix
Java JWT	GitHub - auth0/java-jwt: Java implementation of JSON Web Token (JWT)	***Securité:*** *création et vérification des tokens JWT.*

Justification du choix technique :
Cette bibliothèque se distingue par sa simplicité d'utilisation et sa légèreté, facilitant l'intégration de la sécurité basée sur les tokens dans l'API. Elle permet de créer et de vérifier des tokens JWT de manière efficace, offrant ainsi une méthode sécurisée pour l'authentification et l'échange d'informations entre le client et le serveur. De plus, elle est compatible avec les spécifications JWT les plus récentes, assurant ainsi que les applications restent à jour avec les pratiques de sécurité actuelles.

Pros	***Cons***
• *Sécurité : Permet une authentification sécurisée.* • *Interopérabilité : Compatible avec de nombreux langages et frameworks.* • *Simplicité : Facile à intégrer et utiliser.*	• *Complexité : Peut être difficile à configurer correctement.* • *Taille des tokens : Les tokens peuvent devenir volumineux.* • *Expiration : Gestion des expirations peut être complexe.*

Choix 7 (Project Lombok)

choix technique	lien vers le site / la documentation / une ressource	but du choix
Project Lombok	Maven Repository: org.projectlombok » lombok (mvnrepository.com)	*réduire le boilerplate code en générant automatiquement des méthodes courantes*

Justification du choix technique :
En automatisant la génération de méthodes courantes comme les getters, setters, et les méthodes equals(), hashCode(), et toString() à travers des annotations simples, Lombok améliore significativement la lisibilité et la concision du code. De plus, Lombok facilite la maintenance du code en réduisant les risques d'erreurs manuelles dans ces méthodes générées automatiquement.

Pros	Cons
• *Réduction du code boilerplate : Génère automatiquement les getters, setters, et autres méthodes courantes.* • *Lisibilité : Rend le code plus propre et lisible.* • *Productivité : Accélère le développement.*	• *Dépendance : Nécessite une dépendance supplémentaire.* • *Compatibilité IDE : Peut poser des problèmes avec certains IDE.* • *Debugging : Peut compliquer le débogage et la compréhension du code généré.*

Choix 8 (Jakarta Validation API)

choix technique	lien vers le site / la documentation / une ressource	but du choix
Jakarta Validation API	GitHub - jakartaee/validation: Jakarta Validation	*Validation des données : assure que les données respectent certaines contraintes avant leur traitement.*

Justification du choix technique :
Permet de définir des contraintes de validation directement sur les modèles de données à l'aide d'annotations simples, telles que @NotNull, @Size, @Min, et @Max. En s'assurant que les données respectent les contraintes définies avant d'être traitées ou persistées, Jakarta Validation API contribue à prévenir les erreurs de données et à renforcer l'intégrité des systèmes.

Pros	Cons
• *Standardisé : Basé sur des standards, compatible avec de nombreux frameworks.* • *Annotations : Simplifie la validation avec des annotations.* • *Extensibilité : Facile à étendre avec des validations personnalisées.*	• *Complexité : Peut être complexe pour des validations avancées.* • *Performance : Peut impacter les performances pour des validations lourdes.* • *Dépendance : Nécessite une dépendance supplémentaire.*

Choix 9 (Jackson Databind et Jackson Core)

choix technique	lien vers le site / la documentation / une ressource	but du choix
Jackson Databind et Jackson Core	GitHub - FasterXML/jackson-databind: General data-binding package for Jackson (2.x): works on streaming API (core) implementation(s)	*Sérialisation/désérialisation et de traitement de base JSON.*

Justification du choix technique :
Offrent une solution complète pour le traitement JSON, permettant de convertir facilement des objets Java en représentations JSON et vice versa, tout en permettant un contrôle fin sur le processus grâce à des annotations et des modules personnalisables.

Pros	***Cons***
• *Sérialisation/Désérialisation : Facilite la conversion entre objets Java et JSON.* • *Flexibilité : Supporte de nombreuses configurations et annotations.* • *Performance : Rapide et efficace pour la manipulation de JSON.*	• *Complexité : Peut être complexe pour des cas d'utilisation avancés.* • *Dépendances : Ajoute des dépendances supplémentaires.* • *Taille : Peut augmenter la taille de l'application.*

Choix 10 (MapStruct)

choix technique	lien vers le site / la documentation / une ressource	but du choix
MapStruct	MapStruct – Java bean mappings, the easy way!	*conversion entre les objets DTO et les entités de domaine*

Justification du choix technique :
simplifier et accélérer le processus de mappage entre les objets, en particulier entre les objets de transfert de données (DTO) et les entités de la base de données. En générant automatiquement le code de mappage à la compilation, MapStruct élimine le besoin d'implémentations manuelles, réduisant ainsi le risque d'erreurs tout en offrant une flexibilité pour gérer des cas de mappage

complexes grâce à des stratégies de mappage personnalisables.

Pros	***Cons***
• *Performance : Génère du code de mapping à la compilation, ce qui est rapide et efficace.* • *Simplicité : Facile à utiliser avec des annotations simples.* • *Maintenance : Réduit le code boilerplate, rendant le code plus propre et maintenable.*	• *Dépendance : Nécessite une dépendance supplémentaire.* • *Complexité : Peut être complexe pour des mappings avancés.* • *Debugging : Le code généré peut compliquer le débogage.*

Design Patterns:

Choix 11 (Patterns Inversion of Control (IoC) et Dependency Injection)

choix technique	lien vers le site / la documentation / une ressource	but du choix
Pattern Inversion of Control (IoC) et pattern Dependency Injection (via Spring Boot)	Inversion de contrôle — Wikipédia (wikipedia.org)	*Pour gérer les objets de l'application, avec Dependency Injection pour injecter les dépendances.*

Justification du choix technique :
Ce pattern () déplace la responsabilité de la construction des dépendances d'un objet depuis l'objet lui-même vers un conteneur ou un framework externe. Cette approche favorise également le développement de composants réutilisables et interchangeables, améliorant la modularité de l'application.

Pros	***Cons***
• *Découplage : Facilite la séparation des préoccupations.* • *Testabilité : Simplifie les tests unitaires en permettant l'injection de mocks.* • *Flexibilité : Permet de changer facilement les implémentations.*	• *Complexité : Peut ajouter de la complexité au projet.* • *Courbe d'apprentissage : Peut être difficile à maîtriser* • *Dépendance : Nécessite une dépendance à un conteneur IoC comme Spring.*

Choix 12 (Pattern Repository)

choix technique	lien vers le site / la documentation / une ressource	but du choix
Pattern Repository (via Spring Data JPA)	Implémentez vos entités et les interfaces repository - Utilisez Spring Data pour interagir avec vos bases de données - OpenClassrooms	*Standardiser l'accès aux données*

Justification du choix technique :
Intermédiaire entre le domaine de l'application et la couche de données, il permet ainsi une séparation claire des préoccupations. Cette abstraction facilite la gestion des opérations de données, comme les requêtes et les mises à jour, en offrant une interface simple et cohérente pour accéder aux données, indépendamment de la source de données sous-jacente (bases de données relationnelles, NoSQL, fichiers, etc.). Il simplifie également le remplacement ou la modification de la source de données sans impacter la logique métier.

Pros	Cons
• *Abstraction : Simplifie l'accès aux données en masquant les détails de l'implémentation.* • *Productivité : Génère automatiquement les méthodes CRUD.* • *Extensibilité : Facile à étendre avec des requêtes personnalisées.*	• *Performance : Peut être moins performant pour des requêtes complexes.* • *Dépendance : Nécessite une dépendance à Spring Data JPA.* • *Courbe d'apprentissage : Peut être difficile à maîtriser*

Choix 13 (pattern Decorator)

choix technique	lien vers le site / la documentation / une ressource	but du choix
Pattern Decorator (via Spring Security)	Décorateur (patron de conception) — Wikipédia (wikipedia.org)	*Enrichit les fonctionnalités de sécurité, comme la protection des routes/API.*

Justification du choix technique :
Permet d'ajouter dynamiquement des fonctionnalités supplémentaires à des objets sans modifier leur structure interne. En enveloppant un objet dans une série de "décorateurs", chacun ajoutant un comportement ou des responsabilités supplémentaires, le Decorator permet une composition flexible et réutilisable des comportements. Permet aussi d'ajouter des couches de sécurité, telles que l'authentification ou le chiffrement, sans toucher au code existant.

Pros	Cons
• *Extensibilité : Permet d'ajouter des fonctionnalités dynamiquement.* • *Réutilisabilité : Favorise la réutilisation de composants existants.* • *Flexibilité : Facile à combiner plusieurs décorateurs.*	• *Complexité : Peut rendre le code plus difficile à comprendre.* • *Surcharge : Peut introduire une surcharge de performance.* • *Dépendance : Nécessite une bonne gestion des dépendances entre décorateurs.*

Choix 14 (patterns Unit of Work et Identity Map)

choix technique	lien vers le site / la documentation / une ressource	but du choix
patterns Unit of Work et Identity Map (via Hibernate)	java - How to use UnitOfWork pattern together with IdentityMap pattern? - Stack Overflow	*Optimisent la gestion des transactions et des sessions*

Justification du choix technique :
Le pattern Unit of Work centralise la gestion des opérations de base de données en regroupant plusieurs modifications en une seule transaction.
Le pattern Identity Map maintient un cache des objets déjà chargés dans la mémoire, évitant ainsi les requêtes redondantes à la base de données et améliorant les performances. En garantissant qu'un même objet n'est chargé qu'une seule fois par session, il prévient les problèmes de duplication et assure l'intégrité des données.

Pros	***Cons***
Unit of Work • *Atomicité : Assure que les opérations sont atomiques.* • *Gestion des transactions : Simplifie la gestion des transactions.* • *Performance : Réduit les appels à la base de données.* *Identity Map* • *Cache : Réduit les requêtes redondantes.* • *Consistance : Assure la consistance des objets en mémoire.*	*Unit of Work* • *Complexité : Peut ajouter de la complexité au code.* • *Mémoire : Peut consommer plus de mémoire.* *Identity Map* • *Mémoire : Peut augmenter l'utilisation de la mémoire.* • *Synchronisation : Peut compliquer la synchronisation des états.*

Choix 15 (pattern Model-View-Controller (MVC))

choix technique	lien vers le site / la documentation / une ressource	but du choix
pattern Model-View-Controller (MVC) (via Spring Boot)	Model–view–controller - Wikipedia	*Clarifie la séparation des responsabilités dans l'application.*

Justification du choix technique :
Permet de structurer les applications de manière modulaire et maintenable. En séparant les responsabilités (le Modèle, la Vue et le Contrôleur) le pattern MVC facilite la gestion et l'évolution du code. Le Modèle représente les données et la logique métier, la Vue gère l'affichage et l'interface utilisateur, tandis que le Contrôleur traite les entrées de l'utilisateur et coordonne les interactions entre le Modèle et la Vue. Chaque composant peut être développé et testé indépendamment.

Pros	***Cons***
• *Séparation des préoccupations : Clarifie la structure du code en séparant la logique métier, la présentation et le contrôle.* • *Modularité : Facilite la maintenance et l'évolution du code.* • *Réutilisabilité : Encourage la réutilisation des composants.*	• *Complexité : Peut ajouter de la complexité pour les petits projets.* • *Configuration : Nécessite une configuration initiale.* • *Performance : Peut introduire une surcharge de performance.*

Choix 16 (pattern Builder)

choix technique	lien vers le site / la documentation / une ressource	but du choix
pattern Builder (via Lombok)	@Builder (projectlombok.org)	*Réduit le code superflu.*

Justification du choix technique :

Le pattern Builder permet de generer automatiquement le code nécessaire, réduisant ainsi le boilerplate et accélérant le développement (ex: getters and setters, etc...).

Pros	***Cons***
• *Lisibilité : Rend le code plus lisible et compréhensible.* • *Immutabilité : Facilite la création d'objets immuables.* • *Flexibilité : Permet de construire des objets complexes de manière flexible.*	• *Dépendance : Nécessite une dépendance à Lombok.* • *Complexité : Peut ajouter de la complexité pour des objets simples.* • *Debugging : Le code généré peut compliquer le débogage.*

Choix 17 (pattern Adapter)

choix technique	lien vers le site / la documentation / une ressource	but du choix
Pattern Adapter (via MapStruct)	MapStruct Spring Extensions 1.1.1 Reference Guide	*Facilite le mappage objet-objet*

Justification du choix technique :
Le pattern Adapter favorise la réutilisabilité et la maintenabilité du code en séparant clairement les préoccupations de mappage des autres logiques métier. Il permet de mapper des objets de transfert de données (DTO) à des entités de domaine et vice versa. MapStruct génère automatiquement le code de mappage à la compilation, éliminant ainsi le besoin d'écrire manuellement des convertisseurs souvent fastidieux et sujets aux erreurs.

Pros	***Cons***
• *Découplage : Permet de connecter des interfaces incompatibles sans modifier leur code.* • *Réutilisabilité : Favorise la réutilisation de code existant.* • *Simplicité : Simplifie les conversions entre objets.*	• *Complexité : Peut ajouter de la complexité au projet.* • *Dépendance : Nécessite une dépendance à MapStruct.* • *Maintenance : Peut compliquer la maintenance si les adaptateurs sont nombreux.*

Choix 18 (pattern Strategy)

choix technique	lien vers le site / la documentation / une ressource	but du choix
Pattern Strategy (via JWT)	Strategy pattern : le patron Strategy expliqué avec un exemple - IONOS	*Gère l'authentification par token.*

Justification du choix technique :
Le pattern Strategy permet de définir une famille d'algorithmes, de les encapsuler et de les rendre interchangeables. En utilisant JWT pour l'authentification, on peut facilement changer ou mettre à jour la stratégie d'authentification sans modifier le code client. Les tokens JWT sont auto-contenus, incluant toutes les informations nécessaires pour vérifier l'identité de l'utilisateur et ses permissions, ce qui élimine le besoin de stocker des sessions côté serveur.

Pros	***Cons***
• *Modularité : Permet de séparer et d'encapsuler les différentes stratégies d'authentification.* • *Flexibilité : Facile à changer ou à étendre les stratégies sans modifier le code existant.* • *Testabilité : Simplifie les tests unitaires des composants individuels.*	• *Complexité : Peut ajouter de la complexité au projet.* • *Configuration : Nécessite une configuration initiale.* • *Dépendance : Peut introduire des dépendances supplémentaires.*

Choix 19 (pattern Factory)

choix technique	lien vers le site / la documentation / une ressource	but du choix
Pattern Factory) (via Connector/J)	Déléguez la création des objets avec le Factory Pattern - Utilisez des design patterns en JavaScript - OpenClassrooms	*Assure la connexion à MySQL.*

Justification du choix technique :
En utilisant Connector/J, le driver JDBC officiel pour MySQL, le pattern Factory simplifie et standardise la gestion des connexions à la base de données.

Cette approche centralisée facilite la configuration et la réutilisation des connexions, réduisant ainsi le risque d'erreurs.

Pros	***Cons***
• *Encapsulation : Cache les détails de création des objets.* • *Flexibilité : Facilite l'ajout de nouvelles classes sans modifier le code existant.* • *Réutilisabilité : Encourage la réutilisation du code de création d'objets.*	• *Complexité : Peut ajouter de la complexité au projet.* • *Dépendance : Nécessite une dépendance à Connector/J.* • *Maintenance : Peut compliquer la maintenance si les factories sont nombreuses.*

FRONT-END (UI)

Choix 20 (Framework Angular)

choix technique	lien vers le site / la documentation / une ressource	but du choix
Framework Angular	Home • Angular	*Pour développer l'application web côté client.*

Justification du choix technique :
Angular est un framework robuste et complet développé par Google, offrant une structure solide pour construire une applications web dynamique et réactive.
Il propose une architecture basée sur des composants, ce qui permet de réutiliser et de tester facilement les différentes parties de l'application. Son système de modules permet de charger les fonctionnalités à la demande, améliorant ainsi les performances de l'application

Pros	Cons
• *Modularité : Structure le code en modules, facilitant la maintenance.* • *Two-way Data Binding : Synchronise automatiquement le modèle et la vue.* • *Écosystème : Large écosystème avec de nombreuses bibliothèques et outils.* • *Communauté active : Support et documentation abondants.*	• *Complexité : Peut être complexe à apprendre pour les débutants.* • *Performance : Peut être moins performant pour des applications très simples.* • *Taille : Les applications peuvent devenir volumineuses.*

Choix 21 (Material)

choix technique	lien vers le site / la documentation / une ressource	but du choix
Material	Angular Material UI component library	*Fournit des composants d'interface utilisateur (UI)*

Justification du choix technique :
Angular Material est une bibliothèque de composants UI qui suit les principes de Material Design, garantissant une interface utilisateur cohérente, moderne et esthétique.
Angular Material fournit des composants préconstruits et personnalisables, tels que des boutons, des formulaires, des dialogues et des barres de navigation, optimisés pour les performances et la réactivité, assurant une expérience utilisateur fluide sur différents appareils et tailles d'écran.

Pros	Cons
• *Composants préconstruits : Offre une large gamme de composants UI prêts à l'emploi.* • *Consistance : Assure une apparence et une convivialité cohérentes.* • *Documentation : Bien documenté avec de nombreux exemples.*	• *Personnalisation : Peut être difficile à personnaliser pour des besoins spécifiques.* • *Taille : Peut augmenter la taille de l'application.* • *Dépendance : Nécessite une dépendance supplémentaire à Angular Material.*
