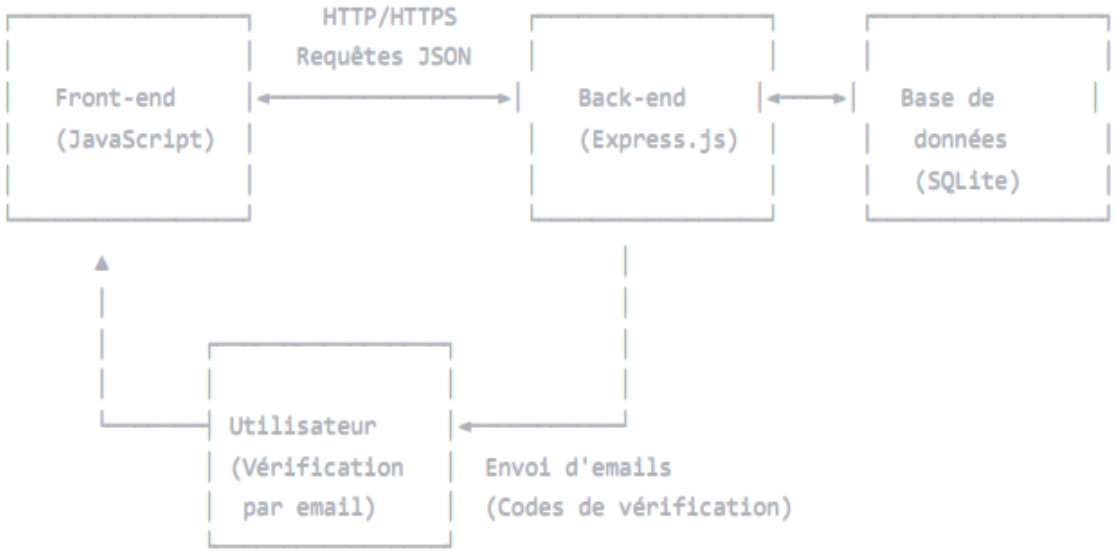
Documentation Technique - Application Web avec Authentification

Table des matières

Architecture générale

L'application est construite selon une architecture client-serveur avec un système d'authentification à double facteur :



- **Flux d'authentification** : L'application utilise un système à double facteur avec des codes de vérification envoyés par email
- **Gestion des sessions** : Authentification basée sur JWT stockés dans des cookies HTTP-only
- **Architecture modulaire** : Séparation claire entre le front-end et le back-end

Structure du projet

```
APPLICATION/
├── back-end/
│   ├── node_modules/     # Dépendances Node.js
│   ├── prisma/
│   │   ├── migrations/   # Migrations de base de données
│   │   ├── dev.db        # Base de données SQLite
│   │   └── schema.prisma # Schéma Prisma
│   ├── .env              # Variables d'environnement
│   ├── .gitignore        # Fichiers ignorés par Git
│   ├── package-lock.json #  versions de dépendances
│   ├── package.json      # Configuration du projet Node.js
│   └── server.js         # Point d'entrée du serveur Express.js
├── front-end/
│   ├── a2f/              # Double authentification
│   │   ├── app.js
│   │   ├── index.html
│   │   └── style.css
│   ├── main/             # Page principale
│   │   ├── app.js
│   │   ├── index.html
│   │   └── style.css
│   ├── pwd-forget/       # Page de récupération de mot de passe
│   │   ├── app.js
│   │   ├── index.html
│   │   └── style.css
│   ├── pwd-reset/        # Page de réinitiali. de mot de passe
│   │   ├── app.js
│   │   ├── index.html
│   │   └── style.css
│   ├── sign-in/          # Page de connexion
│   │   ├── app.js
│   │   ├── index.html
│   │   └── style.css
│   └── sign-up/          # Page d'inscription
│       ├── app.js
│       ├── index.html
│       └── style.css
```

Back-end

Configuration serveur

Le serveur est implémenté avec Express.js et intègre plusieurs middlewares essentiels :

```javascript
const express = require('express');
const cors = require("cors");
const cookieParser = require("cookie-parser");
const app = express();

app.use(cors({
    origin: "http://127.0.0.1:5500",
    methods: ["GET", "POST"],
    allowedHeaders: ["Content-Type"],
    credentials: true
}));

app.use(express.json());
app.use(cookieParser());

app.listen(3000, () => {
    console.log("Serveur démarré sur http://localhost:3000");
});
```

Middlewares clés :

- `cors` : Configuration sécurisée pour les requêtes cross-origin (depuis le front-end)
- `express.json` : Parsing des requêtes au format JSON
- `cookieParser` : Gestion des cookies pour le stockage des JWT

Système d'authentification

L'application implémente un système d'authentification complet avec :

1. **Inscription (`/register`)** : Création d'un compte utilisateur avec validation par email

2. **Vérification (`/verify`)** : Confirmation du compte par code à 6 chiffres envoyé par email
3. **Connexion (`/login`)** : Authentification avec mot de passe + code par email (2FA)
4. **Déconnexion (`/logout`)** : Suppression du cookie JWT
5. **Vérification JWT (`/checkJWT`)** : Validation du token d'authentification
6. **Récupération de mot de passe (`/forgetPwd` et `/resetPwd`)** : Procédure complète de réinitialisation

Exemple du flux d'inscription :

```javascript
app.post('/register', async (req, res) => {
    const { email, password } = req.body;

    // Vérifie si l'utilisateur existe déjà
    const existingUser = await prisma.user.findUnique({ where: { email } });
    if (existingUser) return res.status(400).json({ message: "Utilisateur déjà inscrit" });

    // Hash du mot de passe
    const hashedPassword = await bcrypt.hash(password, 10);
    const verificationCode = Math.floor(100000 + Math.random() * 900000).toString(); // Code à

    // Création de l'utilisateur
    await prisma.user.create({
        data: { email, password: hashedPassword, code: verificationCode }
    });

    // Création du token JWT
    const token = generateJWT(email, false);

    // Sauvegarde du token dans le cookie
    res.cookie('jwt', token, {
        httpOnly: true,
        secure: true,
        sameSite: 'none',
        maxAge: 3600000
    });

    // Réponse envoyé à l'utilisateur
    res.json({ message: "Utilisateur créé, code envoyé par email" });

    // Envoi de l'e-mail
    await sendEmail(email, verificationCode);
});
```

Services d'envoi de mails

L'application utilise Nodemailer pour envoyer des codes de vérification par email :

javascript

```javascript
const transporter = nodemailer.createTransport({
    host: process.env.MAILTRAP_HOST,
    port: process.env.MAILTRAP_PORT,
    auth: {
        user: process.env.MAILTRAP_USER,
        pass: process.env.MAILTRAP_PASS
    }
});

async function sendEmail(to, code) {
    try {
        const info = await transporter.sendMail({
            from: process.env.EMAIL_USER,
            to,
            subject: "Votre code de vérification",
            text: `Votre code de vérification est : ${code}`
        });

        console.log("Email envoyé : ", info.messageId);
    } catch (error) {
        console.error("Erreur lors de l'envoi du mail :", error);
    }
}
```

Configuration requise dans le fichier `.env` :

```
MAILTRAP_HOST=smtp.mailtrap.io
MAILTRAP_PORT=2525
MAILTRAP_USER=votre_user_mailtrap
MAILTRAP_PASS=votre_password_mailtrap
EMAIL_USER=from@example.com
```

Gestion des tokens JWT

L'application utilise des JSON Web Tokens (JWT) pour maintenir les sessions utilisateurs :

```javascript
function generateJWT(email, authenticated = false) {
    const payload = {
        email,
        authenticated
    }

    const token = jwt.sign(
        payload,
        process.env.JWT_SECRET,
        {expiresIn: '1h'}
    );

    return token;
}
```

Les tokens contiennent deux informations clés :

- `email` : L'identifiant de l'utilisateur
- `authenticated` : État de vérification (true/false)

Les tokens sont stockés dans des cookies HTTP-only pour une sécurité renforcée.

Front-end

Le front-end de l'application est organisé en modules distincts pour chaque fonctionnalité :

- **sign-up** : Formulaire d'inscription
- **sign-in** : Formulaire de connexion avec support 2FA
- **pwd-forget** : Demande de récupération de mot de passe
- **pwd-reset** : Réinitialisation de mot de passe
- **main** : Zone principale de l'application (accessible après authentification)
- **a2f** : `Double authentification`

Chaque module suit une structure commune avec :

- `app.js` : Logique JavaScript spécifique
- `index.html` : Structure de la page
- `style.css` : Styles spécifiques au module

Bonnes pratiques front-end recommandées

1. **Communication avec l'API**

```javascript
// Exemple de fonction d'appel API pour l'inscription
async function register(email, password) {
    try {
        const response = await fetch('http://localhost:3000/register', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({ email, password }),
            credentials: 'include' // Important pour les cookies
        });

        const data = await response.json();
        return { success: response.ok, data };
    } catch (error) {
        return { success: false, error: error.message };
    }
}
```

2. **Vérification d'authentification**

javascript

```javascript
// Vérifier si l'utilisateur est authentifié
async function checkAuth() {
    try {
        const response = await fetch('http://localhost:3000/checkJWT', {
            method: 'GET',
            credentials: 'include'
        });

        return response.ok;
    } catch (error) {
        return false;
    }
}

// Rediriger si non authentifié
async function protectRoute() {
    const isAuthenticated = await checkAuth();
    if (!isAuthenticated) {
        window.location.href = '/sign-in/index.html';
    }
}
```

Base de données

Structure Prisma

Basé sur l'utilisation de l'ORM Prisma avec SQLite, voici une structure probable du schéma :

```
prisma

// prisma/schema.prisma
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "sqlite"
  url      = "file:./dev.db"
}

model User {
  id        Int      @id @default(autoincrement())
  email     String   @unique
  password  String
  code      String?  // Code de vérification temporaire
  verified  Boolean  @default(false)
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}
```

Utilisation dans le code

javascript

```javascript
const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();

// Exemple: recherche d'un utilisateur
const user = await prisma.user.findUnique({
    where: { email }
});

// Exemple: mise à jour d'un utilisateur
await prisma.user.update({
    where: { email },
    data: { verified: true, code: null }
});
```

API Reference

Format de réponse standard

Toutes les réponses API suivent un format standard :

```
{
  "message": "Description du résultat de l'opération"
}
```

En cas d'erreur, le code HTTP approprié est retourné avec un message explicatif.

Points d'entrée API

Authentification et gestion des utilisateurs

Endpoint	Méthode	Description	Corps de la requête	Réponse
`/register`	POST	Inscription d'un utilisateur	`{ "email": "user@example.com", "password": "secret" }`	`{ "message": "Utilisateur créé, code envoyé par email" }`
`/verify`	POST	Vérification du code reçu par email	`{ "code": "123456" }`	`{ "message": "Utilisateur authentifié" }`
`/login`	POST	Connexion utilisateur	`{ "email": "user@example.com", "password": "secret" }`	`{ "message": "Code de connexion envoyé par email" }`
`/logout`	GET	Déconnexion utilisateur	-	`{ "message": "Déconnexion réussie" }`
`/checkJWT`	GET	Vérification du token JWT	-	`{ "message": "Accès autorisé" }`
`/forgetPwd`	POST	Demande de réinitialisation de mot de passe	`{ "email": "user@example.com" }`	`{ "message": "Code de connexion envoyé par email" }`
`/resetPwd`	POST	Réinitialisation du mot de passe	`{ "password": "nouveau_mot_de_passe", "code": "123456" }`	`{ "message": "Utilisateur créé, code envoyé par email" }`

Installation et configuration

Prérequis

- Node.js (v14 ou supérieur)
- npm ou yarn

Variables d'environnement

Créez un fichier `.env` dans le dossier back-end avec les variables suivantes :

```
# JWT
JWT_SECRET=votre_clé_secrète_jwt

# Email (Mailtrap)
MAILTRAP_HOST=smtp.mailtrap.io
MAILTRAP_PORT=2525
MAILTRAP_USER=votre_user_mailtrap
MAILTRAP_PASS=votre_password_mailtrap
EMAIL_USER=from@example.com
```

Installation

```
# Cloner le dépôt
git clone https://github.com/WilliammCorreia/Application-
Authentification-S-curis-
cd [NOM_DU_PROJET]

# Installation des dépendances back-end
cd back-end
npm install

# Configuration de la base de données
npx prisma migrate dev --name init
npx prisma generate

# Démarrer le serveur
node server.js
```

```
# Accéder à la base de données
npx prisma studio
```

Déploiement du front-end

Le front-end est conçu pour être servi par un serveur web statique. En développement, vous pouvez utiliser l'extension Live Server de VSCode (port 5500).

Sécurité et bonnes pratiques

Mesures de sécurité implémentées

1. **Hachage des mots de passe** : Utilisation de bcrypt pour stocker les mots de passe de manière sécurisée
2. **Authentification à deux facteurs** : Vérification par code envoyé par email
3. **Cookies HTTP-only** : Protection des tokens JWT contre les attaques XSS
4. **Validation des entrées** : Vérification côté serveur avant toute opération sur la base de données
5. **Sécurité CORS** : Limitation des origines autorisées à communiquer avec l'API