
cruzzle

Sommaire

1. Présentation générale
2. Spécifications fonctionnelles
3. Design graphique
4. Démo
5. Spécifications techniques
6. Fonctions CRUD
7. Sécurisation et contrôles d'accès
8. Axes d'amélioration

Présentation générale

Contexte : créer une plateforme d'idéation

Equipe : 5 personnes

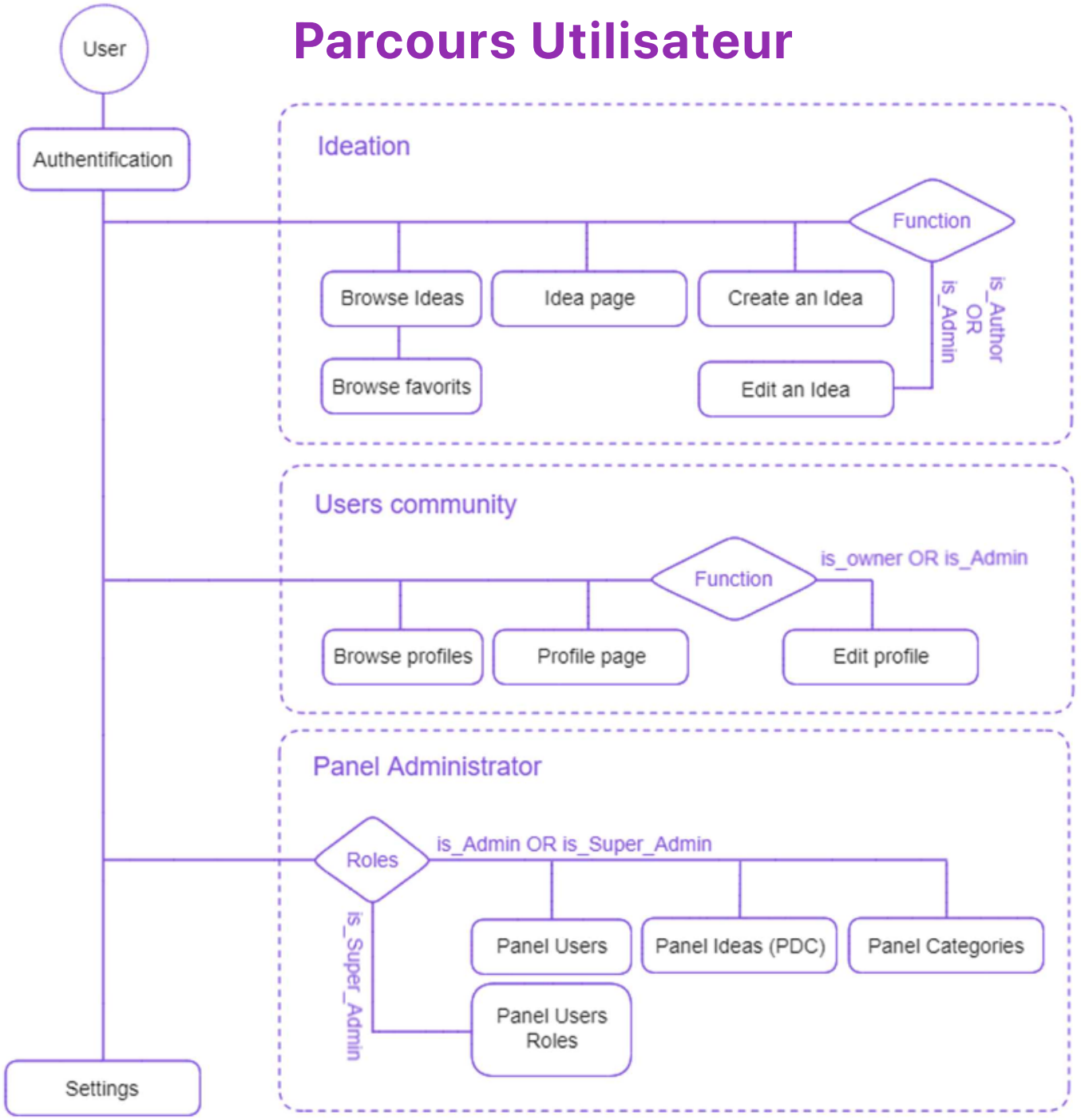
Calendrier : 10 semaines

Méthodologie : Agile SCRUM

Spécifications fonctionnelles

Sécurité	Idées	Utilisateurs	Divers
Login	Equipes	Profils	Dashboard
Admin	Commentaires	Activités	Notifications
	Likes	Gamification	Langues

Parcours Utilisateur



Design graphique

UI : Créativité, Convivialité, Professionnalisme

UX : Accessibilité, Simplicité, Intuitivité

Maquettage : Figma, mobile first, responsive

Démo 🔗

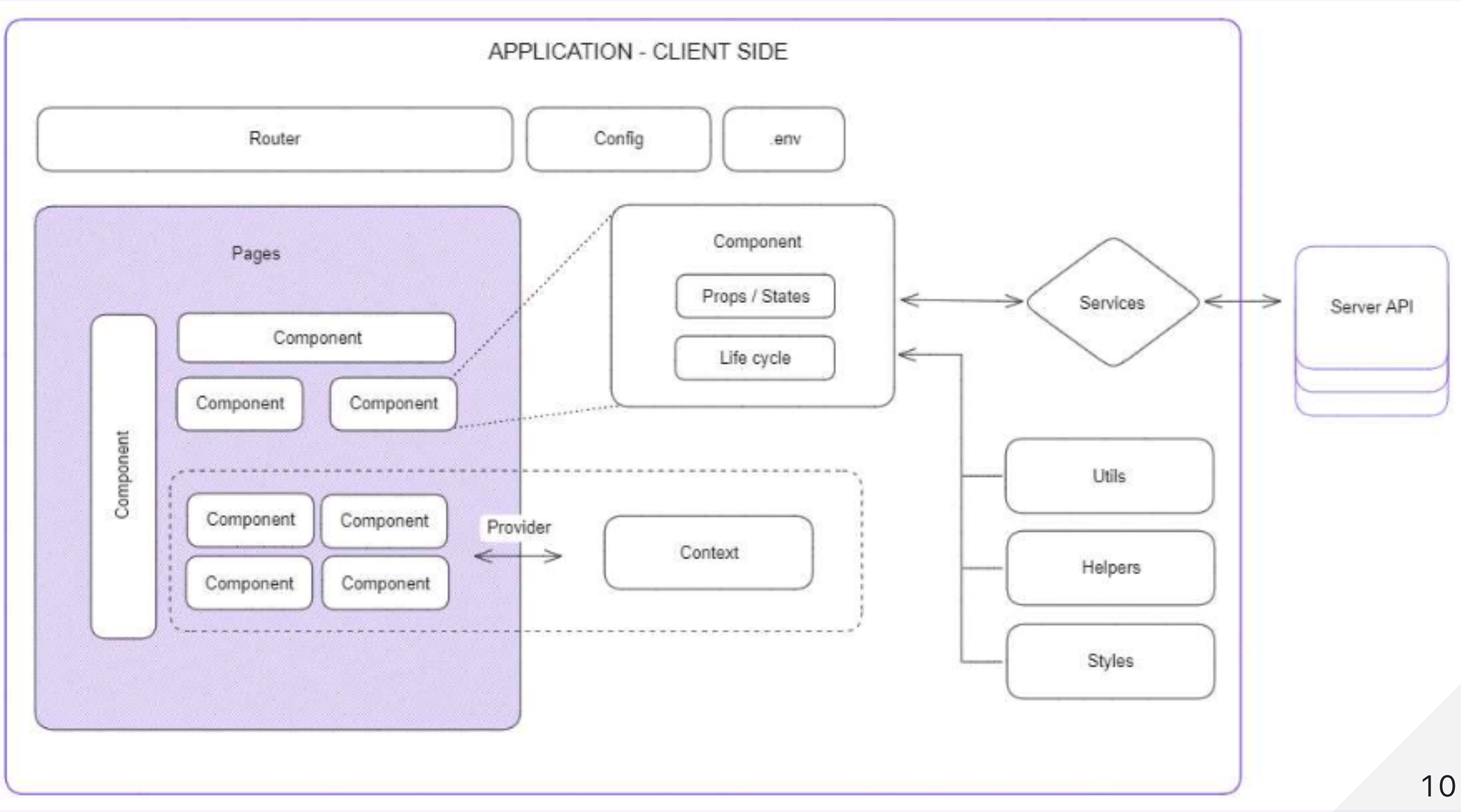
Spécifications techniques

FRONTEND | BDD & BACKEND

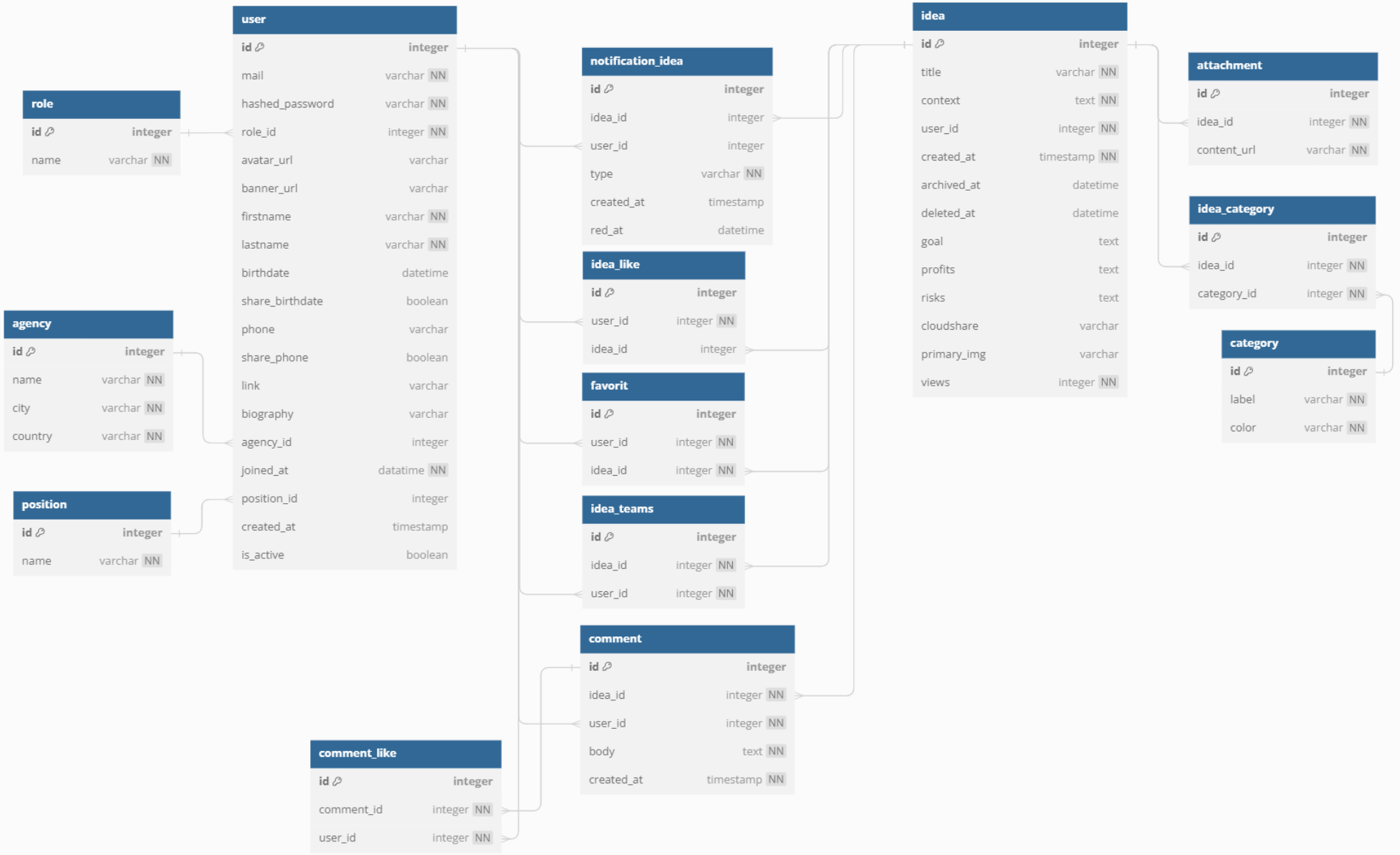
Frontend : stack technique

- React
- Material-UI
- Tailwind CSS
- React Router Dom
- Axios
- Dayjs
- PropTypes
- ESLint / Prettier

Frontend : architecture de l'application



Base de données : diagramme relationnel

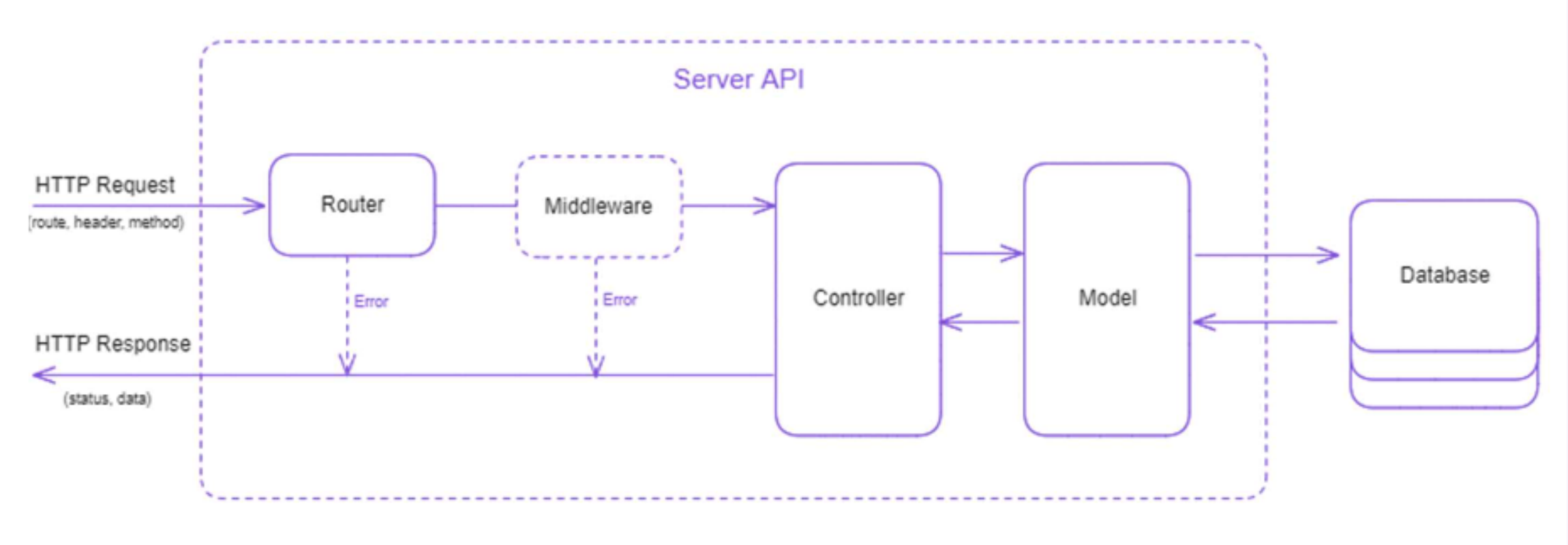


dbdiagram.io

Backend : stack technique

- TypeScript - Node.js
- Express
- CORS - Helmet
- MySQL2
- Prisma
- Axios
- Multer
- Bcrypt / JSON Web Token
- ESLint (Airbnb) / Prettier

Backend : architecture MVC



Fonctions CRUD

CREATE | READ | UPDATE | DELETE

- présentation d'extraits de code pour CREATE & READ-

```jsx
// code from frontend/src/services/api.admin.users.js
import Axios from "../config/axios.config";
const url = import.meta.env.VITE_BACKEND_URL;

export const apiAdminCreateUser = async (newUser) => {
  const route = "/api/admin/users/";
  const response = await Axios.post(`${url}${route}`, newUser);
  return response;
};

// code from frontend/src/components/admin/adminUsers/DialogCreateUser.jsx
import { apiAdminCreateUser } from "../../../services/api.admin.users";

const handleSubmit = async () => {
  apiAdminCreateUser(newUser)
    .then((res) => {
      if (res.status === 201) {
        setUpdateList(true);
        setMessage(
          t(
            "pages.adminpannel.users.tableOfUsers.dialogCreateUser.alert.success.message"
          )
        );
        setOpen(true);
        handleClose();
      } else {
        console.error("Cannot create new user");
      }
    })
    .catch((err) => {
      if (err.response.status === 409) {
        setEmailError(true);
        setEmailErrorMessage(
          t(
            "pages.adminpannel.users.tableOfUsers.dialogCreateUser.alert.error.message"
          )
        );
      } else {
        console.error("error creating new user", err);
      }
    });
};
```

CREATE (POST)

code côté **client** pour la création d'un utilisateur

```
// code from backend/src/routes/admin.routes.ts
router.post("/users", hashPassword, CreateUserByAdmin);

// code from backend/src/controllers/admin.users.controllers.ts
const CreateUserByAdmin = async (req: Request, res: Response) => {
  const newUser = req.body;
  try {
    const data = await createByAdmin(newUser);
    if (data.status === "success") {
      res.status(201).json(data.user);
    } else if (data.status === "conflict") {
      res.status(409).json(data.message);
    } else {
      res.sendStatus(400);
    }
  } catch (error) {
    res.status(500).send(error);
  }
};
```

CREATE (POST)

code **controller** côté **server** pour la création d'un utilisateur

```
// code from backend/src/models/admin.user.model.ts
const createByAdmin = async (newUser: CreateUser) => {
  try {
    const existingUser = await prisma.user.findUnique({
      where: {
        mail: newUser.mail,
      },
    });
    if (existingUser) {
      return { status: "conflict", message: "Email not available" };
    }

    const createdUser = await prisma.user.create({
      data: {/* new user data*/},
    });
    return {
      status: "success",
      message: "User created successfully.",
      user: createdUser,
    };
  } catch (error) {
    console.error("Error creating user:", error);
    throw new Error("Error creating user.");
  } finally {
    await prisma.$disconnect();
  }
};
```

CREATE (POST)

code **model** côté **server** pour la création d'un utilisateur

```jsx
// code from frontend/src/pages/admin/AdminUsers.jsx
//...
useEffect(() => {
    apiAdminUsers()
      .then((res) => {
        if (res.status === 200) {
          setUserlist(res.data);
        } else {
          navigate("/error", {
            state: {
              error: {
                status: res.status,
              },
            },
          });
          console.error("Cannot get users from panel admin");
        }
      })
      .finally(() => setUpdateList(false))
      .catch((error) => {
        navigate("/error", {
          state: {
            error: {
              status: 500,
            },
          },
        });
        console.error("Error getting users from panel admin", error);
      });
  }, [updateList]);
//...
```

READ (GET)

code côté **client** pour la récupération de la liste des utilisateurs

(choix de la vérité vs optimistic ui)

```ts
// code from backend/src/models/admin.user.model.ts
const findAllByAdmin = async () => {
  try {
    const data = await prisma.user.findMany({
      select: {
        id: true,
        mail: true,
        role: {
          select: {
            id: true,
            name: true,
          },
        },
        // ...
        is_active: true,
        position: {
          select: {
            id: true,
            name: true,
          },
        },
      },
    });
    return data;
  } finally {
    await prisma.$disconnect();
  }
};
```

READ (GET)

code **model** côté **server** pour la création d'un utilisateur

```sql
SELECT
  u.id,
  u.mail,
  u.firstname,
  u.lastname,
  u.joined_at,
  u.created_at,
  u.is_active,
  r.id AS role_id,
  r.name AS role_name,
  a.id AS agency_id,
  a.name AS agency_name,
  a.city AS agency_city,
  a.country AS agency_country,
  p.id AS position_id,
  p.name AS position_name
FROM
  users AS u
  JOIN roles AS r ON u.role_id = r.id
  JOIN agencies AS a ON u.agency_id = a.id
  JOIN positions AS p ON u.position_id = p.id;
```

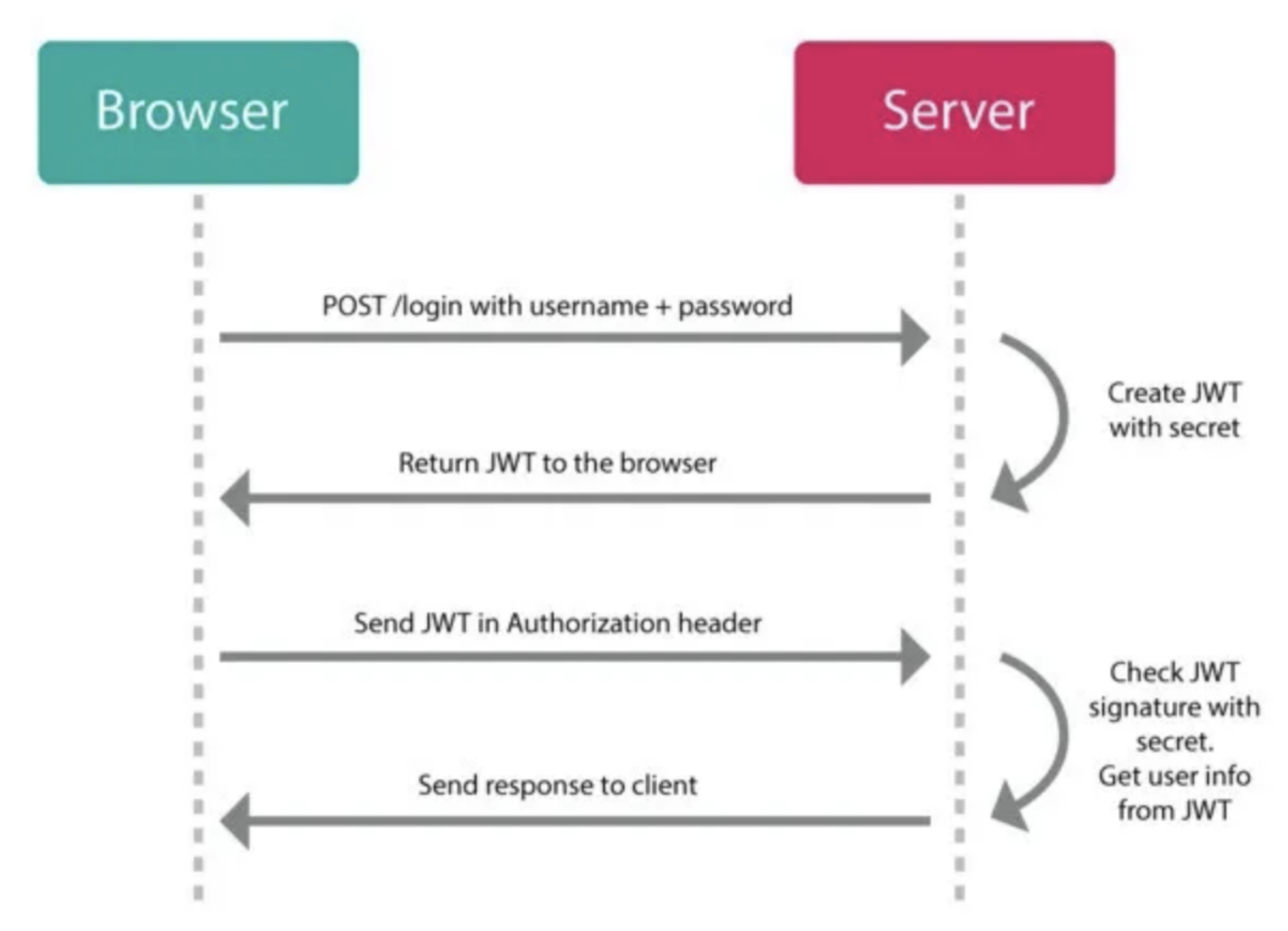
READ (GET)

requête **SQL** équivalente pour la récupération de la liste des utilisateurs

Sécurisation et controlles d'accès

Authentification Vs Authorization

Echanges client / serveur



Axes d'amélioration

- Refactorisation et Typage du code
- Optimisation des requêtes
- Intégration de tests unitaires et fonctionnels
- Gestion des jetons d’authentification

Merci pour votre attention !