

INFO 2A - SAE 3.01 Création et déploiement de services applicatifs/développement

Compte rendu de projet - Équipe projet n°3



ChateauTresor

Simon ZERU, Jess ROUSSELARD, Paul MONCENIX-LARUE, Rahim BOUGHENDJOUR, Yvan D'ETTORRE et Matis MALANDRINO

Sommaire

1. Synthèse du projet.....	3
Introduction.....	3
Contexte.....	3
Objectifs.....	3
Résumé du projet.....	3
2. Bilan des risques.....	4
3. Mesures organisationnelles.....	5
Gestion du temps.....	5
Communication au sein de l'équipe.....	5
Suivi des livrables.....	5
4. Mesures techniques.....	6
Tests.....	6
Revue de code.....	6
Documentation.....	6
Gestion des bugs et des retours.....	6
5. Synthèse technique.....	7
Modélisation.....	7
Choix Techniques.....	7
Adéquation et Cohérence.....	7
Retour Critique.....	8
6. Bilan général.....	9
Points forts.....	9
Points faibles.....	9
Améliorations possibles.....	9
Conclusion.....	9
7. Annexes.....	10

1. Synthèse du projet

Contexte

Le patrimoine culturel français recèle de trésors historiques souvent méconnus. Malgré leur richesse, ces lieux peinent parfois à attirer l'attention du public. Notre équipe de 6 étudiants a été missionnée de réaliser une application web utilisant les données issues de l'open data pour valoriser le patrimoine Français. C'est pourquoi notre projet, [ChateauTresor](#) a été lancé pour remplir cette mission.

Projet

Notre application *ChateauTresor* permet de créer et de participer à des chasses au trésor dans les domaines de nos châteaux partenaires. L'application s'adresse à trois types d'utilisateurs, les participants des chasses au trésor, les équipes organisatrices de chasses qui peuvent être des entreprises ou des particuliers (qui devront tous être vérifiés avant la création de leur compte) et les propriétaires de châteaux, qui loue leurs châteaux aux équipes organisatrices.

Pour chaque chasse, un système de points est mis en place. Une chasse contient des énigmes qui elles même contiennent des indices. Par défaut, chaque indice est caché. Pour chaque indice révélé, le participant perd des points. À la fin de la chasse, le participant peut débloquent à l'aide de ses points restants une ou plusieurs récompenses disponibles Le participant est ensuite invité à donner des retours.

Objectifs

L'objectif principal de ChateauTresor est de valoriser les châteaux de la Loire en renforçant le tourisme et en préservant ce patrimoine historique. L'application répond aux besoins spécifiques de trois types d'utilisateurs :

Pour les participants de chasses au trésor, nous offrons une expérience ludique et immersive en les transformant en détectives. En résolvant des énigmes et en recherchant des QR codes cachés, ils découvrent des anecdotes et des secrets historiques, rendant leur visite enrichissante et mémorable.

Pour les organisateurs, qu'ils soient entreprises ou associations, nous fournissons un outil intuitif leur permettant de créer et gérer des chasses au trésor. Cela leur permet de diversifier leurs activités, de générer des revenus et de proposer des expériences captivantes aux participants.

Pour les propriétaires de châteaux, nous offrons une solution pour valoriser leur domaine en accueillant des chasses au trésor. Cela leur permet de générer des revenus pour l'entretien et la préservation de leur patrimoine, tout en leur offrant un contrôle total sur leur emploi du temps et les événements organisés.

2. Bilan des risques

Durant le projet, nous avons rencontré plusieurs risques. Grâce à une bonne mitigation et à une bonne communication, nous avons su les gérer sans impact majeur sur notre avancement.

Parfois, certains membres du groupe ne pouvaient pas nous rejoindre au poste de travail pour des raisons justifiées. Le risque était de se dé-synchroniser et prendre du retard dans la communication. Comme mitigation, nous avons mis en place du télétravail en utilisant Discord pour communiquer en temps réel. Par exemple, lorsqu'un des membres est tombé malade, il a télétravaillé et est resté en appel vocal avec nous qui travaillions depuis les locaux de l'IUT 2.

Un autre défi technique était l'extraction des données sur les châteaux. Nous n'avons pas utilisé de logiciels ETL comme prévu initialement. À la place, nous avons téléchargé une base de données issue du site du gouvernement. Puis, nous avons transformé ces données pour correspondre aux données dont nous avons besoin. Les données n'étant pas satisfaisantes, nous avons décidé d'enrichir ces données en faisant des requêtes à l'API Wikipédia. Une fois ces données satisfaisantes, nous les avons insérées en base.

Concernant TypeScript et React, nous avons anticipé le risque de manque de maîtrise. C'est pourquoi avant de commencer la programmation, notre chef de projet a réalisé une documentation pour que l'on maîtrise l'essentiel de ces technologies. Nous nous sommes formés en suivant ses tutoriels et en pratiquant des exercices. Si une personne bloquait, elle demandait de l'aide à un membre plus expérimenté ou se documentait pour trouver une solution. Cette entraide a été essentielle pour avancer sereinement tout au long du projet.

Sur le plan sécuritaire, nous avons utilisé Supabase pour gérer nos données. Qui sécurise et chiffre les informations sensibles. Par exemple, les mots de passe des utilisateurs sont chiffrés en Bcrypt. Cela nous a évité d'avoir à mettre en place des solutions complexes pour protéger les données. Nous sommes au courant qu'une méthode de chiffrement comme argon2 aurait été plus intéressante mais notre technologie nous limite à l'utilisation de bcrypt. Pour plus d'informations sur la sécurité, se référer à l'annexe [check-list de sécurité](#).

Enfin, nous avons pensé au risque d'utilisateurs malveillants. Nous avons donc mis en place un système de vérification des comptes des organisateurs. Ils doivent fournir une pièce d'identité ou un numéro d'entreprise pour créer un compte et se faire valider par nos administrateurs. L'interface de Supabase nous permet aussi de supprimer les chasses ne respectant pas nos [conditions générales d'utilisation](#) ou de bannir des utilisateurs si nécessaire.

3. Mesures organisationnelles

Gestion du temps

Nous avons adopté une approche simple, mais efficace pour gérer notre temps. Au lieu de créer un planning complexe, nous nous sommes fixés des dates précises pour chaque étape du projet. Chaque semaine, nous nous répartissions les tâches à faire en fonction des envies et des compétences de chacun. Par exemple, si quelqu'un était plus à l'aise avec le développement front-end, il prenait en charge les interfaces utilisateur, tandis qu'un autre se concentrait sur le back-end ou la gestion des données. Cette méthode nous a permis de rester motivés et de travailler sur des sujets qui nous intéressaient. Grâce à cette répartition naturelle, nous n'avons rencontré aucun problème de retard ou de surcharge. Pour plus de détails, se référer à l'annexe du [diagramme de Gantt](#).

Communication au sein de l'équipe

La communication a été un pilier essentiel de notre organisation. Nous avons utilisé Discord comme outil principal pour échanger en temps réel. Plusieurs canaux ont été créés pour structurer nos discussions : un canal pour les annonces importantes, un autre pour les tâches à faire, et des canaux dédiés à des sujets spécifiques comme le front-end, le back-end ou les tests. Cela nous a permis de garder les discussions claires et organisées. En plus des échanges sur Discord, nous organisons des réunions régulières pour faire le point sur l'avancement et résoudre les éventuels blocages.

Suivi des livrables

Pour le suivi des livrables, nous avons adopté une approche inspirée des méthodes agiles. Le projet a été découpé en petites étapes, avec des objectifs clairs à chaque fois. Par exemple, une semaine pouvait être consacrée à la création de l'API et du modèle, une autre à la création des interfaces utilisateurs. Cette méthode nous a permis de réaliser des versions fonctionnelles régulièrement et de valider rapidement les fonctionnalités. Pour gérer les versions du code, nous avons utilisé GitLab. Contrairement à une approche classique avec plusieurs branches, nous avons choisi de travailler sur une seule branche principale. Chaque membre de l'équipe faisait des commits réguliers et prévenait les autres avant de pousser ses modifications. Cette méthode nous a permis de garder l'application toujours fonctionnelle, car chaque modification était immédiatement intégrée et testée. De plus, cela a simplifié la gestion des conflits et assuré une collaboration fluide entre tous les membres de l'équipe.

4. Mesures techniques

Développement guidé par les tests et Approche centrée utilisateur

Nous avons réalisé des [tests unitaires](#) pour vérifier le bon fonctionnement des modules isolés, comme la validation des réponses aux énigmes ou la gestion des réservations. Ces tests nous ont permis de détecter rapidement des erreurs dans les fonctions critiques et d'assurer une base solide au développement. L'API a aussi été testé manuellement avant chaque implémentation de nouvelle méthode avec le logiciel POSTMAN.

En complément nous avons notamment réalisé des tests utilisateurs quantitatifs et qualitatifs comme SUS et Think Aloud. Les tests Think aloud ont été plus convaincants grâce à la mise en contexte et le côté concret, ce qui nous a permis de relever des problèmes d'interfaces (manque de guidance, charge de travail) qui ont été corrigés par la suite. Le test SUS quant à lui n'a pas été fait par assez de personnes (12 personnes) pour être assez convaincant, nous avons quand même obtenu un score de 71 ce qui reste encourageant.

Revue de code et Programmation par les pairs

La revue de code a été une pratique essentielle pour maintenir la qualité du code. Avant chaque fusion dans la branche principale, le code était relu par au moins un autre membre de l'équipe. Nous avons également pratiqué le **pair programming**, comme Simon et Rahim sur le développement du back-end, ou Jess et Paul sur les tests unitaires. Cette approche a renforcé la collaboration, accéléré la résolution de problèmes et partager les bonnes pratiques au sein de l'équipe.

Documentation

La documentation a été soignée pour faciliter la compréhension et la maintenance du projet. Nous avons ajouté des commentaires dans le code pour expliquer les parties complexes ou les choix techniques. Par exemple, les fonctions de calcul de score initial d'une chasse au trésor ou de calcul du score moyen d'une chasse sont accompagnées de commentaires détaillés. En plus des commentaires, nous avons rédigé un README complet, qui décrit les étapes de déploiement, les dépendances nécessaires et les principales fonctionnalités de l'application. Ce fichier est une ressource précieuse pour tout nouveau développeur qui rejoindrait le projet.

Gestion des bugs et des retours

La gestion des bugs se faisait de manière collaborative et directe. Lorsque nous étions ensemble et qu'un bug était identifié, ou même si nous avions une remarque à faire, nous en discussions immédiatement pour trouver des solutions. Par exemple, nous avons corrigé des erreurs d'interface utilisateur, comme souci de concordance des codes ou des couleurs, et ajouté des popups de confirmation pour éviter des actions involontaires, comme la suppression d'un élément sans avertissement. Chaque problème était noté pour s'assurer qu'il ne soit pas oublié, et si un bug n'était pas résolu avant la fin de la journée, nous le reprenions le lendemain. Les corrections étaient testées avant intégration, garantissant ainsi la qualité et la fiabilité du projet.

5. Synthèse technique

Modélisation

La modélisation initiale de l'application s'est appuyée sur plusieurs outils et schémas pour organiser les données et les processus. Le Schéma Entité-Association (SEA) a permis d'identifier les entités principales (Utilisateur, Château, Chasse, Énigme, etc.) et leurs relations. Il a ensuite été traduit en Schéma Logique-Relationnel (SLR), guidant la conception technique de la base de données. Un diagramme BPMN a été élaboré pour modéliser des processus critiques, comme la vérification des comptes des organisateurs. Cette approche a assuré la qualité des services et facilité le suivi des tâches.

Sur le plan logiciel, une architecture modulaire a été adoptée. L'architecture suit la structure de base d'un projet NextJS avec quelques spécifications. Le front-end, développé en React, est relié à un back-end développé en TypeScript, lui-même relié à une base de données PostgreSQL, gérée via Supabase. L'application est hébergée sur Vercel, qui garantit une performance optimale puisque ce sont les créateurs de nextJS.

Choix Techniques

Les choix technologiques ont été motivés par la volonté de développer une application moderne et performante tout en répondant aux besoins spécifiques du projet. TypeScript a été retenu pour son typage statique, censé réduire les erreurs et améliorer la robustesse du code. Ce langage a été couplé avec React pour créer des interfaces utilisateur dynamiques et modulaires. Next.js a été privilégié en raison de ses capacités de rendu côté serveur (SSR) et de génération statique (SSG), optimisant ainsi les performances et le SEO.

Pour la gestion des données, PostgreSQL a été choisi pour sa robustesse, tandis que Supabase a simplifié son intégration grâce à ses outils. Pour la carte interactive, nous utilisons la bibliothèque react Leaflet qui utilise l'API cartographique OpenStreetMap. L'application est hébergée sur Vercel, optimisé pour Next.js, facilitant le déploiement.

Adéquation et Cohérence

Globalement, les choix techniques ont bien répondu aux besoins définis lors de la modélisation. Next.js s'est avéré performant et a permis de développer une application rapide et optimisée. La combinaison PostgreSQL et Supabase a facilité la gestion des données complexes. L'absence d'une modélisation détaillée des types et du modèle au début du projet a conduit à des modifications importantes au cours du développement. Cela a entraîné une perte de temps considérable et des complications dans la gestion des versions de code. Par ailleurs, l'utilisation de TypeScript n'a pas été aussi bénéfique que prévu. Compte tenu de la taille et de la complexité du projet, ce langage a engendré une charge de travail supplémentaire sans apporter une valeur ajoutée significative, et il aurait été plus pertinent d'utiliser JavaScript standard.

Retour Critique

Le projet présente plusieurs réussites majeures. L'architecture adoptée a permis de développer une application performante, offrant une expérience utilisateur intuitive et fluide. Les interfaces sont bien adaptées aux différents profils d'utilisateurs (participants, organisateurs, propriétaires de châteaux), et la modularité de React a simplifié le développement. L'utilisation de Supabase a également permis de gagner du temps grâce à ses fonctionnalités intégrées, comme l'authentification et le stockage cloud. Enfin, le choix de Next.js et Vercel a permis de combiner performance et simplicité de déploiement.

Cependant, plusieurs points faibles ont été identifiés. L'absence d'une modélisation initiale des types et du modèle a causé des ajustements coûteux en termes de temps et d'efforts. L'utilisation de TypeScript, bien qu'elle visait à renforcer la fiabilité du code, s'est révélée inadaptée à l'échelle de ce projet et a davantage ralenti le développement. Un langage comme JavaScript aurait suffi, tout en réduisant la complexité initiale. Par ailleurs, les limitations des plans gratuits de Vercel et Supabase pourraient poser un problème si le projet venait à s'étendre.

Pour éviter ces problèmes à l'avenir, il aurait été crucial de renforcer la phase de conception initiale, en incluant une modélisation détaillée des types et des diagrammes de classes. De plus, un choix plus simple en termes de langage, comme JavaScript, aurait permis de se concentrer sur les fonctionnalités clés sans complexifier inutilement le développement.

Nous aurions pu aussi implémenter de l'intégration continue car nous avons déployé l'application au dernier moment, ce qui a engendré du stress et plusieurs problèmes. C'est important de la mettre en place dans nos futurs projets car cela fait perdre beaucoup de temps et d'énergie.

Initialement, nous avons beaucoup de doutes sur la structure entre l'API et le modèle. Après avoir discuté à deux reprises avec nos coachs pour identifier les meilleures pratiques, nous avons décidé de créer un DAO pour chaque classe du modèle, afin de communiquer avec l'API. Cependant, nous nous sommes rendus compte que nous répétions les méthodes d'accès aux objets retournés par les API.

Cette pratique comporte des avantages et des inconvénients. Les avantages de cette pratique sont que nous avons uniquement à interagir avec les classes depuis le contrôleur et que la structure était modulaire, facilitant ainsi la maintenance du code. Les inconvénients sont la redondance des méthodes qui sont répétées plusieurs fois et la surcharge du code avec une complexité ajoutée.

Par exemple, nous récupérons les objets de la base via une API externe permettant d'obtenir tous les avis, puis nous appelons une méthode `getAllAvis()` dans l'utilitaire du DAO, qui faisait l'appel à l'API. Parallèlement, nous avons implémenté la méthode statique `Avis.getAll(id_avis)`, utilisant elle aussi l'utilitaire du DAO. Nous aurions pu supprimer cette dernière, qui ne servait pas à grand-chose. Une solution plus efficace aurait été d'utiliser un ORM (Objet Relationnel Mapping) comme Prisma ou Drizzle pour automatiser ces interactions, ce qui nous aurait fait gagner du temps. Au moins, cette expérience nous a permis d'apprendre de nos erreurs pour nos futurs projets.

6. Bilan général

Points forts

Le projet ChâteauTresor a été une réussite sur plusieurs aspects. Tout d'abord, nous avons quasiment atteint tous nos objectifs en termes de fonctionnalités dans les délais impartis. Malgré le fait que l'application ne soit pas finie à 100 %, toutes les fonctionnalités principales ont été implémentées et sont fonctionnelles. Cela démontre une bonne gestion du temps et une répartition efficace des tâches au sein de l'équipe. De plus, la communication fluide et la collaboration entre les membres ont grandement contribué à cette réussite. Enfin, l'utilisation d'outils comme Discord pour la communication et GitLab pour la gestion du code a permis de maintenir une organisation solide tout au long du projet.

Points faibles

Malgré ces succès, nous avons rencontré quelques difficultés, notamment sur la gestion des types, des classes et des DAO. Au début du projet, nous avons eu des doutes sur la manière de structurer ces éléments : fallait-il privilégier des DAO ou des classes ? Cette hésitation nous a fait perdre un peu de temps. De plus, nous n'avions pas suffisamment préparé le modèle lors de la phase de modélisation. En conséquence, lors de l'implémentation des fonctionnalités, nous devions souvent modifier les types et les classes, ce qui a ralenti notre progression.

Améliorations possibles

Si nous devions refaire ce projet, nous ferions quelques ajustements pour gagner en efficacité. Tout d'abord, nous n'utiliserions pas TypeScript. Bien que ce langage offre des avantages en termes de typage fort, il nous a fréquemment fait perdre du temps en raison de sa complexité. À la place, nous opterions pour JavaScript, qui répond parfaitement à nos besoins et simplifierait le développement. Ensuite, nous préparerions mieux le modèle dès la phase de modélisation. Nous créerions un diagramme de classe en amont pour éviter les modifications répétitives pendant l'implémentation.

Nous avons aussi envisagé comme solution de commencer le développement de l'API et du modèle en phase 2 du projet. Cela nous aurait permis de gagner une semaine sur la phase de développement. Un panel admin aurait été plus adapté pour gérer l'application.

Conclusion

Le ressenti global sur ce projet est très positif. Malgré les défis techniques et les ajustements nécessaires, nous sommes fiers du résultat obtenu. L'application répond à nos objectifs initiaux, et nous avons tous apprécié travailler sur ce projet. La collaboration au sein de l'équipe a été excellente, et chacun a pu contribuer selon ses compétences et ses envies. Ce projet nous a permis d'apprendre beaucoup, tant sur le plan technique que organisationnel, et nous en retirons une expérience très enrichissante.

7. Annexes

Sommaire

7. Annexes.....	9
Sommaire.....	9
ReadMe.....	10
Identifiants de connexion.....	10
Identifiants de connexion de l'application.....	10
Identifiants de connexion Supabase :.....	10
Installation et exécution du projet en local.....	10
Prérequis.....	10
Étapes d'installation.....	10
1. Installation des outils requis.....	10
2. Configuration de PowerShell.....	11
3. Clonage du projet.....	11
4. Installation des dépendances.....	11
5. Lancement du projet.....	12
6. Lancement des tests.....	12
Cahier de tests.....	13
Traitements ETL effectués.....	16
1. Phase d'Extraction (Extract).....	16
2. Phase de Transformation (Transform).....	17
3. Phase de Chargement (Load).....	17
Structure du code.....	18
Arborescence du projet.....	18
Organisation du code.....	18
Feuille check-list sécurité.....	20
Diagramme de Gantt.....	22

ReadMe

L'application est disponible en ligne à l'adresse suivante : <https://chateau-tresor.vercel.app>

Identifiants de connexion

Voici les identifiants de connexion pour les différents comptes de l'application (fonctionne en local et en ligne) et de Supabase :

Identifiants de connexion de l'application

1. Compte organisateur
 - login : testmembreequipe@chateautresor.com
 - mot de passe : 12345
2. Compte propriétaire châteaux
 - login : tesproprietairechateau@chateautresor.com
 - mot de passe : 12345
3. Compte participant
 - login : testparticipant@chateautresor.com
 - mot de passe : 12345

Identifiants de connexion Supabase :

1. Compte Supabase
 - login : chateautresor@gmail.com
 - mot de passe : %7Z+N6!+R@PW%RJJa

Installation et exécution du projet en local

Suivez les étapes ci-dessous pour configurer et exécuter le projet sur votre machine Windows.

Prérequis

Assurez-vous d'avoir une connexion internet active et les droits administratifs sur votre machine.

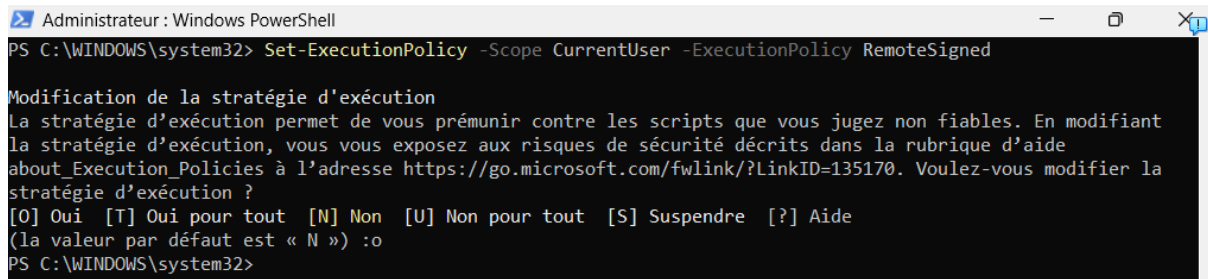
Étapes d'installation

1. Installation des outils requis

1. Installez Visual Studio Code (VSCode)
 - Téléchargez et installez [VSCode](#)
2. Installez Git
 - Téléchargez et installez [Git](#)
3. Installez Node.js
 - Téléchargez et installez [Node.js](#)
4. Redémarrez votre PC

2. Configuration de PowerShell

1. Ouvrez Windows PowerShell en mode administrateur (clic droit > "Exécuter en tant qu'administrateur").
2. Exécutez la commande suivante pour définir la politique d'exécution :
3. Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned

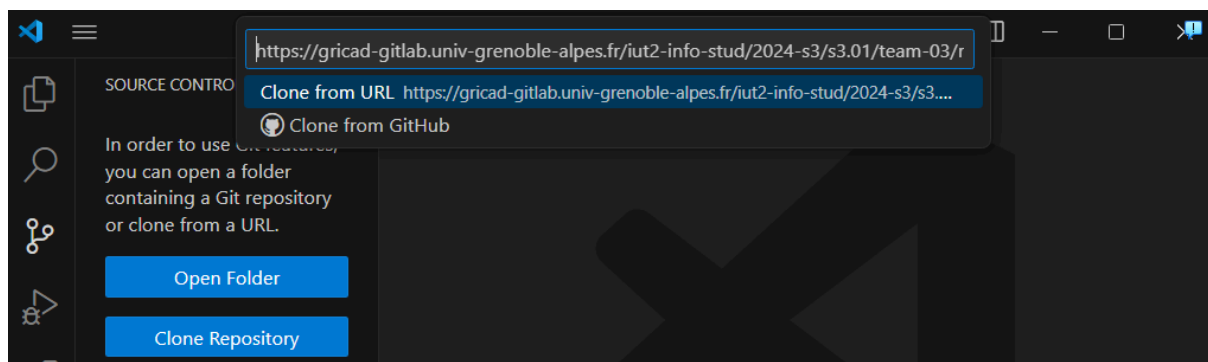


```
Administrateur : Windows PowerShell
PS C:\WINDOWS\system32> Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned

Modification de la stratégie d'exécution
La stratégie d'exécution permet de vous prémunir contre les scripts que vous jugez non fiables. En modifiant
la stratégie d'exécution, vous vous exposez aux risques de sécurité décrits dans la rubrique d'aide
about_Execution_Policies à l'adresse https://go.microsoft.com/fwlink/?LinkID=135170. Voulez-vous modifier la
stratégie d'exécution ?
[O] Oui [T] Oui pour tout [N] Non [U] Non pour tout [S] Suspendre [?] Aide
(la valeur par défaut est « N ») : o
PS C:\WINDOWS\system32>
```

3. Clonage du projet

1. Ouvrez VSCode.
2. Clonez le répertoire Git du projet via :
3. git clone
<https://gricad-gitlab.univ-grenoble-alpes.fr/iut2-info-stud/2024-s3/s3.01/team-03/rendus>

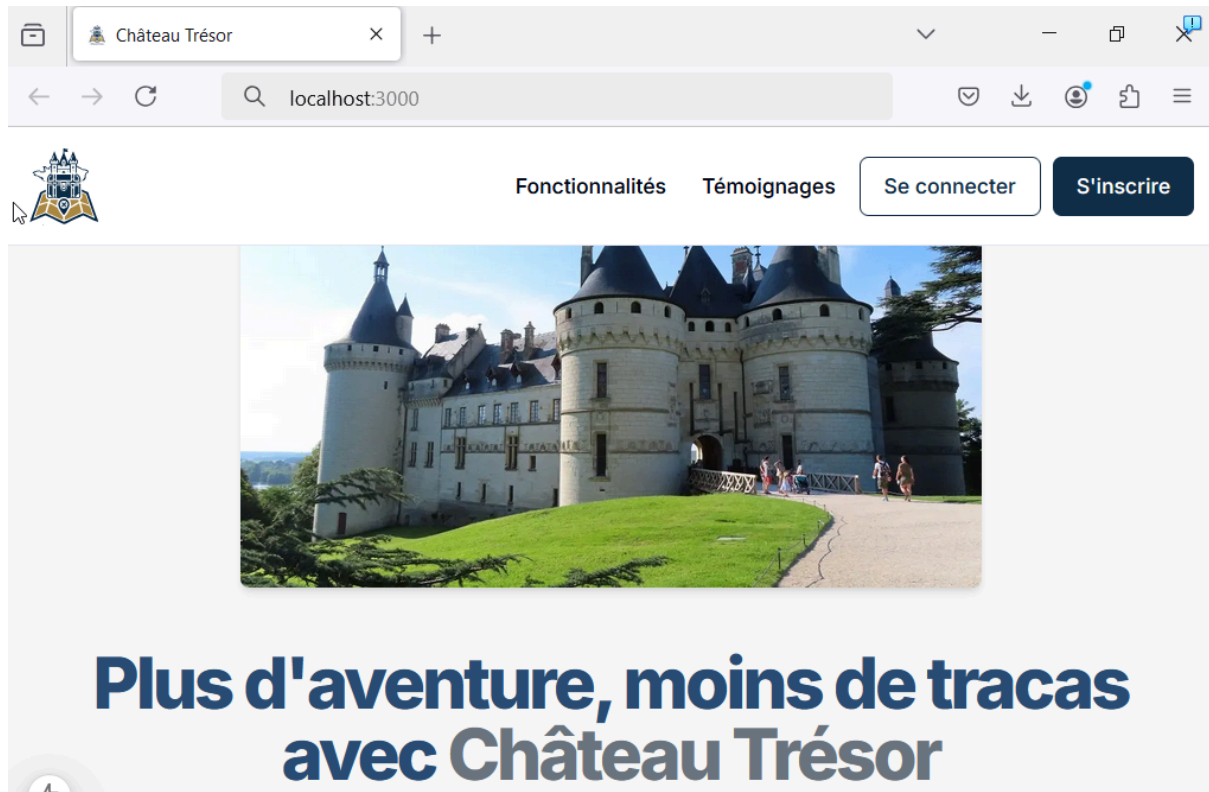


4. Installation des dépendances

1. Ouvrez un terminal dans VSCode.
2. Exécutez la commande suivante pour installer les dépendances du projet :
3. npm install --legacy-peer-deps
4. Mettez à jour npm à la dernière version 6 globalement :
5. npm install -g npm@latest-6
6. Installez les dépendances essentielles pour le projet :
7. npm install next react react-dom --legacy-peer-deps

5. Lancement du projet

1. Pour démarrer le projet en local, exécutez :
2. `npm run dev`
3. Accédez à l'application en ouvrant votre navigateur sur : `http://localhost:3000`



6. Lancement des tests

1. Pour exécuter les tests unitaires du projet, utilisez la commande suivante :
2. `npm run test`
3. Veillez que le serveur local est en route (voir l'étape 6)

Si vous rencontrez des problèmes, vérifiez que toutes les étapes ont été suivies correctement et consultez la documentation officielle des outils utilisés.

Cahier de tests

Catégorie	Classe/Module	Test	Description	Résultat Attendu	Résultat Obtenu	Statut
Tests Unitaires	Avis	Initialisation	Vérifie que les propriétés de l'avis sont correctement initialisées.	Les propriétés sont correctement définies.	Les propriétés sont correctement définies.	Passé
		Lecture par ID	Teste la récupération d'un avis par son ID.	Un avis est retourné avec les bonnes données.	Un avis est retourné avec les bonnes données.	Passé
		Ajout de like	Teste l'incrément du nombre de likes.	Le nombre de likes est incrémenté de 1.	Le nombre de likes est incrémenté de 1.	Passé
	Chasse	Initialisation	Vérifie que les propriétés de la chasse sont correctement initialisées.	Les propriétés sont correctement définies.	Les propriétés sont correctement définies.	Passé
		Ajout de participant	Teste l'ajout d'un participant à une chasse.	Le participant est ajouté avec succès.	Le participant est ajouté avec succès.	Passé
	Enigme	Calcul du temps moyen	Teste le calcul du temps moyen de résolution des énigmes.	Le temps moyen est correctement calculé.	Le temps moyen est correctement calculé.	Passé

	Indice	Initialisation	Vérifie que les propriétés de l'indice sont correctement initialisées.	Les propriétés sont correctement définies.	Les propriétés sont correctement définies.	Passé
	Participant	Calcul des statistiques	Teste le calcul des scores, durées moyennes, et autres statistiques.	Les statistiques sont correctement calculées.	Les statistiques sont correctement calculées.	Passé
Tests Fonctionnels	Inscription	Inscription d'un organisateur	Teste le processus complet d'inscription, de la création du compte à la validation.	Le compte est créé et validé avec succès.	Le compte est créé et validé avec succès.	Passé
	Résolution d'énigmes	Résolution d'une énigme	Teste la résolution d'une énigme par un participant.	L'énigme est résolue avec succès.	L'énigme est résolue avec succès.	Passé
	Paiement	Paiement avec Stripe	Teste le processus de paiement et l'enregistrement des transactions.	La transaction est enregistrée avec succès.	La transaction est enregistrée avec succès.	Passé
Tests d'API	Supabase	Récupération des données	Teste la récupération des données depuis Supabase.	Les données sont correctement récupérées.	Les données sont correctement récupérées.	Passé
		Ajout, modification, suppression	Teste les opérations CRUD sur les données.	Les opérations CRUD fonctionnent correctement.	Les opérations CRUD fonctionnent correctement.	Passé

	Permissions	Vérification des permissions	Teste les règles d'accès à la base de données.	Seuls les utilisateurs autorisés peuvent accéder aux données.	Seuls les utilisateurs autorisés peuvent accéder aux données.	Passé
--	-------------	------------------------------	--	---	---	-------

Traitements ETL effectués

L'application ChateauTrésor repose sur des données fiables et enrichies concernant les châteaux de la Loire afin de fournir aux utilisateurs une expérience immersive et informative. Pour cela, un processus ETL (Extract, Transform, Load) est mis en place afin d'extraire les données disponibles sur des sources publiques libres de droit, de les transformer en informations exploitables, puis de les charger dans la base de données de l'application.

1. Phase d'Extraction (Extract)

L'extraction consiste à récupérer les données des châteaux à partir de sources fiables telles que :

- **Fichier CSV de data.gouv.fr** : contenant des informations sur les immeubles protégés au titre des Monuments Historiques, incluant le nom, la localisation, le type de monument et les coordonnées GPS.
- **Fichier CSV des données manquantes via l'API de Wikipédia** : récupération des données manquantes comme l'image, et la description en utilisant l'API de Wikipédia.

Les étapes d'extraction incluent :

1. Téléchargement du fichier CSV depuis data.gouv.fr.
2. Utilisation d'un script python pour récupérer uniquement les données souhaitées, on récupère uniquement certaines colonnes du fichier (Nom, coordonnées, adresse...).
3. Utilisation d'un script python pour effectuer des requêtes à l'API de Wikipédia pour récupérer les données manquantes des châteaux (image et description)
4. Utilisation d'un script python pour combiner les deux fichiers csv.

Script python pour les requêtes sur l'API de Wikipédia :

```
def recuperer_informations_principales(titre_page):
    """
    Récupère depuis l'API Wikipédia (via prop=extracts|pageimages|info|coordinates) :
    - Extrait (intro)
    - URL de l'article
    - URL de l'image (si disponible)
    - Coordonnées géographiques (si disponibles)
    """
    url = "https://fr.wikipedia.org/w/api.php"
    params = {
        "action": "query",
        "prop": "extracts|pageimages|info|coordinates",
        "exintro": True,
        "explaintext": True,
        "piprop": "original",
        "inprop": "url",
        "format": "json",
        "titles": titre_page
    }

    response = requests.get(url, params=params)
    data = response.json()

    pages = data.get("query", {}).get("pages", {})
    if not pages:
        return None

    for page_id, page_info in pages.items():
        if int(page_id) < 0:
            # Page non valide ou page spéciale
            continue

        extrait = page_info.get("extract", None)
        lien_page = page_info.get("fullurl", None)

        # Image
        lien_image = None
        if "original" in page_info:
```

2. Phase de Transformation (Transform)

Une fois les données extraites, elles doivent être nettoyées, enrichies et structurées afin d'être utilisées efficacement par l'application.

Opérations effectuées :

- **Nettoyage des données** : suppression des doublons, suppression des données aberrantes (coordonnées non renseignées, nom non renseigné etc...) via un script python.
- **Sélection des données essentielles** : conservation uniquement des colonnes "nom", "adresse", "latitude" et "longitude" que l'on combine dans une seule colonne "coordonnees" en utilisant un script python.
- **Enrichissement** : utilisation du fichier csv obtenu via les requêtes à l'API de Wikipédia pour récupérer les images et les descriptions des châteaux pour chaque château avec un script python et les insérer dans le fichier CSV obtenu précédemment.

3. Phase de Chargement (Load)

Après transformation, les données ont été extraites dans un fichier CSV, qui a ensuite été chargé dans la table `chateau` de notre projet Supabase.

Structure du code

Arborescence du projet

Le projet *ChateauTresor* est organisé de manière modulaire pour faciliter la maintenance et l'évolution du code. L'arborescence suit les bonnes pratiques de **Next.js 15.1.3**, en tirant parti des fonctionnalités modernes comme le **React Server Components (RSC)**, les **Server Actions**, et une gestion optimisée des routes.

Organisation du code

Le code est structuré selon une architecture modulaire, séparant les responsabilités pour une meilleure lisibilité et maintenabilité. Cette organisation est alignée avec les fonctionnalités de Next.js 15.1.3, notamment l'utilisation des Server Components pour optimiser les performances et réduire la taille du bundle côté client. Voici les principaux éléments :

Dossier `app` :

Ce dossier utilise le **nouveau système de routage basé sur les dossiers** introduit dans Next.js 13 et optimisé dans les versions ultérieures. Chaque sous-dossier correspond à une route ou une fonctionnalité spécifique (exemple : `organismes`, `participants`, `auth`).

- `layout.tsx` : Définit la structure commune à toutes les pages, en utilisant les **Server Components** pour le rendu côté serveur.
- `page.tsx` : Contient le code de la page d'accueil, avec une combinaison de **Server Components** et de **Client Components** si nécessaire.

Dossier `components` :

Il regroupe tous les composants réutilisables dans l'application. Ces composants sont organisés par fonctionnalité (exemple : `auth`, `organismes`, `participants`).

- `Footer.tsx` : Un composant pour afficher le footer de notre application, conçu comme un **Client Component** pour interagir avec l'utilisateur.

Dossier `utils` :

Ce dossier contient les utilitaires et les fonctions métier. Il est divisé en sous-dossiers pour une meilleure organisation :

- `dao` : Gestion des accès aux données, optimisée pour les **Server Actions**.

- **db** : Interactions avec la base de données, utilisant des requêtes asynchrones côté serveur.
- **stripe** : Intégration de Stripe pour les paiements, avec des fonctions sécurisées côté serveur.
- **supabase** : Configuration et utilisation de Supabase, alignée avec les bonnes pratiques de Next.js 15.1.3.

Dossier **constants** :

Il regroupe les constantes utilisées dans tout le projet, comme les URLs d'API ou les messages d'erreur.

Dossier **types** :

Il contient les définitions TypeScript pour les types de données utilisés dans l'application, avec une prise en charge optimale des types génériques et des interfaces.

Dossier **classes** :

Ce dossier contient toutes les classes métier de l'application, encapsulant la logique spécifique à chaque entité. Ces classes sont utilisées pour modéliser les données et les comportements associés, en respectant les principes de la programmation orientée objet (POO). Voici les classes principales :

1. **Organisateur.ts** :

Représente un organisateur de chasses au trésor. Cette classe étend la classe **User** et ajoute des propriétés spécifiques comme **eventsCreated** (liste des événements organisés) et des méthodes pour gérer ces événements.

2. **Participant.ts** :

Représente un participant à une chasse au trésor. Cette classe étend également **User** et inclut des propriétés comme **eventsJoined** (liste des événements auxquels le participant s'est inscrit) et des méthodes pour gérer les inscriptions.

Autres dossiers :

- **constants** : Regroupe les constantes utilisées dans tout le projet.
- **types** : Contient les définitions TypeScript pour les types de données utilisés dans l'application.
- **public** : Contient les ressources statiques, comme les images ou les icônes.

Feuille check-list sécurité

Attaque possible	Défense(s)	Mesures de défense mises en place
Vulnérabilités dans les logiciels utilisés		
Exploitation de vulnérabilités logicielles dans des logiciels fournis par Debian.	.	
Exploitation de vulnérabilités logicielles dans des logiciels tiers.	Mise à jour de la version de NodeJS automatique par Vercel	Nous utilisons <code>npm</code> avec un fichier <code>package-lock.json</code> pour figer les versions des dépendances. Les mises à jour sont surveillées via des outils comme <code>npm audit</code> et appliquées manuellement après tests.
Confidentialité et intégrité des données circulant sur les réseaux		
Accès aux données circulant sur le réseau entre vos clients et votre serveur Web par écoute passive (sniffing).	Mise en place de HTTPS avec certificat signé géré par Vercel	
Accès aux données circulant sur le réseau entre vos clients et votre serveur Web par écoute	Mise en place de HTTPS avec certificat validé par une autorité de certification. Nécessite une	Nous utilisons HTTPS avec un certificat validé.

active (attaque man-in-the-middle).	adresse IP publique et un enregistrement DNS.	
Accès aux données circulant sur le réseau entre le serveur Web et le serveur de BD.	Vérifier que la connexion (PDO ou autre) utilise bien le protocole TLS.	Toutes les connexions entre notre application et Supabase utilisent TLS pour chiffrer les échanges de données.
Contrôle d'accès		
Tentatives d'accès à votre serveur de BD sans passer par l'application Web.		Nous avons configuré Supabase pour n'autoriser les connexions à la base de données que depuis les adresses IP des serveurs Vercel. Un pare-feu est également en place pour bloquer les accès non autorisés.
Tentatives pour deviner les mots de passe des utilisateurs en mode on-line.	Mise en place d'un nombre maximal d'essais successifs par minute.	Nous avons configuré Supabase pour limiter les tentatives de connexion à 5 essais par minute, avec un blocage temporaire en cas de dépassement.
Tentatives pour deviner les mots de passe des utilisateurs en mode off-line.	Stockage des mots de passe avec hachage (obligatoire), salage (indispensable) et poivrage (bon à avoir).	Les mots de passe sont hachés avec bcrypt pour rendre les attaques par force brute hors ligne inefficaces.
Vulnérabilités dans les logiciels développés		
Injection SQL.	Utilisation de l'API de supabase empêchant les injections	Nous requêtons la base uniquement avec l'API de

		supabase pour empêcher les injections SQL.
Failles XSS.	Filtrage des données saisies par les utilisateurs.	Toutes les entrées utilisateur sont filtrées et échappées automatiquement par react avec les <code>{}</code> avant d'être affichées ou stockées dans la base de données. De plus l'implémentation de DOMPurify serait nécessaire ! (pas encore fait)
Failles CSRF/XSRF.	Jeton de validité dans les formulaires.	Nous implémenterons des jetons CSRF dans tous les formulaires pour empêcher les attaques intersites. (pas encore fait)

Diagramme de Gantt

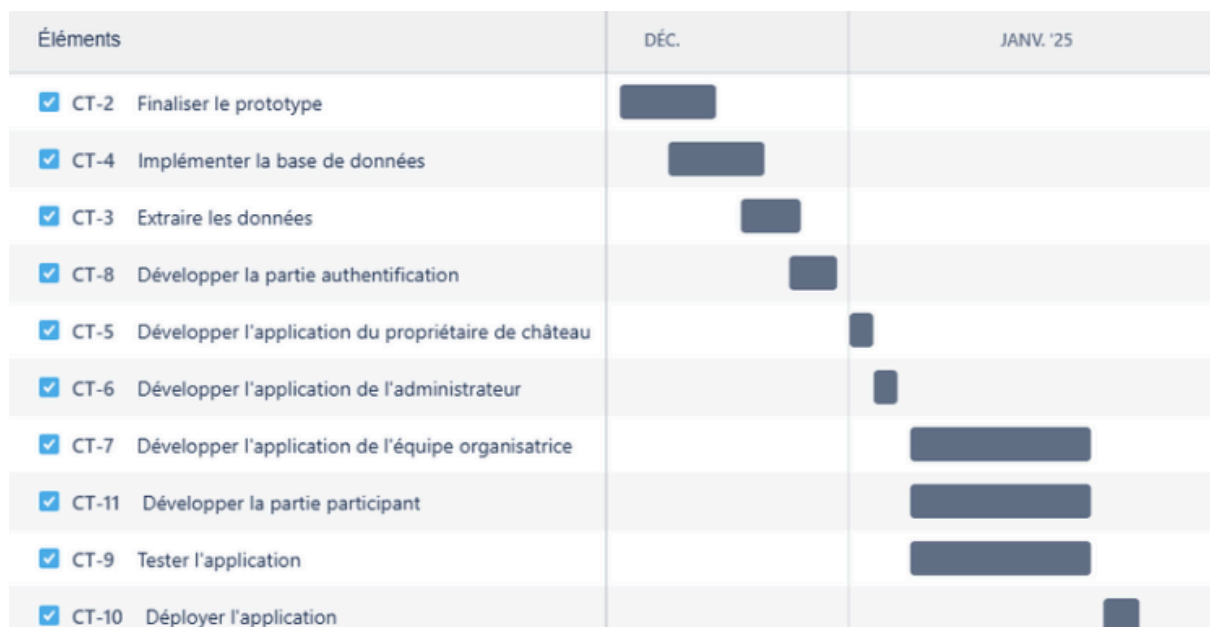


Diagramme de Gant du projet ChateauTresor

Diagramme détaillé détaillant l'organisation globale de la phase de réalisation. Les différentes phases et les durées prévisionnelles sont détaillées à chaque étape.