Présentation de l'Application ReactJS et Express

Cette présentation détaillera le développement d'une application d'un site pour garagiste construite à l'aide des technologies ReactJS et Express. Nous explorerons la conception, l'architecture et les fonctionnalités clés de cette solution innovante.

by Brian MOREAU

Modélisation de l'Application

Architecture

```
/client
   /dist
   /node_modules
   /public
      vite.svg
   /src
      /assets
         react.svg
      /components
         AuthPage.css
         AuthPage.jsx
         Header.css
         Header.jsx
         HomePage.css
         HomePage.jsx
         SigninForm.css
         SigninForm.jsx
         SignupForm.css
         SignupForm.jsx
      /utils
         cookies.jsx
      App.css
      App.jsx
      index.css
      main.jsx
   .env.development
   .env.production
   .eslintrc.cjs
   .gitignore
   index.html
   package.json
   package-lock.json
   README.md
   vite.config.js
/configs
   garage.sql
/node_modules
/src
   authUtils.js
/tests
   auth.test.js
   authUtils.test.js
package.json
package-lock.json
server.js
```

Cas d'utilisation

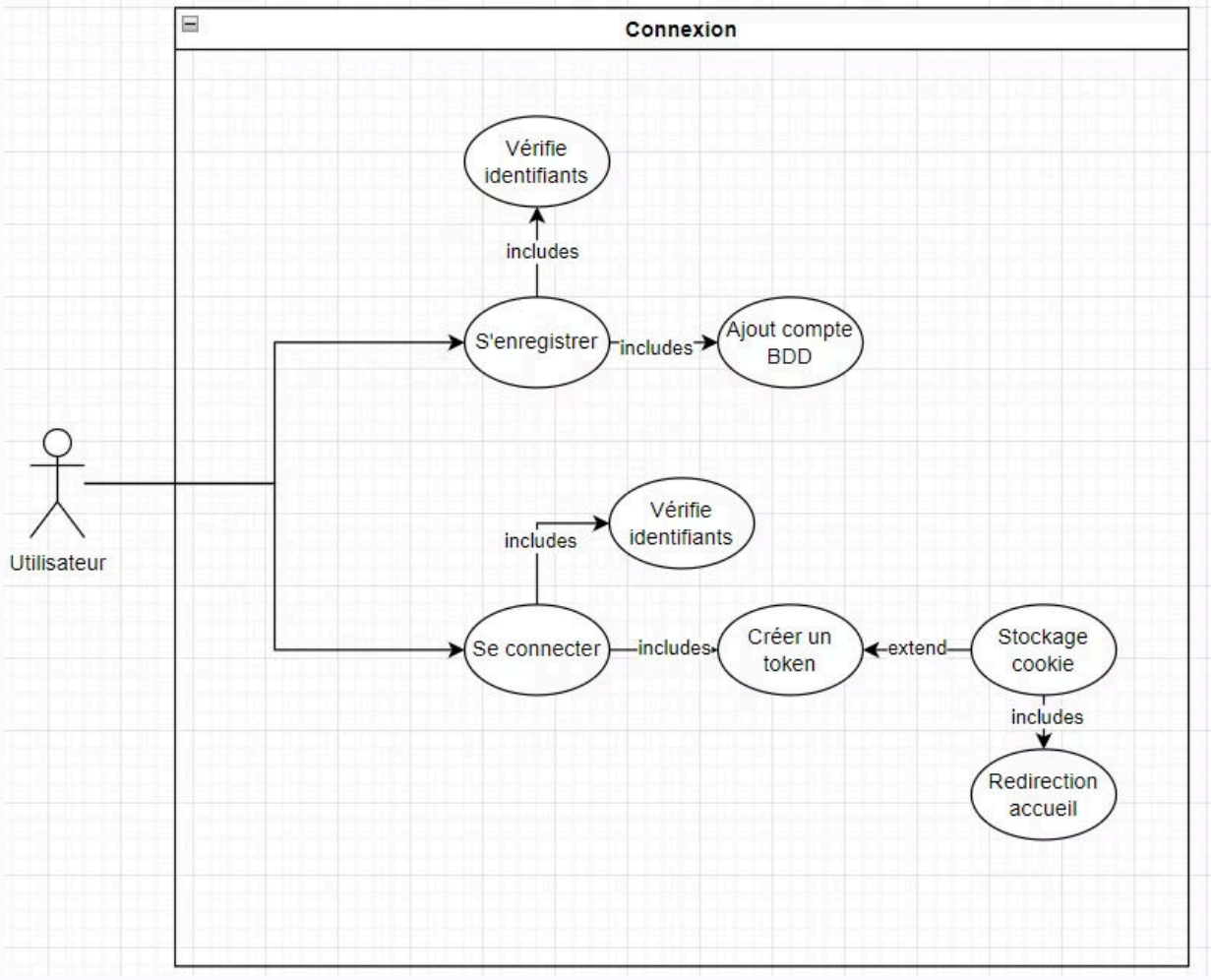
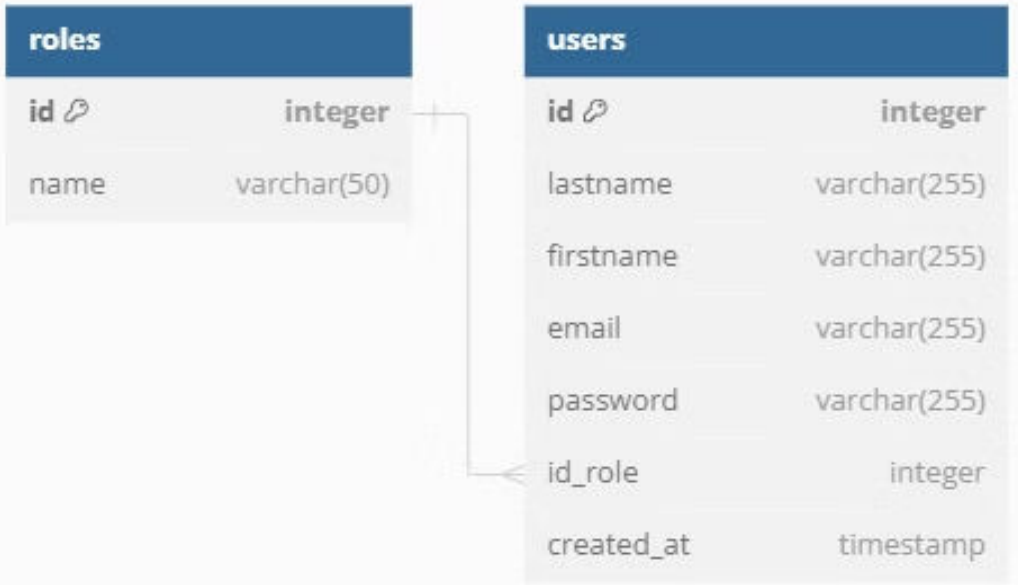
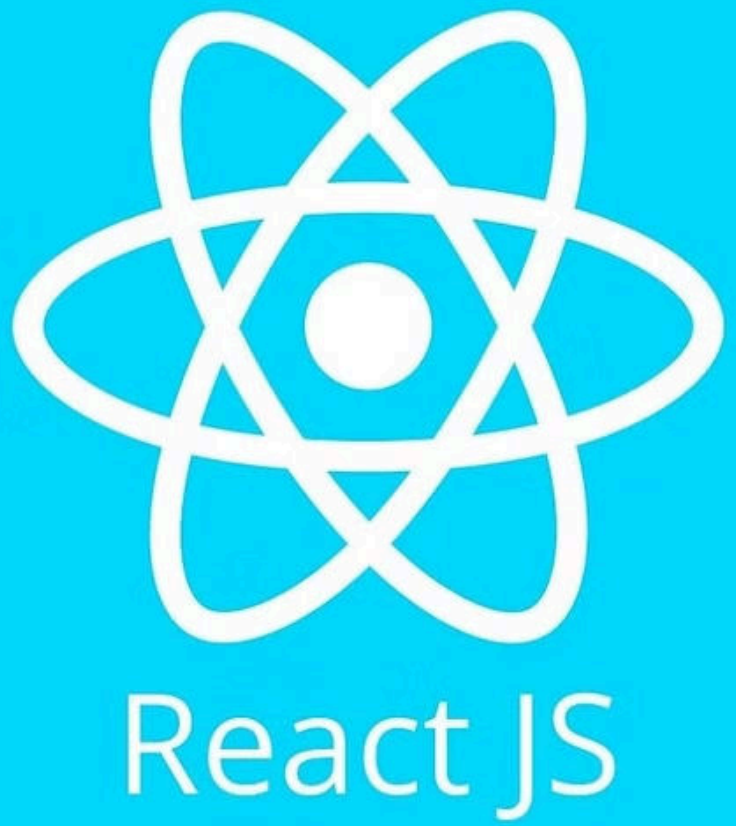


Schéma de la Base de Données





Choix des Composants client

1 ReactJS

Bibliothèque pour construire l'interface utilisateur de façon dynamique et réactives

2 React Router

Bibliothèque permettant la gestion des routes et des vues dans une application React

3 Vite

Outil de construction moderne pour les applications front-end conçu pour un développement rapide et une construction optimisée pour la production

Choix des Technologies serveur

Express

Express a été choisi pour créer une API RESTful performante et évolutive.

Base de données MySQL

Système de gestion de base de données (SGBD) relationnelle open-source.
Stocke les données sous forme de tables.

Bcrypt

Permet de hacher les mots de passe de manière sécurisée en utilisant l'algorithme bcrypt

Json Web Token (JWT)

Bibliothèque permettant de créer et vérifier des JWT, utilisés pour la gestion des sessions utilisateur et l'authentification

Implémentation de la sécurité

1

Gestion des mots de passe

Hashage avec bcrypt pour sécuriser les mots de passe avant stockage en base de données

2

Authentification

- Mise en place d'un JWT pour la gestion des sessions et la protection des routes.
- Middleware pour vérifier la présence et la validité du JWT

3

Cross-Origin Resource Sharing

Configuration des CORS avec la librairie "cors" d'Express

- Accepte uniquement les requêtes provenant du site

Implémentation de la sécurité

1

Content Security Policy

La CSP aide à détecter certaines types d'attaques comme les XSS. Elle permet de spécifier les sources de contenu autorisées pour une page Web

2

Protection contre les failles XSS

Utilisation des fonctions d'échappements pour éviter l'exécution de scripts malveillants dans les pages Web

3

Protection contre les attaques CSRF

Ajout de token CSRF dans les formulaires pour éviter les attaques CSRF

4

Protection contre l'injection SQL

Utilisation des requêtes paramétrées pour se protéger des injections en séparant les données des commandes SQL

Phase de tests

Tests fonctionnels

Test le comportement complet d'un composant avec JEST

- Test de connexion :

```
describe("POST /api/signin", () => {
  Run | Debug
  it("doit retourner un code 200 et un token valide", async () => {
    const response = await request(app).post("/api/signin").send({
      email: "email@email.fr",
      password: "Password",
    });

    expect(response.status).toBe(200);
    expect(response.body).toHaveProperty("token");
  });

  Run | Debug
  it("doit retourner un code 401 pour mauvais identifiants", async () => {
    const response = await request(app).post("/api/signin").send({
      email: "email@email.fr",
      password: "wrongpassword",
    });

    expect(response.status).toBe(401);
  });
```

Résultat :

```
PASS tests/auth.test.js
  POST /api/signin
    √ doit retourner un code 200 et un token valide (97 ms)
    √ doit retourner un code 401 pour mauvais identifiants (25 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        1.031 s
Ran all test suites.
```


Phase de tests

Tests unitaires

Test le comportement de code isolées avec JEST

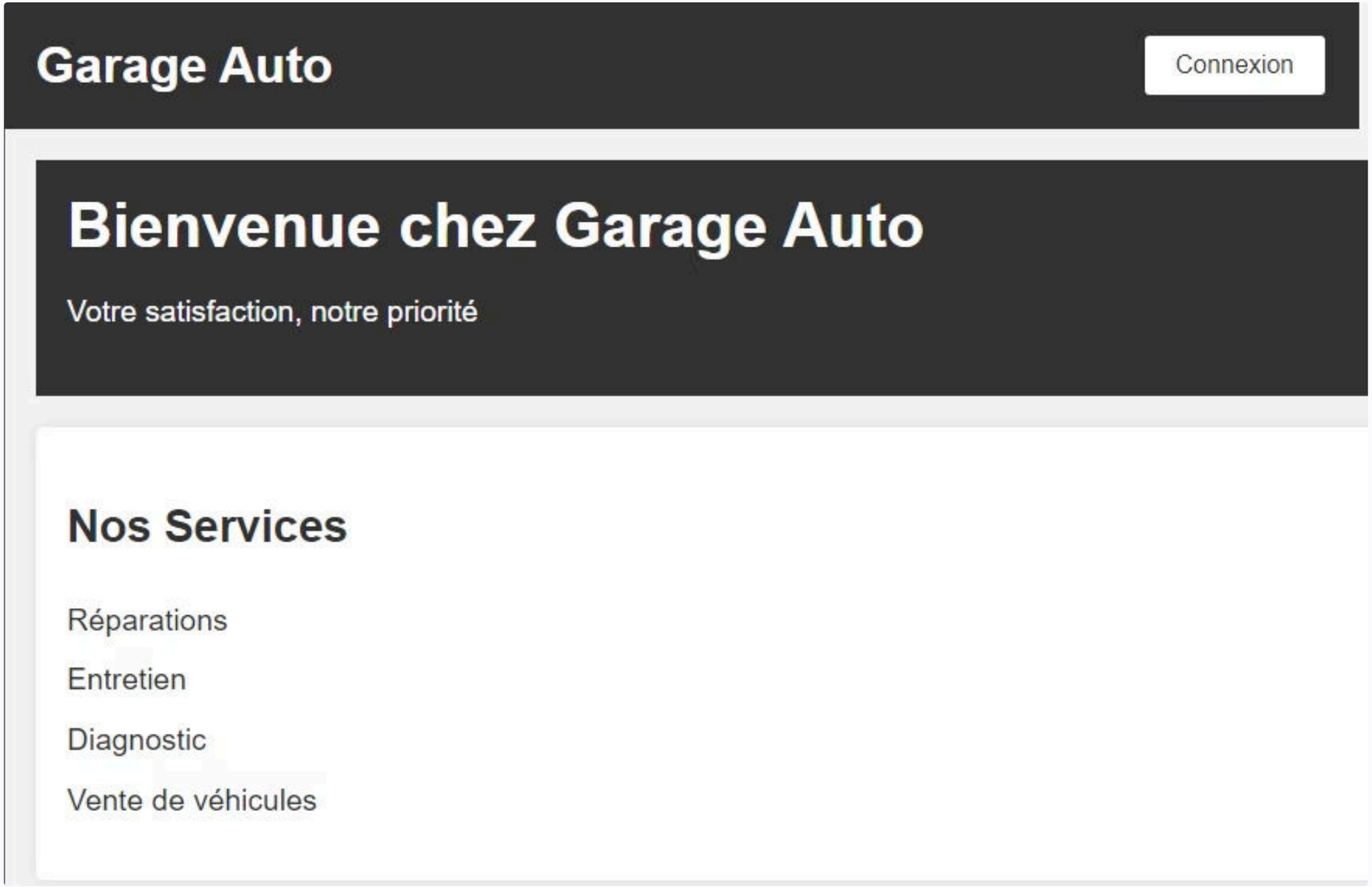
- Test de hachage des mots de passe:

```
describe("hashPassword", () => {
  Run | Debug
  it("doit hasher le mot de passe", () => {
    const password = "password";
    const hashedPassword = hashPassword(password);

    expect(hashedPassword).not.toBe(password);

    const isMatch = bcrypt.compareSync(password, hashedPassword);
    expect(isMatch).toBe(true);
  });
});
```

Résultat :

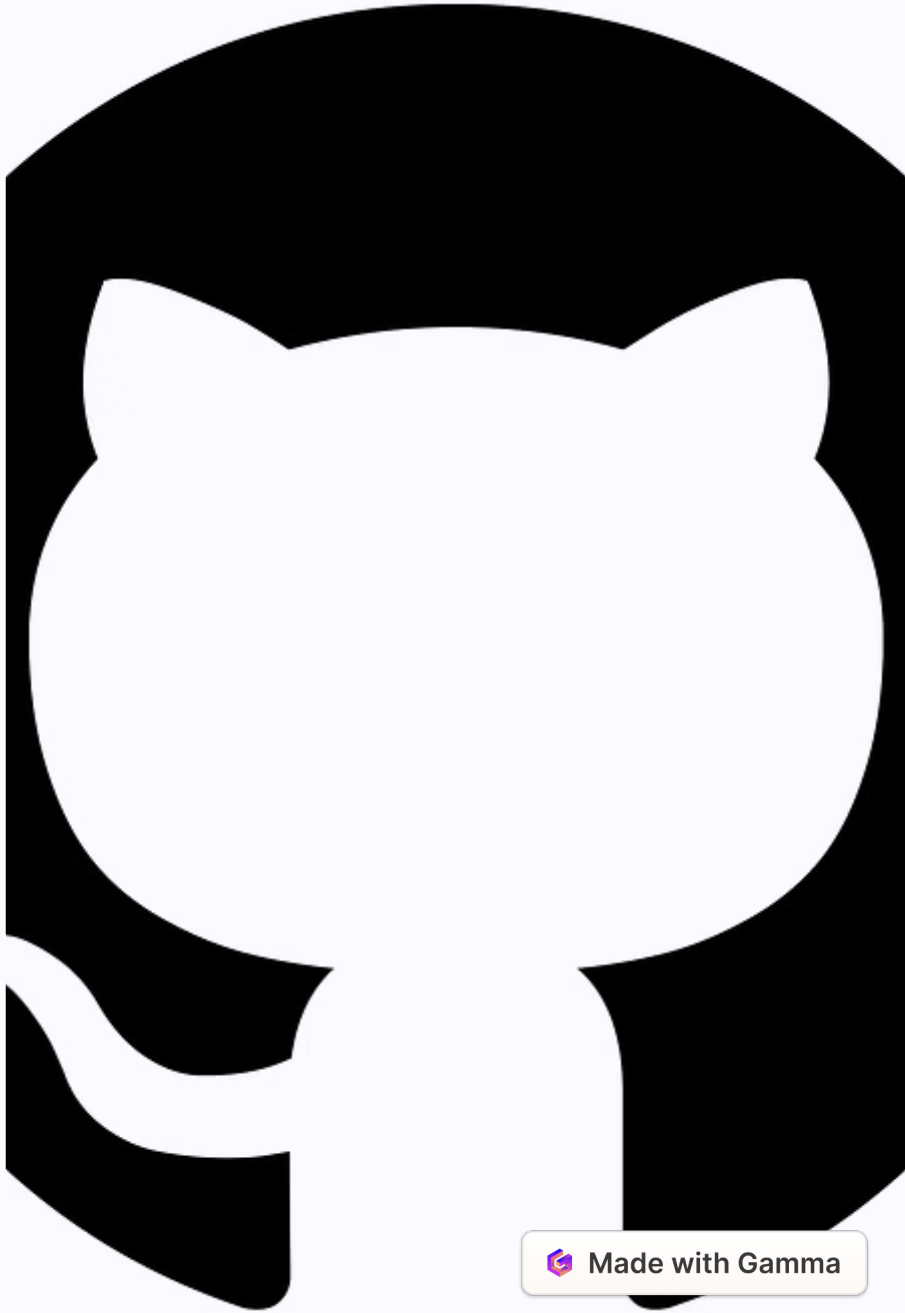
```
 PASS  tests/authUtils.test.js
  hashPassword
    √ doit hasher le mot de passe (44 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.406 s, estimated 1 s
```


Démonstration de l'Application



Fichiers Sources

Les fichiers sources de l'application sont disponibles sur le **dépôt GitHub**.
Vous y trouverez le code de l'application ReactJS et l'API Express



Conclusion et Perspectives

Cette application ReactJS et Express offre une solution sécurisée et interactif. Nous avons mis en place des mesures de sécurité, comme les tokens JWT, les CSP, .. assurant la protection des données. Le système est testé et prêt pour une utilisation fiable et évolutive