PRICE MAKER

Par Sandrine Da Silva

Presentation

Sandrine Da Silva

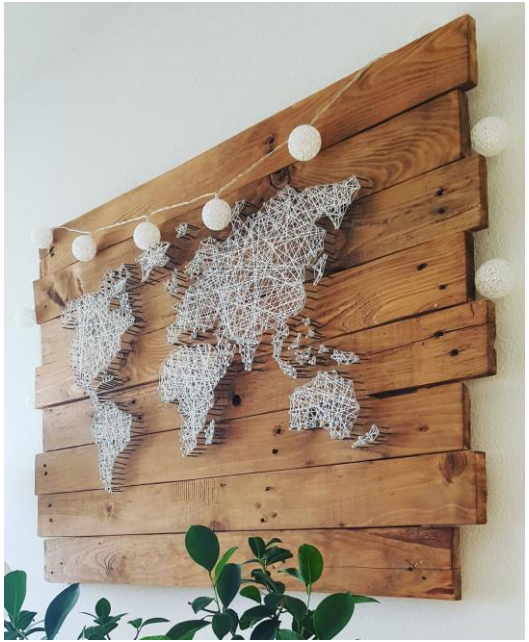
31 years old

STG degree (management sciences & technology)

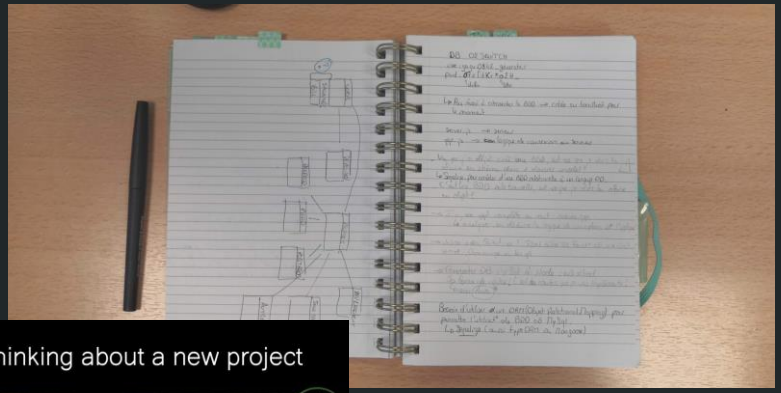
Professional experience :

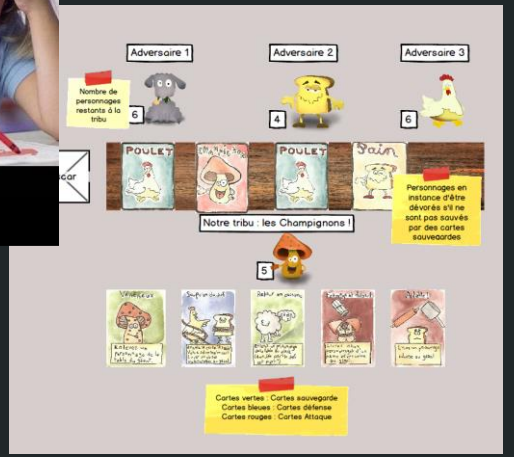
Secretariat - sales - microenterprises

My passions



Programming





What’s next ?

{EPITECH.}

fullsave

Stage : Fullsave

LE PROJET

Price Maker,
générateur de tarif
pour vidéaste freelance

Au programme :

- Présentation du projet
- Spécifications fonctionnelles
- Spécifications techniques
- Implémentation
- Bilan et conclusion

Comment fixer son prix ?

VIDÉASTE FREELANCE

Estimation

- Salaire estimé
- Impôts, taxes et autres charges sociales
- Dépenses mensuelles
- Charge de travail réalisable

Projet

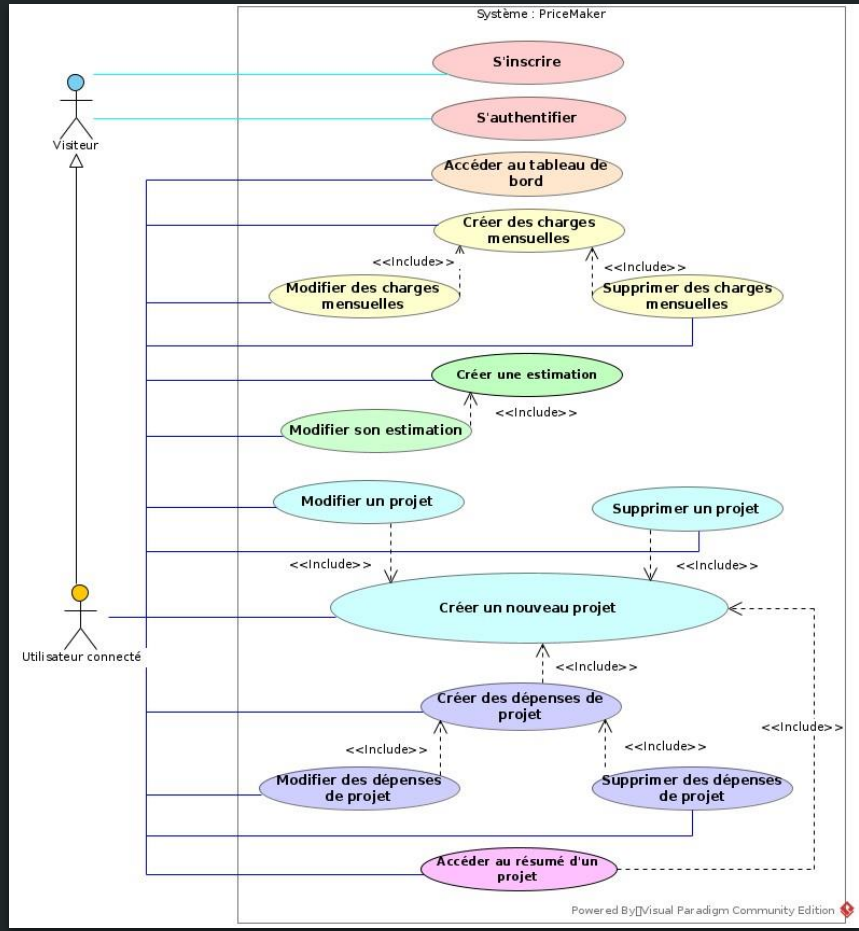
- Dépenses liées à ce projet
- Charge de travail propre à ce projet

Les cas d'utilisation

4 activités principales :

- Créer une estimation
- Créer des charges mensuelles
- Créer un projet
- Créer des dépenses de projet

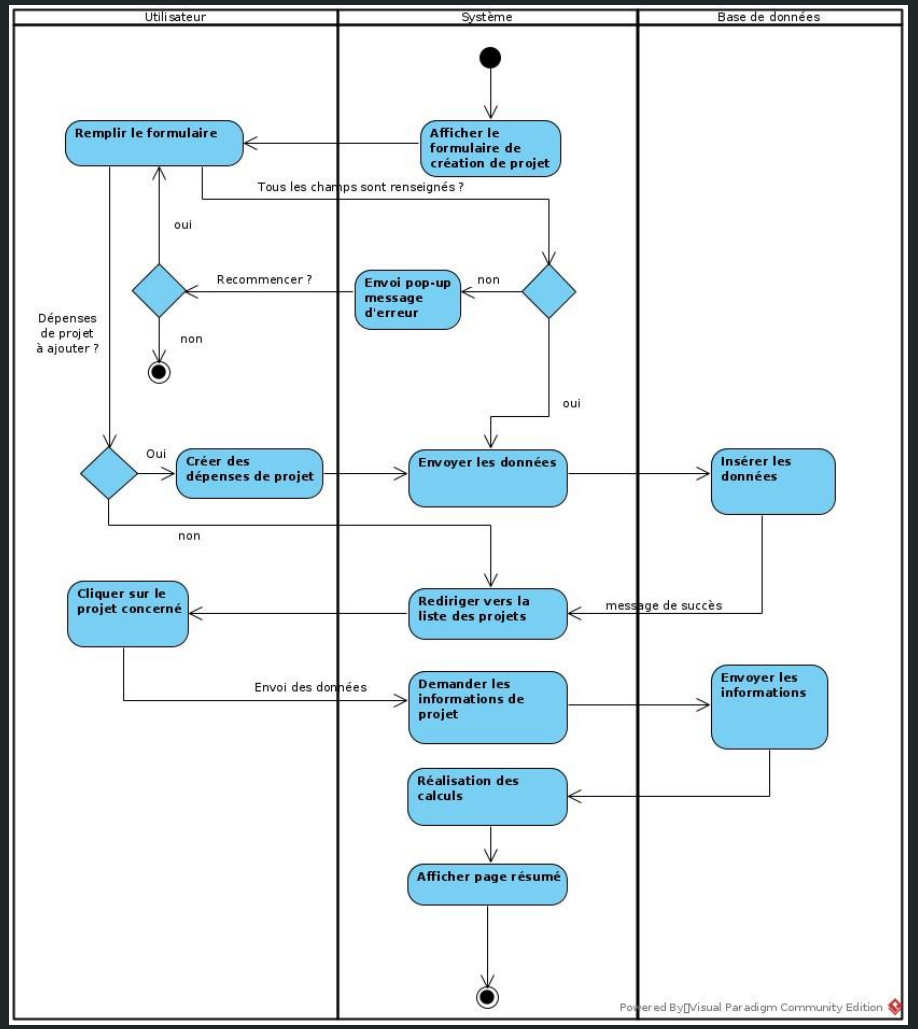
Impliquent d'être authentifié



Le diagramme d’activité

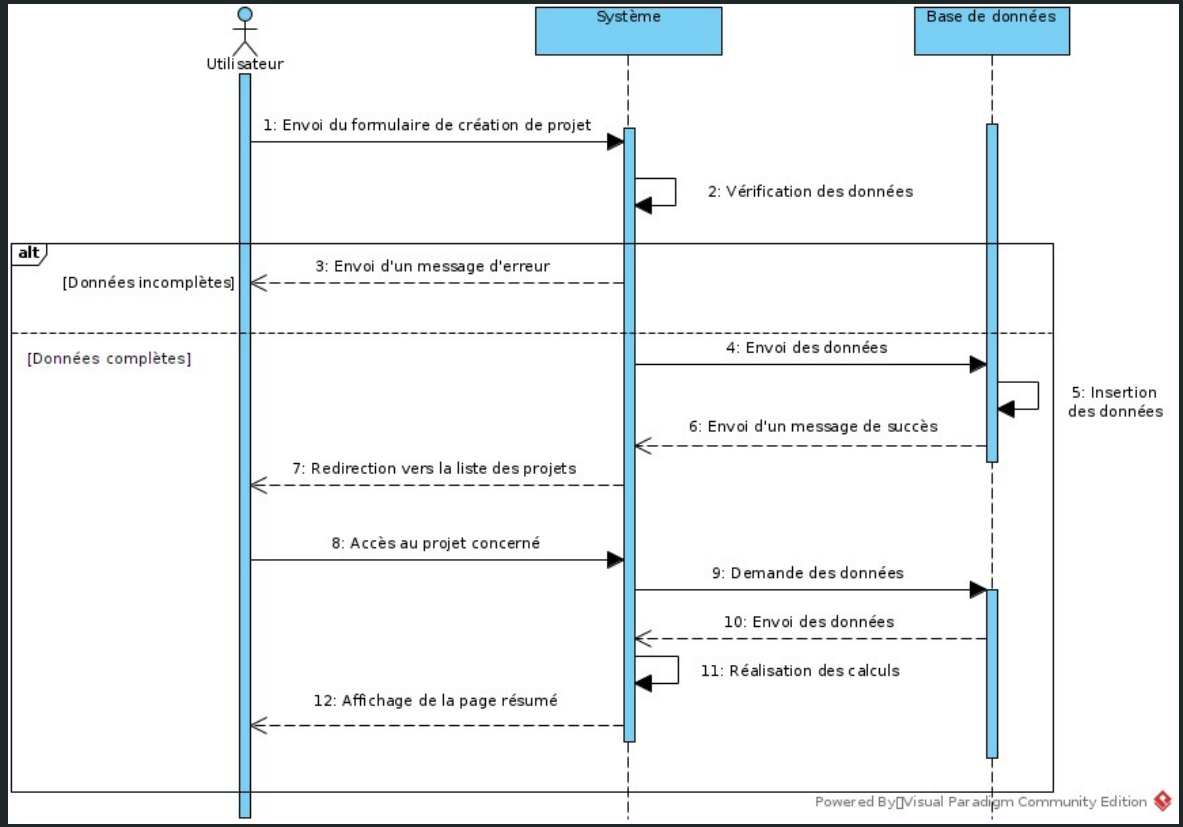
Cas d’utilisation observé :
Affichage du résumé de projet

3 noeuds de décision

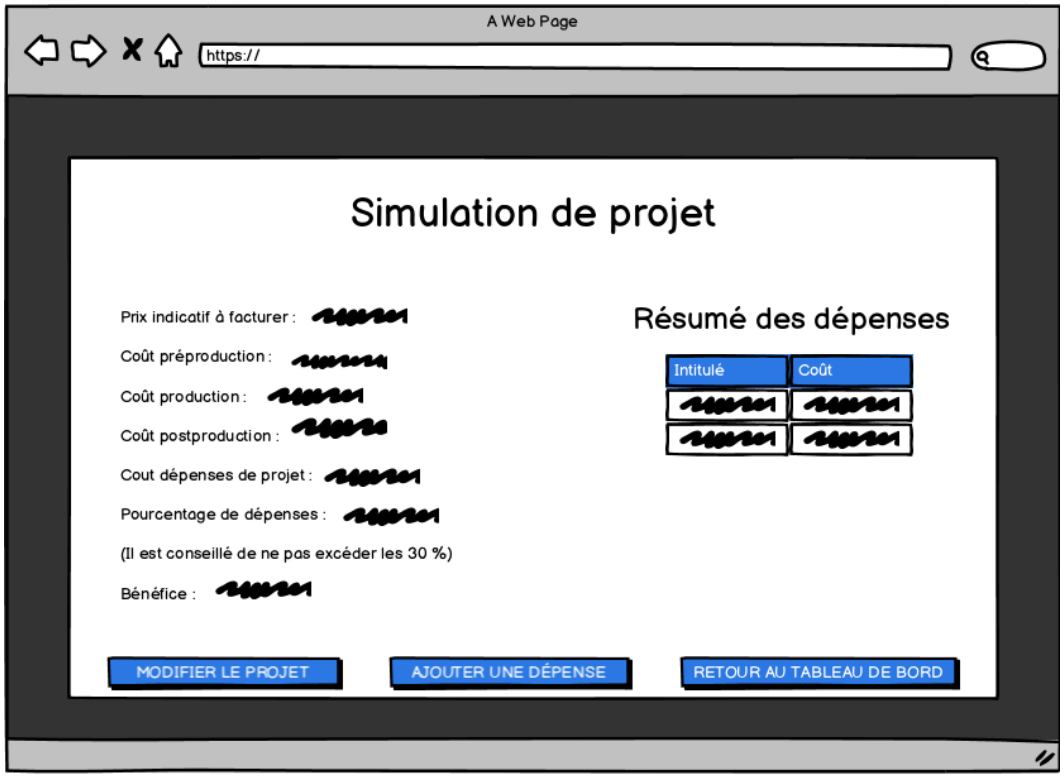


Le diagramme de séquence

Cas d'utilisation observé :
Affichage du résumé de projet



Les maquettes



Le dictionnaire de données

2 types de données :

- **Insérées par l'utilisateurs**
- Données calculées

Insérées dans la base de données

5 entités : mes futures tables

CODE	DÉSIGNATION	TYPE	TAILLE
user_lastname	Nom utilisateur	Varchar	255
user_firstname	Prénom utilisateur	Varchar	255
user_mail	Mail utilisateur	Varchar	255
user_password	Mot de passe utilisateur	Varchar	255
monthly_charge_name	Intitulé charge mensuelle	Varchar	255
monthly_charge_cost	Coût charge mensuelle	Integer	
wage	Salaire visé	Integer	
rate	Taux de cotisation sociale à appliquer	Integer	
number_project_by_month	Nombre de projet estimés par mois	Integer	
number_hour_shooting	Nombre d'heure de tournage estimé par projet	Integer	
number_hour_postprod	Nombre d'heure de postproduction estimé par projet	Integer	
number_hour_preprod	Nombre d'heure de préproduction estimé par projet	Integer	
price_day_preprod	Prix de la journée de préproduction	Integer	
project_name	Nom de projet	Varchar	255
number_day_preprod	Nombre de jour pour la préproduction	Integer	
number_day_prod	Nombre de jour pour la production	Integer	
number_hour_postprod	Nombre d'heures pour la postproduction	Integer	
project_expense_type	Type de dépense de projet	Varchar	255
project_expense_cost	Coût de la dépense de projet	Integer	

Les données calculées

2 types de données :

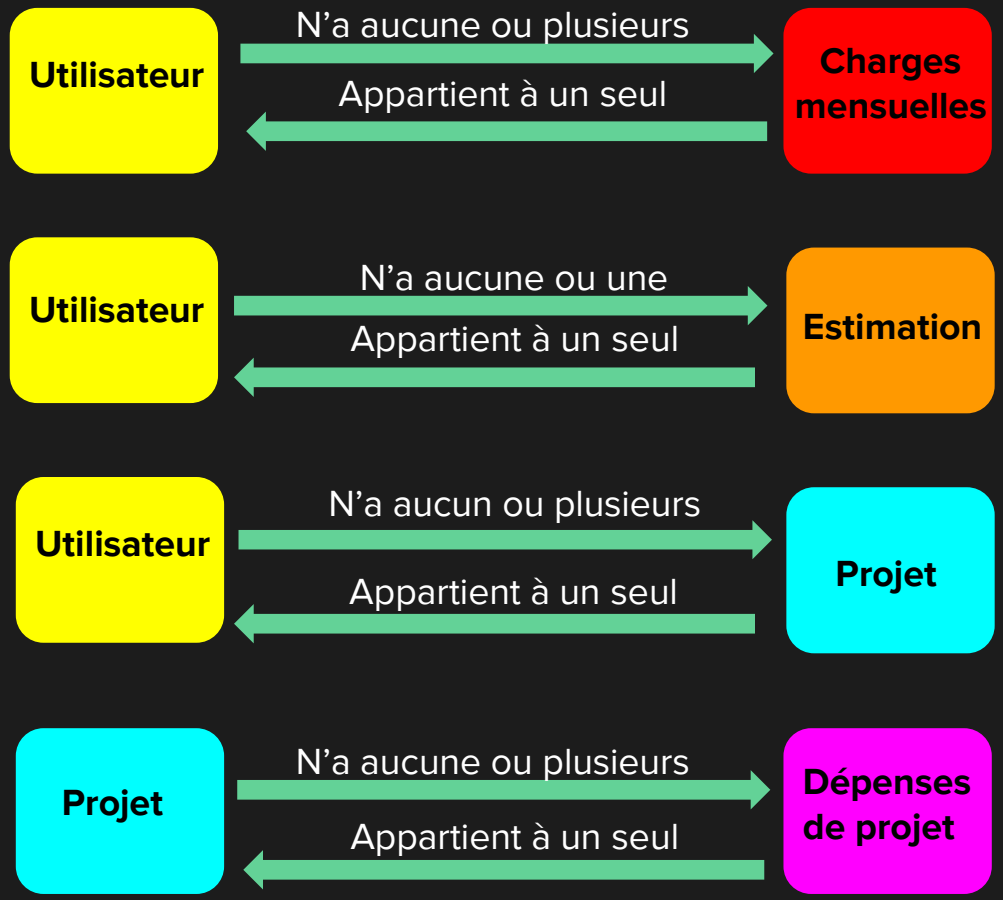
- Insérées par l'utilisateurs
- **Données calculées**

Seront implémentées dans l'API

Facturation moyenne	averageInvoice
Nombre de projets nécessaires par an	grossSalesByYear
Total dépenses mensuelles	totalMonthlyCharge
Frais de production	totalProjectExpense
Pourcentage de dépense	expenseRate
Bénéfice	benefit
Chiffre d'affaires à réaliser avant paiement des cotisations	preTaxes
Chiffre d'affaires à réaliser après paiement des cotisations	postTaxes
Revenu annuel net	revenueByYear
Tarif horaire	hourlyFee
Taux journalier	dailyRate
Nombre d'heures travaillées par mois	hourWorkedByMonth
Total des frais de production pour un projet	totalProd
Total des frais de préproduction pour un projet	totalPreprod
Total des frais de postproduction pour un projet	totalPostprod
Total dépenses de projet	totalProjectExpense
Facture client (dépenses de projet incluses)	invoice

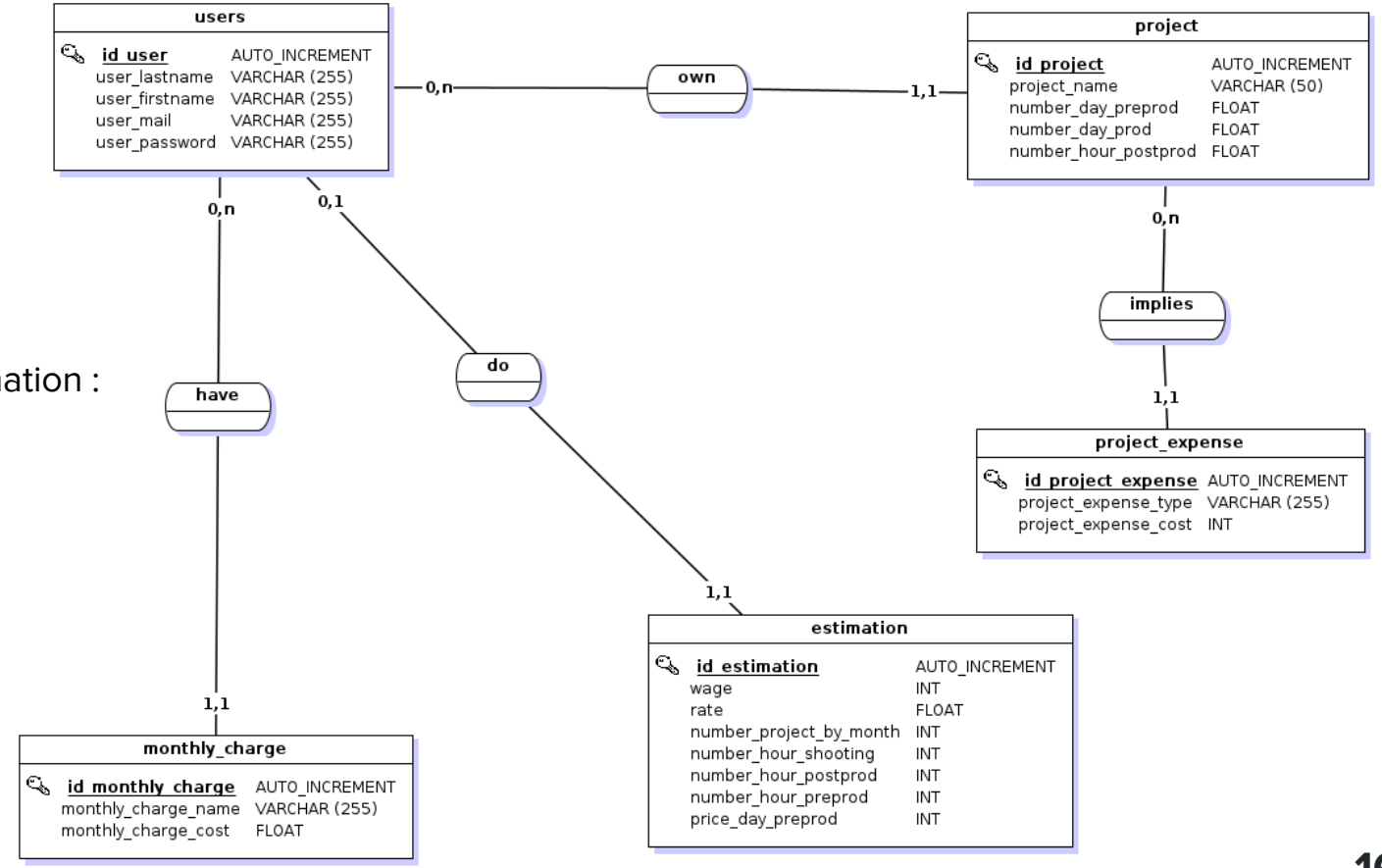
Les règles de gestion

Objectif :
Définir les relations entre les entités



Le MCD

Cardinalité users - estimation :
0.1 - 1.1



Choix des technologies

Back-end :

- MySQL
- Node
- Express JS

Front-end :

- React
- Axios
- Material UI

Architecture de l’application

Style d’architecture employé :
Architecture 3 tiers
selon le modèle de conception MVC

Architecture 3-tiers

Vue

Contrôleur

Modèle

Couche de présentation :

- Front-end : React
- Routes de l’API

Couche métier :

- Requêtes à la base de données
- calculs

Couche d’accès aux données :

- Base de données

Implémentation.

Les requêtes SQL

Contraintes :

- Clé primaire
- Clé étrangère
- Contrainte Unique

Insertion d'un jeu de données :

Tester la base de données

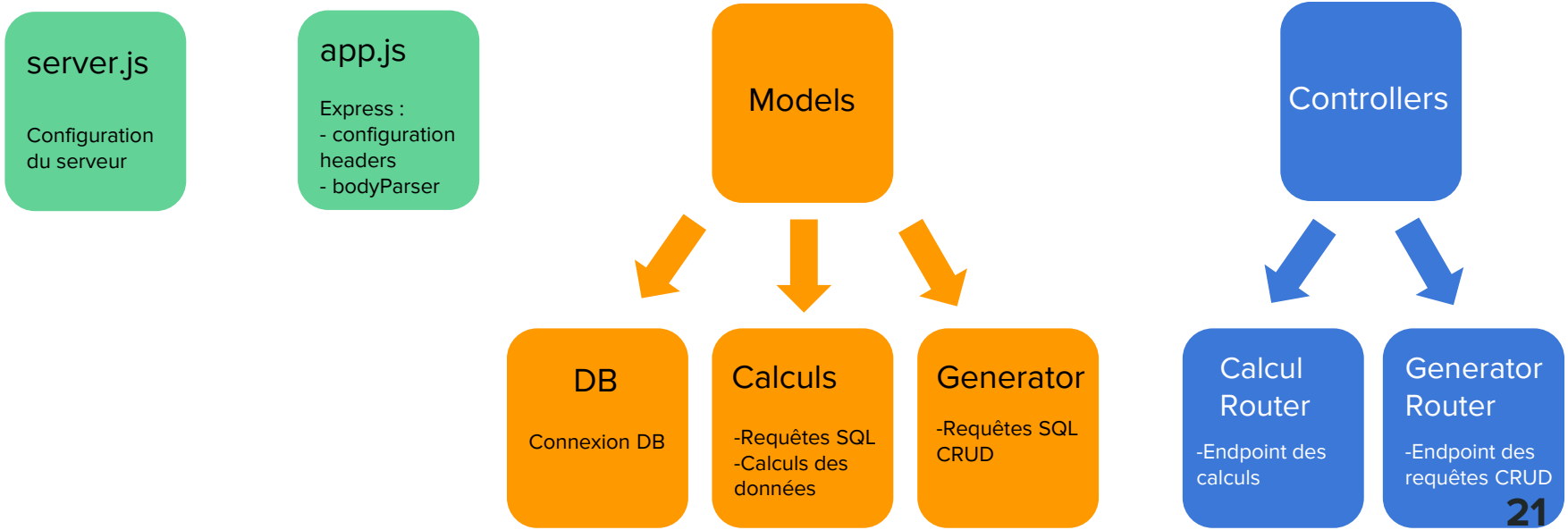
```sql
CREATE TABLE estimation(
        id_estimation           Int  Auto_increment  NOT NULL ,
        wage                    Int NOT NULL ,
        rate                    Float NOT NULL ,
        number_project_by_month Int NOT NULL ,
        number_hour_shooting    Int NOT NULL ,
        number_hour_postprod    Int NOT NULL ,
        number_hour_preprod     Int NOT NULL ,
        price_day_preprod       Int NOT NULL ,
        id_user                 Int NOT NULL
    ,CONSTRAINT estimation_PK PRIMARY KEY (id_estimation)

    ,CONSTRAINT estimation_users_FK FOREIGN KEY (id_user) REFERENCES users(id_user)
    ,CONSTRAINT estimation_users_AK UNIQUE (id_user)
)ENGINE=InnoDB;
```

```sql
INSERT INTO users (user_lastname, user_firstname, user_mail, user_password)
VALUES
('Da Silva', 'Sandrine', 'dasilva.sandrine31@gmail.com', 'password'),
('Gardes', 'Guillaume', 'gardes.g@gmail.com', 'password'),
('Gardes', 'Estelle', 'estelle.gardes@gmail.com', 'password'),
('Da Silva', 'Joey', 'joey.ds@gmail.com', 'password'),
('Delcol', 'Anaïs', 'delcol.anais@gmail.com', 'password');

INSERT INTO monthly_charge (monthly_charge_name, monthly_charge_cost, id_user)
VALUES
('abonnement suite Adobe', 49.50, 1),
('assurance professionnelle', 20, 1),
('crédit achat caméra', 100, 1),
('abonnement suite Adobe', 49.50, 2),
('assurance professionnelle', 20, 2),
('crédit achat caméra', 100, 2);
```

Côté back-end
Schéma API



Configuration des endpoints

- Méthodes asynchrones
- Try Catch
- Mise en place du token d'identification
- Hash du mot de passe

```
//Connexion
router.post("/login", async (req, res, next) => {
  try {
    let results = await login.logUser(req.body.mail, req.body.password);
    const token = await jwt.sign({ results }, [redacted]);
      let message = await res.json({
        token: token,
        user_id: results
      });
      return message
  } catch (e) {
    console.log(e);
    res.sendStatus(500).json({ message: "utilisateur non trouvé" });
  }
});
```

```
generator.logUser = async (mail, password) => {
  const user = await query("SELECT * FROM users WHERE user_mail = ?", [mail]);
  await bcrypt.compare(password, user[0].user_password, (err, res) => {
    if (err) {
      console.log("Password doesn't match !");
      return {
        message: "Le mot de passe ou le mail n'est pas valide",
      };
    }
    if (res) {
      console.log("Password matches!");
      return user[0].id_user;
    } else {

      return {
        message: "Un problème est survenu"
      };
    }
  });
};
```

Méthode CRUD

Create

Verbe POST

Vérification du token

SQL : INSERT INTO ... VALUES

```
//nouvelle charge
router.post("/new-monthly-charge", ensureToken, async (req, res, next) => {
  const idUser = jwt.verify(req.token, [redacted], function (err, data) {
    if (err) {
      res.sendStatus(403);
    } else {
      return data.results;
    }
  });
  try {
    let results = await insertCharge.insertMonthlyCharge(
      req.body.nameValue,
      req.body.costValue,
      idUser
    );
    res.json(results);
  } catch (e) {
    console.log(e);
    res.sendStatus(500);
  }
});
```

```
generator.insertMonthlyCharge = (nameValue, costValue, id) => {
  return query(
    "INSERT INTO monthly_charge(monthly_charge_name, monthly_charge_cost, fk_id_user) VALUES (?,?,?);",
    [[nameValue], [costValue], [id]]
  );
};
```

Méthode CRUD

Read

Verbe GET

Vérification du token

SQL : SELECT * ... WHERE

```
router.get("/dashboard/get-charge", ensureToken, async (req, res, next) => {
  const idUser = jwt.verify(req.token, [redacted], function (err, data) {
    if (err) {
      res.sendStatus(403);
    } else {
      return data.results;
    }
  });
  try {
    let results = await getCharge.getMonthlyCharge(idUser);
    res.json(results);
  } catch (e) {
    console.log(e);
    res.sendStatus(500);
  }
});
```

```
generator.getMonthlyCharge = (id) => {
    return query(
        "SELECT * FROM monthly_charge WHERE fk_id_user = ?;",
        [id],
    );
};
```

Méthode CRUD

Update

Verbe PATCH

Vérification du token

SQL : UPDATE ...
SET ... WHERE

```
// modifier charge
router.patch(
  "/update-monthly-charge/:idMonthlyCharge",
  ensureToken,
  async (req, res, next) => {
    const idUser = jwt.verify(req.token, [redacted], function (err, data) {
      if (err) {
        res.sendStatus(403);
      } else {
        return data.results;
      }
    });
    try {
      let results = await updateCharge.updateMonthlyCharge(
        req.body.nameValue,
        req.body.costValue,
        req.params.idMonthlyCharge
      );
      res.json(results);
    } catch (e) {
      console.log(e);
      res.sendStatus(500);
    }
  }
);
```

```
generator.updateMonthlyCharge = (nameValue, costValue, id) => {
  return query(
    "UPDATE monthly_charge SET monthly_charge_name = ?, monthly_charge_cost = ? WHERE id_monthly_charge = ?;",
    [[nameValue], [costValue], [id]]
  );
};
```

Méthode CRUD

Delete

Verbe DELETE

Vérification du token

SQL : DELETE FROM ... WHERE

```
//supprimer charge
router.delete(
  "/delete-monthly-charge/:idMonthlyCharge",
  ensureToken,
  async (req, res, next) => {
    const idUser = jwt.verify(req.token, [redacted], function (err, data) {
      if (err) {
        res.sendStatus(403);
      } else {
        return data.results;
      }
    });
    try {
      let results = await deleteCharge.deleteMonthlyCharge(
        req.params.idMonthlyCharge
      );
      res.json(results);
    } catch (e) {
      console.log(e);
      res.sendStatus(500);
    }
  }
);
```

```
generator.deleteMonthlyCharge = (id) => {
  return query("DELETE FROM monthly_charge WHERE id_monthly_charge = ?;", [id]);
};
```

Méthode de calcul

- Boucle for of permettant d'additionner des valeurs
- Méthode Math.round retournant un arrondi à l'entier le plus proche
- Méthode Math.ceil retournant le plus petit entier supérieur ou égal

```
ca //récupération de l'estimation du nombre d'heures de tournage:
  let shooting = await query(
    "SELECT number_hour_shooting FROM estimation WHERE fk_id_user = ?",
    [id]
  );
  //priceDayPreprod :
  let priceDayPreprod = await query(
    "SELECT price_day_preprod FROM estimation WHERE fk_id_user = ?",
    [id]
  );
  priceDayPreprod = priceDayPreprod[0].price_day_preprod;
  dayPreprod = dayPreprod * priceDayPreprod;

  //number_hour_postprod
  let hourPostprod = await query(
    "SELECT number_hour_postprod FROM project WHERE id_project = ?",
    [idProject]
  );
  hourPostprod = hourPostprod[0].number_hour_postprod;
  hourPostprod = Math.ceil(hourPostprod * hourlyFee);

  //totalProjectExpense
  let totalExpense = await query(
    "SELECT project_expense_cost FROM project_expense WHERE fk_id_project = ?",
    [idProject]
  );
  let sumExpense = 0;
  for (let attr of totalExpense) {
    let result = attr.project_expense_cost;
    sumExpense += result;
  }
  totalExpense = sumExpense;
  return dayProd + dayPreprod + hourPostprod + totalExpense;
};

  );
  dayPreprod = dayPreprod[0].number_day_preprod;
```

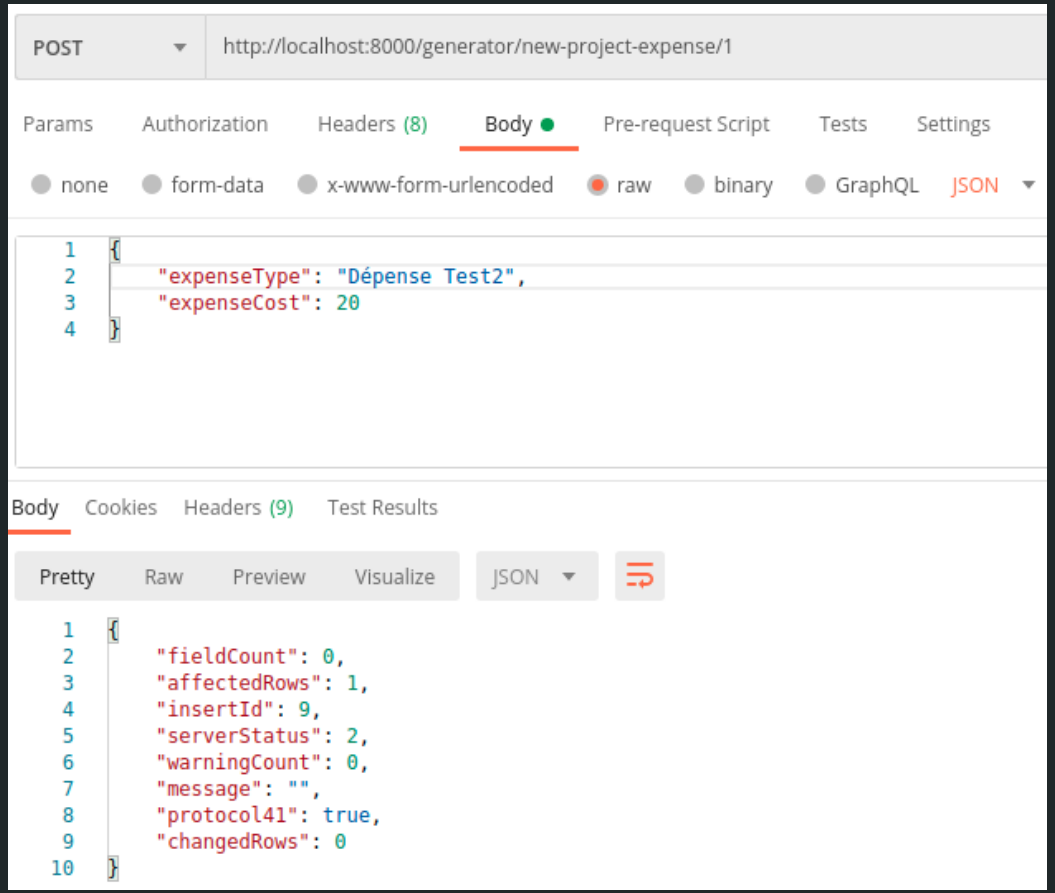
Test d'API

Postman

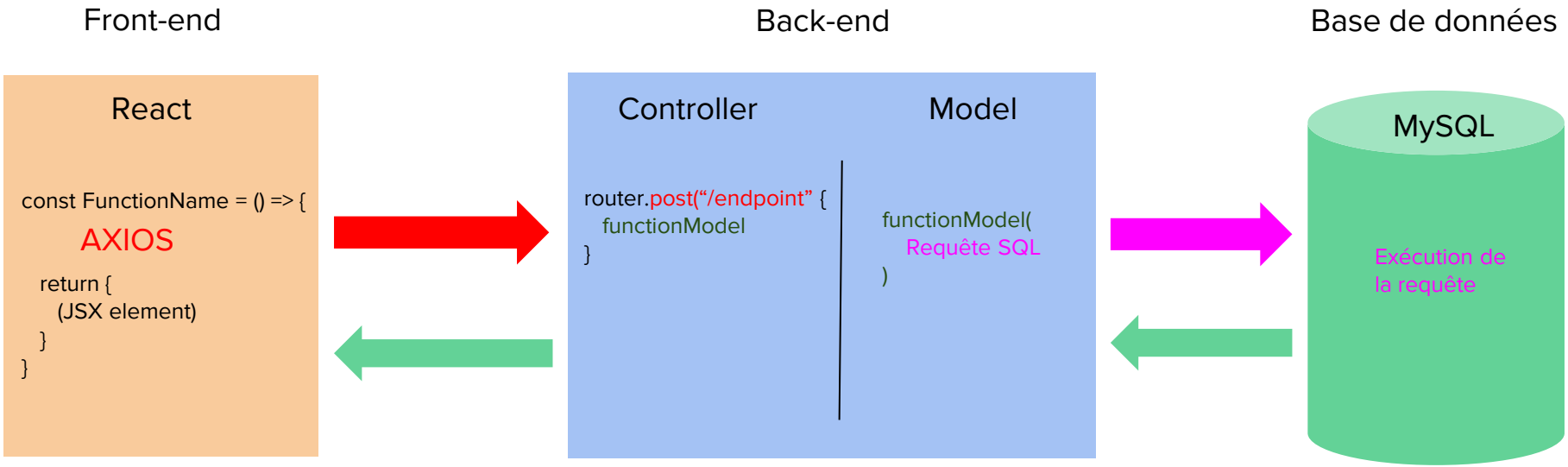
Création de bibliothèques

Envoi des données en JSON

Configuration des headers



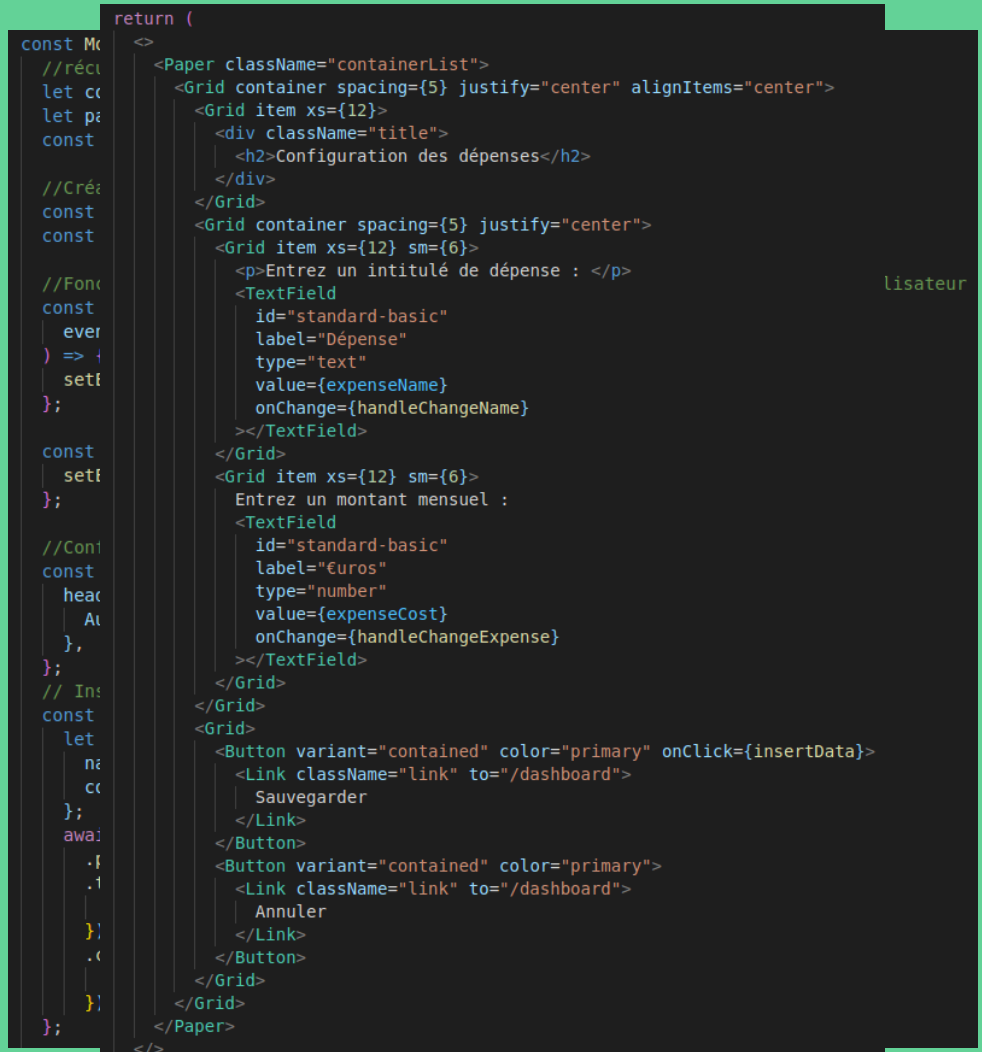
Communication entre les acteurs



Zoom sur une fonctionnalité

- Composant de fonction
- hook useState
- Typescript
- Récupération du cookie contenant le token
- Configuration du header de la requête HTTP
- Ciblage de la requête par l'endpoint

- Material UI
- React router dom



Démonstration

Points d’amélioration

- Ajout des **fonctionnalités** :
  - Déconnexion
  - Modification des données de l’utilisateur
  - Possibilité de faire plusieurs estimations
- Récupération d’id depuis le **cookie**
- **Refactorisation** de code
- **Réorganisation** des composants
- Revoir typage des variables sous **Typescript**
- Réalisation de **tests** unitaires, fonctionnels et d’intégration

Ce que j’ai appris

- Recherches Google
- Raisonner de manière logique
- Débugger mon code
- Apprentissage de React et Node

Conclusion