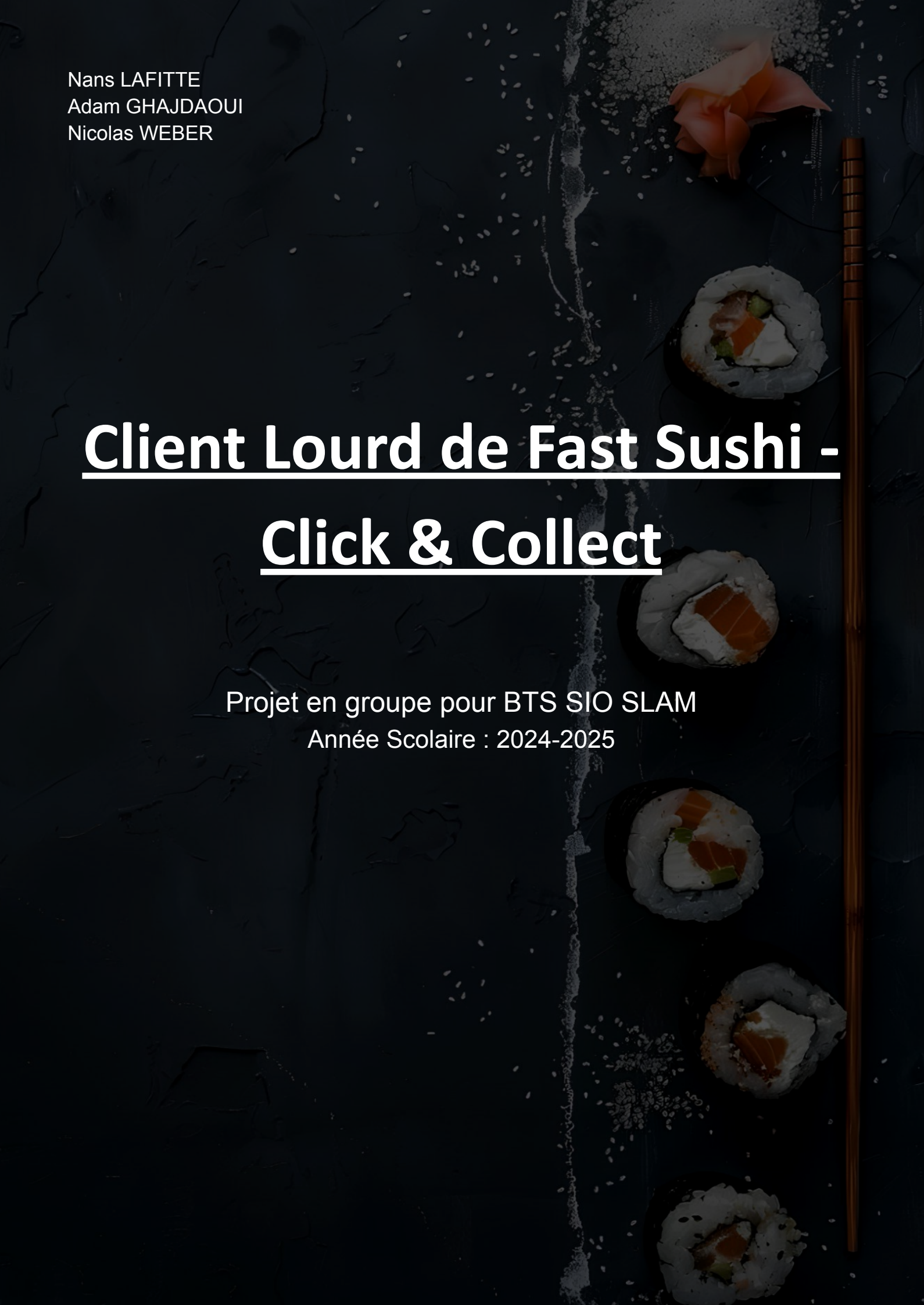


Nans LAFITTE  
Adam GHAJDAOUI  
Nicolas WEBER

# Client Lourd de Fast Sushi - Click & Collect

Projet en groupe pour BTS SIO SLAM  
Année Scolaire : 2024-2025



# **Sommaire**

Cahier des charges FastSushi.....	2
Spécification.....	4
Client Lourd.....	4
1.Objectif du client Lourd.....	4
2.L'authentification.....	4
3.Interface du logiciel.....	4
Interfaces.....	5
Produits.....	5
Commandes.....	5
Détails.....	6
Historique.....	6
Administration.....	7
Partie Technique.....	7
Planification & Organisation.....	7
Base de données.....	7
Environnement de travail local Docker.....	8
Client Lourd - JavaFX.....	8
Annexes.....	9
A1. Use Case - Client Lourd.....	9
A2. Planning - Client Lourd.....	10
A3. Modélisation BDD (MCD + MLD).....	11
A4. Arborescence.....	12
A5. HistoryCommandesView.fxml.....	13
A6. HistoryCommandesController.java.....	14
A7. CommandesService.java.....	16
A8. Commandes.java.....	21
A9. Produit.java.....	22

# Cahier des charges FastSushi

L'entreprise FastSushi est spécialisée dans la restauration rapide, et principalement dans les produits alimentaires d'inspiration japonaise tels que les sushis, les makis, les sashimis, les springs rolls, ... La société qui a vu le jour en 2008, et se spécialise pour faire découvrir à sa clientèle la vraie cuisine japonaise à base de produits locaux et de qualités. Le restaurant garantit que tous ses poissons sont traités entiers et sélectionnés suivant les arrivages pour conserver un maximum de fraîcheur. Les produits sont certifiés AB et Label rouge. Le gérant de l'entreprise est M. Makytori

## **Expressions des besoins :**

La société FastSushi veut réaliser un site pour faire du click & collect auprès de sa clientèle. De plus son offre permet la possibilité de composer soit même les makis, california rolls et springs rolls (à partir des listes d'ingrédients fournies), ces recettes spéciales sont appelées recettes composables, elles peuvent être constituées de 2, 3 ou 4 ingrédients. Afin de simplifier la réalisation et le traitement, il vous ai demandé de réaliser deux applications qui permettent la création et le suivi des recettes, ventes, et collecte des produits. L'application backoffice sera utilisée par les administrateurs et ses préparateurs, tandis que le frontoffice sera utilisé par les clients pour réaliser leur panier et passer leur commande de produits. Vous réaliserez pour répondre à ce besoin deux applications (client lourd , client léger) :

### **Application FastSushi FrontOffice - Client léger :**

- Les utilisateurs peuvent créer leurs comptes eux-mêmes sur le site avant de commander (nom, téléphone, adresse email, mot de passe).
- L'utilisateur peut naviguer grâce à un menu vers les différentes catégories.
- Lorsqu'un utilisateur choisit une recette composable, une fenêtre s'ouvre pour lui permettre de choisir la composition.
- Chaque fois qu'un produit est ajouté (via un bouton acheté), il est ajouté dans le panier de l'utilisateur qui recense les différents composants de sa commande.
- L'utilisateur peut consulter son panier via un bouton en forme de panier en haut de la page, cet écran recense donc le contenu du panier avec le prix total à payer en bas de la page.
- Sur l'écran du panier, un bouton permet de valider la commande. C'est l'appui sur ce bouton qui passe la commande en statut à préparer par FastSushi.
- Il n'est pas nécessaire de gérer les moyens de paiement, ceux-ci se faisant au retrait en boutique.

### **Application FastSushi BackOffice - Client Lourd :**

- Les administrateurs/préparateur disposent d'un identifiant et d'un mot de passe pour se connecter à l'application qui sera installée sur son poste de travail et communique avec une base de données distantes (le plus sécurisé possible).
- Les administrateurs/préparateur s'occupent d'administrer la disponibilité des produits.
- Les préparateurs disposent d'un écran facilement accessible qui leur indique les commandes en cours qu'il faut préparer
- A partir de l'écran précédent, ils peuvent valider la remise de la commande au client quand celui-ci l'a récupéré en boutique.
- Les administrateurs ont la possibilité d'accéder aux historiques des commandes.

# Spécification

## Client Lourd

---

### 1.Objectif du client Lourd

Le **client lourd** est un logiciel utilisé par les employés de **FastSushi**.

Il permet la gestion de la disponibilité des produits et inclut également un système de gestion des commandes et des utilisateurs.

---

### 2.L'authentification

Les **employés** (cuisiniers et administrateurs) ont leurs propres comptes utilisateurs.

Les cuisiniers mettent à jour l'état des commandes en fonction de leur avancée et peuvent gérer la disponibilité des produits en cas de pénurie. Quant aux administrateurs, ils ont accès à toutes les fonctionnalités, y compris les deux interfaces **commandes** et **produits**, ainsi que l'interface **Administrateur**, qui permet la gestion des comptes.

---


### 3.Interface du logiciel

Une fois connecté, on a la possibilité d'avoir 3 menu:


1. Produits : Accessible aux **administrateurs** et aux **cuisiniers**, ce menu présente un tableau avec le **nom**, le **statut** et la **disponibilité**, contenant une checkbox pour chaque produit permettant de mettre à jour la disponibilité des plats sur le site. (  [Interface Produits](#) )
2. Administration : Réservé exclusivement aux **administrateurs**, ce menu permet la gestion des comptes utilisateurs du **client lourd**. (  [Administration](#) ).
3. Commandes : Accessible aux **administrateurs** et aux **cuisiniers**, ce menu permet d'indiquer le statut des commandes. (  [à préparer](#),  [en attente de réception](#), etc.). Mais également de consulter l'historique de toutes les commandes passées (  [Historique](#) ) ou encore le détail d'une commande.(  [Détails](#) )

# Interfaces


## Produits

<div><b>Admin</b></div> <div>Produits</div> <div>Commandes</div> <div>Administration</div>	Produits Fixes		
	Nom	Statut	Disponibilité
	Sushi Assortiment	non	<input type="checkbox"/>
	Maki Saumon	oui	<input checked="" type="checkbox"/>
	California Roll Thon	oui	<input checked="" type="checkbox"/>
	Ramen Classique	oui	<input checked="" type="checkbox"/>
	Ramen au Poulet	oui	<input checked="" type="checkbox"/>
	Ramen Vegan	oui	<input checked="" type="checkbox"/>
	Yakitori Poulet	oui	<input checked="" type="checkbox"/>
	Yakitori Saumon	oui	<input checked="" type="checkbox"/>
	Yakitori Legumes	oui	<input checked="" type="checkbox"/>
	Bobun Classique	oui	<input checked="" type="checkbox"/>
	Bobun au Poulet	oui	<input checked="" type="checkbox"/>
	Bobun Vegan	oui	<input checked="" type="checkbox"/>
	Tempura Crevettes	oui	<input checked="" type="checkbox"/>
	Tempura Legumes	oui	<input checked="" type="checkbox"/>
	Tempura Assortiment	oui	<input checked="" type="checkbox"/>
	Maki	oui	<input checked="" type="checkbox"/>
	California Rolls	oui	<input checked="" type="checkbox"/>
	Spring Rolls	oui	<input checked="" type="checkbox"/>

## Commandes

<div><b>Admin</b></div> <div>Produits</div> <div>Commandes</div> <div>Administration</div>	Les Commandes									
	À préparer					En attente de récupération				
	N° Commande	Date Création	État	Details	Gestion	N° Commande	Date Traité	État	Details	Gestion
	3 - H8AQTWP...	2025-04-15 1...	Crée	Details	Passer en A...	4 - FNJFP28KI9	2025-04-18 17:...	en Attente	Details	Passer en Finis
	7 - CF5W95CL...	2025-04-25 1...	Crée	Details	Passer en A...	6 - LVVFSXNBCQ	2025-04-23 09:...	en Attente	Details	Passer en Finis
	8 - XJNWMAV...	2025-04-25 1...	Crée	Details	Passer en A...	9 - EXOP5DMQ...	2025-04-25 12:...	en Attente	Details	Passer en Finis
	12 - LHDGXTB...	2025-04-25 1...	Crée	Details	Passer en A...	10 - XUIGPVC...	2025-04-25 12:...	en Attente	Details	Passer en Finis
						11 - UFB5MOB...	2025-04-25 12:...	en Attente	Details	Passer en Finis
						13 - 3ZKVZ4BZ...	2025-04-25 12:...	en Attente	Details	Passer en Finis

# Détails



Admin

Produits

Commandes

Administration

Retour

ID Commande: 8 - XJNWMAVVQR

Date de commande: 2025-04-25 11:47:04.0

État: Créée

Client: john@example.com

Plats Fixes:

Ramen au Poulet - Prix: 13.5€

Détails: Nouilles ramen, bouillon de poulet, filet de poulet, champignons de Paris, œuf, oignons verts.

California Roll Thon - Prix: 14.0€

Détails: Riz à sushi, nori, bâtonnets de crabe (kanikama), avocat, concombre, mayonnaise japonaise, graines de sésame.

Maki Saumon - Prix: 12.0€

Détails: Riz à sushi, nori, saumon frais, servi avec sauce soja, wasabi et gingembre mariné.

Plats Composés:

Maki - Prix: 12.6€

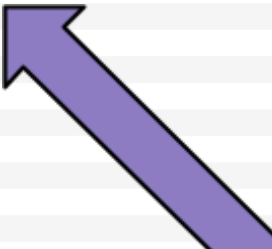
Détails: Avocat, Surimi, Boursin, Nori.

Spring Rolls - Prix: 12.2€


Détails: Avocat, Surimi, Mayonnaise japonaise.

California Rolls - Prix: 12.65€

Détails: Avocat, Surimi, Nori, Feuille de riz.



# Historique



Admin

Produits

Commandes

Administration

Retour

Historique

N°Commande	Date Création	Etat	
2 - O3SBASTWK0	2025-04-15 15:10:19.0	F	Détails
3 - H8AQTPWZQF	2025-04-15 17:03:30.0	F	Détails
4 - FNJFP28KI9	2025-04-18 17:26:44.0	F	Détails
5 - P87XLW3PDK	2025-04-23 09:31:05.0	F	Détails
6 - LVVFSXNBCQ	2025-04-23 09:31:52.0	F	Détails
7 - CF5W95CLS2	2025-04-25 11:45:29.0	F	Détails
8 - XJNWMAVVQR	2025-04-25 11:47:04.0	F	Détails
9 - EXOP5DMQLO	2025-04-25 12:34:41.0	F	Détails
10 - XUIGPVCMTS	2025-04-25 12:34:58.0	F	Détails
11 - UFBSMOBBHZ	2025-04-25 12:35:08.0	F	Détails
12 - LHDGXTBNIW	2025-04-25 12:35:16.0	F	Détails
13 - 3ZKVZ4BZXH	2025-04-25 12:35:24.0	F	Détails

Détails

Détails

Détails

Détails

Détails

Détails

Détails

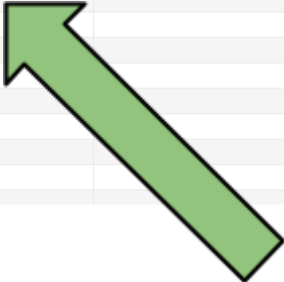
Détails

Détails

Détails

Détails

Détails



## **Administration**

[illegible]

## **Partie Technique**

## Planification & Organisation

- Utilisation de **Excel** pour planifier les tâches, suivre l'avancement, et attribuer les responsabilités.
- Utilisation de **Discord** pour la communication entre les collaborateurs.
- Versioning avec **GitHub**, permettant le travail collaboratif et la gestion de branches.

## Base de données

- Planification du schéma de la base de données : tables `users`, `produits`, `commandes`, `panier`, etc.
- Mise en place des **migrations Laravel** pour créer les tables.
- phpMyAdmin une manipulation facilitée.

## Environnement de travail local **Docker**

- Base de données via un conteneur MySQL (port 3306).
- phpMyadmin via un conteneur (port 8080).

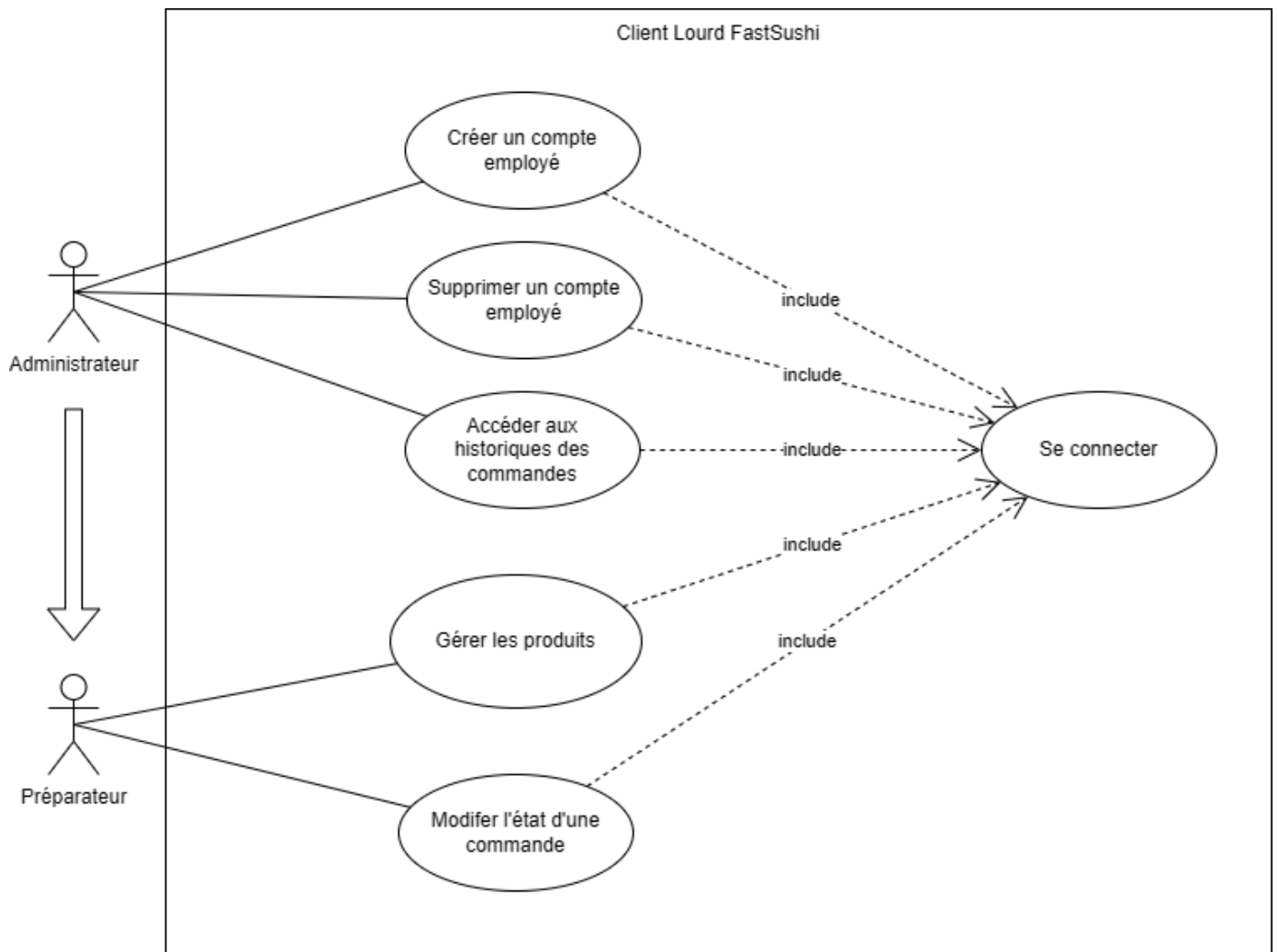
## Client Lourd - JavaFX

- Front avec **JavaFX** et **Scene Builder**
- **Fonctionnalités :**
  - **Gestion de produits**
    - Vue
      - produitView.fxml
    - Controller
      - produitFixeController.java
    - DAO
      - produitFixeDAO
  - **Gestion de comptes**
    - Vue
      - AdministrationView.fxml
    - Controller
      - AdministrationController.java
    - Service / Model
      - UsersService.java
      - User.java
  - **Gestion des commandes**
    - Vues
      - CommandesView.fxml
      - CommandeDetailView.fxml
      - HistoryCommandesView.fxml
    - Controller
      - HistoryCommandesController.java
      - CommandesController.java
      - CommandesDetailController.java
    - Service / Model
      - CommandeService.java
      - Commande.java



# Annexes

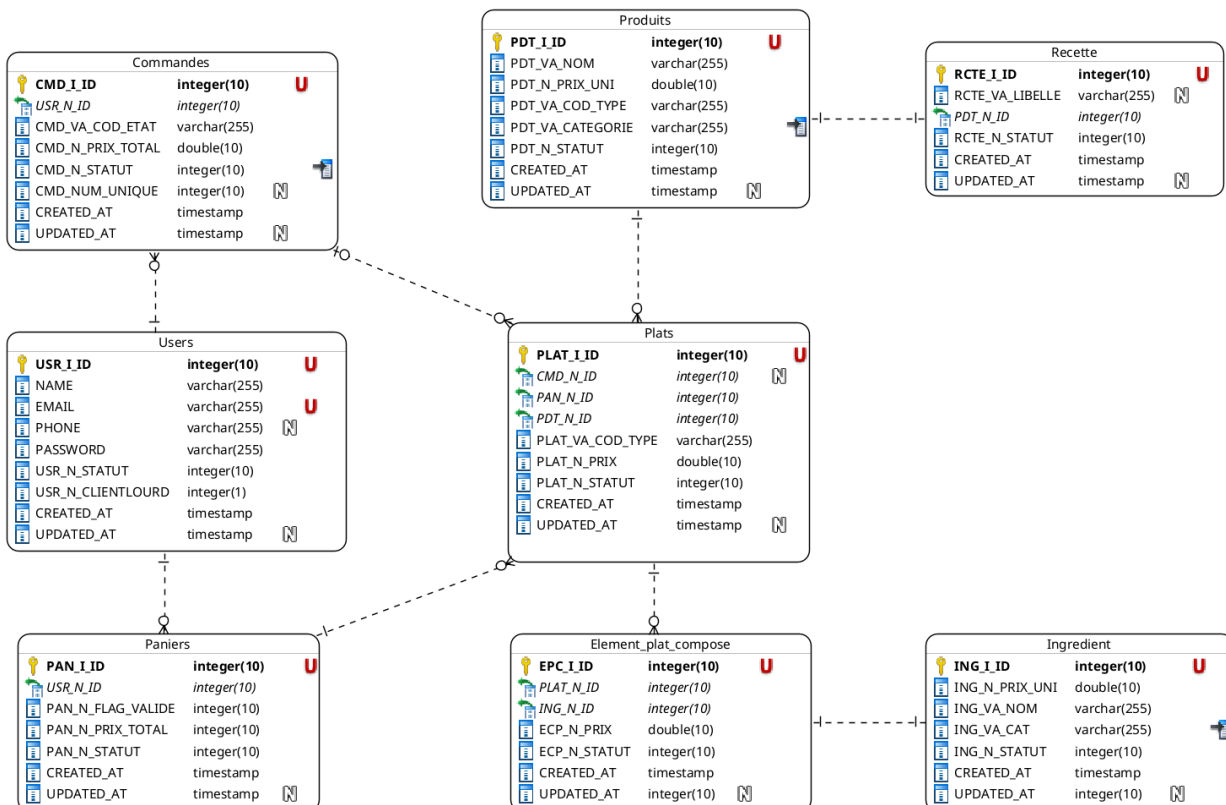
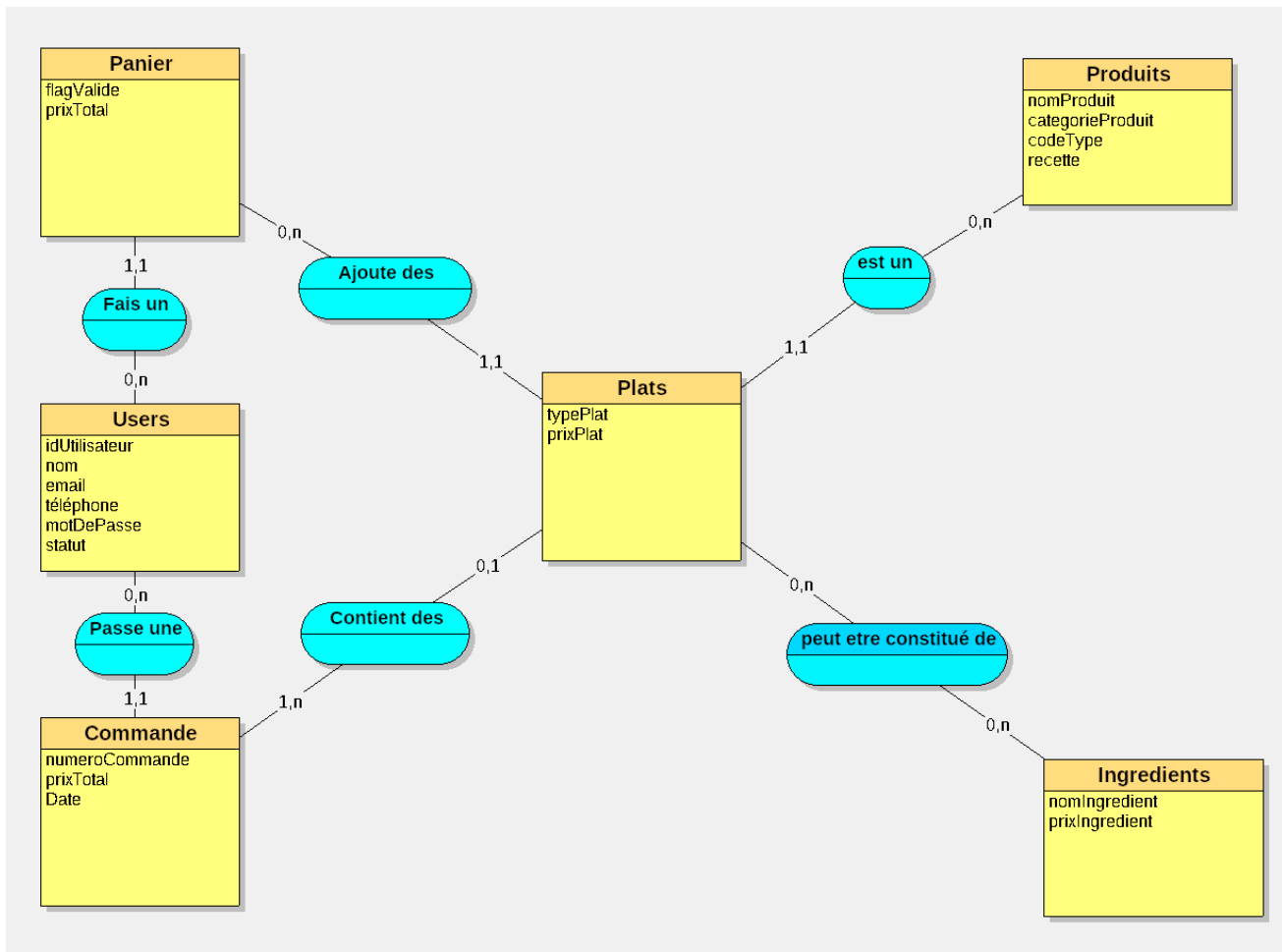
## A1. Use Case - Client Lourd



## A2. Planning - Client Lourd

Tâches	Sous-tâches	Description	Durée estimée
GitHub	Créer Repository distant : CLIENT LOURD	Créer repo, Ajouter collaborateurs, Créer les branches	0,5 J
	Lien avec le dossier local	Récupérer le repo distant, s'identifier (generer token sur GitHub)	0,5 J
Modélisation	UseCase	Conception de la vision globale du système	0,5 J
	MCD / MLD	Structure BDD	1 J
Dépendance	JavaFX	Intégrer les librairie JavaFX au buildpath	0,5 J
	Bcrypt	Hasher le mot de passe	0,5 J
	MySQL connector	Librairie pour se connecter à la BDD	0,5 J
SceneBuilder	Installation / prise en main SceneBuilder	Plugin pour modélisation des interfaces logiciel (génère FXML)	2 J
Login	Vue	2 champ (email + mdp), bouton se connecter	0,5 J
	Controller	Gère les actions utilisateur	0,5 J
	DAO	Compare le MDP saisi par l'utilisateur avec le mdp stocker en bdd	0,5 J
Produit	Vue	Affiche une table des produit (pdt_va_cod_type = PF), possibilité d'assigner la disponibilité	0,5 J
	Controller	Insère une nouvelle valeur pour la disponibilité	1 J
	DAO	Récupère les produits fixe	0,5 J
Commande	Vue	- 2 interfaces, une des commandes à préparer, l'autre des commande traités - Possibilité de voir l'historique des commandes + modification de leurs états - Possibilité de voir le détail de chaque commandes	2 J
	Controller	Mappage des colonnes avec les données + insertion de bouton pour chaque ligne	1 J
	Service	Récupération des commandes créé et traité + possibilité de récupérer les produits d'une commande	0,5 J
Detail	Vue	Affiche les éléments d'une commande (produit fixe + produit composé)	0,5 J
	Controller	Mappage des détails	0,5 J
Administration	Vue	Ajout, Modification, Suppression + affichage des utilisateurs	1 J
	controller	Gestion des ajouts + mappage des utilisateurs	0,5 J
	Service	Récupération des utilisateurs + insertion/ suppression des utilisateurs	0,5 J

## A3. Modélisation BDD (MCD + MLD)



## A4. Arborescence

```
|— config.properties
|— src
|   |— fastSushi
|   |   |— AdministrationController.java
|   |   |— BDD.java
|   |   |— CommandeDetailsController.java
|   |   |— Commande.java
|   |   |— CommandesController.java
|   |   |— CommandesService.java
|   |   |— HistoryCommandesController.java
|   |   |— HomeController.java
|   |   |— LoginController.java
|   |   |— Main.java
|   |   |— ProduitFixeController.java
|   |   |— ProduitFixeDAO.java
|   |   |— Produit.java
|   |   |— SceneManager.java
|   |   |— User.java
|   |   |— UserSession.java
|   |   |— UsersService.java
|   |   |— org
|   |   |   |— mindrot
|   |   |   |   |— jbcrypt
|   |   |— view
|   |   |— AdministrationView.fxml
|   |   |— CommandeDetailsView.fxml
|   |   |— CommandesView.fxml
|   |   |— HistoryCommandesView.fxml
|   |   |— home.fxml
|   |   |— LoginView.fxml
|   |   |— ProduitsView.fxml
|   |   |— style
|   |   |   |— AdministrationViewStyle.css
|   |   |   |— CommandesViewStyle.css
|   |   |   |— HistoryStyle.css
|   |   |   |— HomeStyle.css
```

## A5. HistoryCommandesView.fxml

```
<BorderPane maxHeight="-Infinity" maxWidth="-Infinity" minWidth="-Infinity"
style="-fx-background-color: #e4e6c3;" xmlns="http://javafx.com/javafx/23.0.1"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="fastSushi.HistoryCommandesController">
<stylesheets>
    <String fx:value="/view/style/HistoryStyle.css"/>
</stylesheets>

<top>
    <BorderPane minHeight="50.0" BorderPane.alignment="CENTER">
        <left>
            <Pane BorderPane.alignment="CENTER">
                <children>
                    <Text layoutX="10.0" layoutY="35.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="Historique">
                        <font>
                            <Font name="System Bold" size="30.0" />
                        </font>
                    </Text>
                </children>
            </Pane>
        </left>
        <right>
            <Button fx:id="buttonRetour" mnemonicParsing="false"
onAction="#afficheCommandes" text="Retour" BorderPane.alignment="CENTER">
                <BorderPane.margin>
                    <Insets right="20.0" />
                </BorderPane.margin>
            </Button>
        </right>
    </BorderPane>
</top>
<center>
    <TableView fx:id="tableHistorique" BorderPane.alignment="CENTER">
        <columns>
            <TableColumn fx:id="colNumCommande" minWidth="225.0" prefWidth="-1.0"
text="N°Commande" />
            <TableColumn fx:id="colDateCreation" minWidth="225.0" prefWidth="-1.0"
text="Date Création" />
            <TableColumn fx:id="colEtat" editable="true" minWidth="225.0"
prefWidth="-1.0" text="Etat" />
            <TableColumn fx:id="colDetail" minWidth="225.0" prefWidth="-1.0"
text="Détails" />
        </columns>
        <columnResizePolicy>
            <TableView fx:constant="CONSTRAINED_RESIZE_POLICY" />
        </columnResizePolicy>
    </TableView>
</center>
</BorderPane>
```

## A6. HistoryCommandesController.java

```
package fastSushi;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.control.cell.TextFieldTableCell;
import javafx.beans.property.ReadOnlyStringWrapper;
import javafx.beans.property.SimpleStringProperty;
import java.sql.Timestamp;
import javafx.util.converter.DefaultStringConverter;
public class HistoryCommandesController {
    @FXML
    private TableView<Commande> tableHistorique;
    @FXML
    private TableColumn<Commande, String> colNumCommande;
    @FXML
    private TableColumn<Commande, String> colDateCreation;
    @FXML
    private TableColumn<Commande, String> colEtat;
    @FXML
    private TableColumn<Commande, Void> colDetail;

    private CommandesService commandesService = new CommandesService();
    @FXML

    public void initialize() {
        // Resize automatique
        tableHistorique.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
        // Mappage des colonnes
        colNumCommande.setCellValueFactory(cellData ->
            new SimpleStringProperty(cellData.getValue().getIdCommande() + " - " +
cellData.getValue().getNumUnique()));
        colDateCreation.setCellValueFactory(cellData -> {
            Timestamp timestamp = cellData.getValue().getCreatedAt();
            return new SimpleStringProperty(timestamp != null ? timestamp.toString() :
""");
        });

        //Colonne Etat
        // rendre la colonne éditable
        tableHistorique.setEditable(true);
        colEtat.setCellFactory(TextFieldTableCell.forTableColumn(new
DefaultStringConverter()));
        // Enveloppe la valeur brute dans un ObservableValue
        colEtat.setCellValueFactory(cellData -> new
ReadOnlyStringWrapper(cellData.getValue().getCodeEtat()));
        // Mise à jour manuelle du champ
        colEtat.setOnEditCommit(event -> {
            Commande commande = event.getRowValue();
            String newEtat = event.getNewValue();
            if (newEtat.equals("T") || newEtat.equals("C") || newEtat.equals("F")) {
                commandesService.changerEtatCommande(commande.getIdCommande(),
newEtat);
            }
        });
    }
}
```

```

        chargerHistorique();
    } else {
        // Alerte d'erreur si la valeur est invalide
        Alert alert = new Alert(Alert.AlertType.WARNING);
        alert.setTitle("Valeur invalide");
        alert.setHeaderText("État non reconnu");
        alert.setContentText("Seules les valeurs T, C ou F sont autorisées.");
        alert.showAndWait();
        // Restaurer la valeur précédente
        event.getTableView().refresh();
    }
});

// Bouton Détails
colDetail.setCellFactory(param -> new TableCell<Commande, Void>() {
    private final Button btnDetails = new Button("Détails");
    @Override
    protected void updateItem(Void item, boolean empty) {
        super.updateItem(item, empty);
        if (empty) {
            setGraphic(null);
        } else {
            btnDetails.setOnAction(event -> {
                Commande commande = getTableView().getItems().get(getIndex());
                SceneManager.afficheDetails(commande);
            });
            setGraphic(btnDetails);
        }
    }
});

// Charger les données
chargerHistorique();
}

private void chargerHistorique() {
    commandesService.refreshCommandesListes();

    ObservableList<Commande> historique =
FXCollections.observableArrayList(commandesService.getCommandesFinis());

    tableHistorique.setItems(historique);
}

@FXML
public void afficheCommandes() {
    SceneManager.afficheCommandes();
}
}

```

## A7. CommandesService.java

```
public class CommandesService {
    private List<Commande> commandesEnCreation = new ArrayList<>();
    private List<Commande> commandesTraitees = new ArrayList<>();
    private List<Commande> commandesFinis = new ArrayList<>();
    public CommandesService() {
        // MAJ des listes
        refreshCommandesListes();
    }

    // Getter pour récupérer les commandes en création
    public List<Commande> getCommandesEnCreation() {
        return this.commandesEnCreation;
    }
    // Getter pour récupérer les commandes traitées
    public List<Commande> getCommandesTraitees() {
        return this.commandesTraitees;
    }
    public List<Commande> getCommandesFinis() {
        return this.commandesFinis;
    }

    public void changerEtatCommande(int idCommande, String nouvelEtat) {
        //MAJ BDD
        BDD db = new BDD();
        db.openConnexion();

        String updateCommandeSQL = "UPDATE Commandes SET cmd_va_cod_etat = ? "
                                   + "WHERE cmd_i_id = ?;";

        try {
            PreparedStatement stmt =
db.getConnection().prepareStatement(updateCommandeSQL);
            stmt.setString(1, nouvelEtat);
            stmt.setInt(2, idCommande);
            stmt.executeUpdate();
        }
        catch (SQLException e) {
            System.err.println("Erreur lors de l'exécution de la requête pour les
commandes traitées : " + e.getMessage());
        }
        db.closeConnexion();
        //MAJ des Listes
        refreshCommandesListes();
    }

    public void refreshCommandesListes() {
        BDD db = new BDD();
        db.openConnexion();
        this.commandesEnCreation.clear();
        this.commandesTraitees.clear();
        this.commandesFinis.clear();

        //Recupération des Commandes Cree
        String commandesEnCreationSQL = "SELECT cmd_i_id, cmd_va_cod_etat,
cmd_num_unique, created_at, updated_at "
```



```

+ "FROM Commandes WHERE
cmd_va_cod_etat = 'C' AND cmd_n_statut = 1";
    try {
        Statement stmt = db.getConnexion().createStatement();
        ResultSet rs = stmt.executeQuery(commandesEnCreationSQL);

        while (rs.next()) {
            int id = rs.getInt("cmd_i_id");
            String etat = rs.getString("cmd_va_cod_etat");
            String numUnique = rs.getString("cmd_num_unique");
            Timestamp created_at = rs.getTimestamp("created_at");
            Timestamp updated_at = rs.getTimestamp("updated_at");

            this.commandesEnCreation.add(new Commande(id, etat,
numUnique,created_at,updated_at ));
        }
    } catch (SQLException e) {
        System.err.println("Erreur lors de l'exécution de la requête pour les
commandes en création : " + e.getMessage());
    }

    //CommandeTraite
    String commandesTraiteSQL = "SELECT cmd_i_id, cmd_va_cod_etat, cmd_num_unique,
created_at, updated_at "
        + "FROM Commandes WHERE cmd_va_cod_etat = 'T' AND cmd_n_statut = 1";
    try {
        Statement stmt = db.getConnexion().createStatement();
        ResultSet rs = stmt.executeQuery(commandesTraiteSQL);

        while (rs.next()) {
            int id = rs.getInt("cmd_i_id");
            String etat = rs.getString("cmd_va_cod_etat");
            String numUnique = rs.getString("cmd_num_unique");
            Timestamp created_at = rs.getTimestamp("created_at");
            Timestamp updated_at = rs.getTimestamp("updated_at");

            this.commandesTraitees.add(new Commande(id, etat, numUnique,
created_at,updated_at));
        }
    } catch (SQLException e) {
        System.err.println("Erreur lors de l'exécution de la requête pour les
commandes traitées : " + e.getMessage());
    }

    //CommandeFinis
    String commandesFinisSQL = "SELECT cmd_i_id, cmd_va_cod_etat, cmd_num_unique,
created_at, updated_at "
        + "FROM Commandes WHERE cmd_va_cod_etat = 'F' AND cmd_n_statut = 1";
    try {
        Statement stmt = db.getConnexion().createStatement();
        ResultSet rs = stmt.executeQuery(commandesFinisSQL);

        while (rs.next()) {
            int id = rs.getInt("cmd_i_id");
            String etat = rs.getString("cmd_va_cod_etat");
            String numUnique = rs.getString("cmd_num_unique");
            Timestamp created_at = rs.getTimestamp("created_at");
            Timestamp updated_at = rs.getTimestamp("updated_at");

```

```

        this.commandesFinis.add(new Commande(id, etat, numUnique,
created_at,updated_at));
    }
} catch (SQLException e) {
    System.err.println("Erreur lors de l'exécution de la requête pour les
commandes finis : " + e.getMessage());
}
db.closeConnexion();
}

public void afficherCommandes() {
    System.out.println("Commandes en Création:");
    for (Commande commande : commandesEnCreation) {
        System.out.println("ID Commande: " + commande.getIdCommande() +
            ", Code Etat: " + commande.getCodeEtat() +
            ", Numéro Unique: " + commande.getNumUnique());
    }
    System.out.println("\nCommandes Traitées:");
    for (Commande commande : commandesTraitees) {
        System.out.println("ID Commande: " + commande.getIdCommande() +
            ", Code Etat: " + commande.getCodeEtat() +
            ", Numéro Unique: " + commande.getNumUnique());
    }
}

public List<Produit> getProduitsFixe(Commande commande) {
    BDD db = new BDD();
    db.openConnexion();

    int idCommande = commande.getIdCommande();
    String ProduitsFixeCommandeSQL = "SELECT pr.pdt_i_id, pr.pdt_va_nom,
pr.pdt_n_prix_uni "
        + "FROM Plats pl "
        + "INNER JOIN Produits pr ON pl.pdt_n_id =
pr.pdt_i_id "
        + "WHERE pl.cmd_n_id = ? AND pl.plat_va_cod_type
= 'PF' AND pl.plat_n_statut = 1;";
    List<Produit> produitsFixe = new ArrayList<>();

    try {
        // Les Produits
        PreparedStatement stmt =
db.getConnexion().prepareStatement(ProduitsFixeCommandeSQL);
        stmt.setInt(1, idCommande);
        ResultSet rsPF = stmt.executeQuery();
        while (rsPF.next()) {
            // Les Détails
            String DetailsPFSQL = "SELECT r.rcte_va_libelle "
                + "FROM Recettes r WHERE r.pdt_n_id = ?";
            PreparedStatement stmt2 =
db.getConnexion().prepareStatement(DetailsPFSQL);
            stmt2.setInt(1, rsPF.getInt("pdt_i_id"));
            ResultSet rsPFDetail = stmt2.executeQuery();
            List<String> DetailsPF = new ArrayList<>();
            while (rsPFDetail.next()) {
                DetailsPF.add(rsPFDetail.getString("rcte_va_libelle"));
            }
        }
    }
}

```

```

        }
        Produit newPF = new Produit(rsPF.getString("pdt_va_nom"),
DetailsPF.toArray(new String[0]), rsPF.getDouble("pdt_n_prix_uni"));
        produitsFixe.add(newPF);
    }
} catch (SQLException e) {
    System.err.println("Erreur lors de l'exécution de la requête pour le détail
de la commande n°" + idCommande + " : " + e.getMessage());
} finally {
    db.closeConnexion();
}
return produitsFixe;
}

public List<Produit> getProduitsCompose(Commande commande) {
    BDD db = new BDD();
    db.openConnexion();
    int idCommande = commande.getIdCommande();

    String ProduitsComposeCommandeSQL = "SELECT pr.pdt_i_id, pl.plat_i_id,
pr.pdt_va_nom, pl.plat_n_prix "
        + "FROM Plats pl "
        + "INNER JOIN Produits pr ON pl.pdt_n_id =
pr.pdt_i_id "
        + "WHERE pl.cmd_n_id = ? AND
pl.plat_va_cod_type = 'PC' AND pl.plat_n_statut = 1;";
    List<Produit> produitsCompose = new ArrayList<>();

    try {
        // Les Produits
        PreparedStatement stmt =
db.getConnection().prepareStatement(ProduitsComposeCommandeSQL);
        stmt.setInt(1, idCommande);
        ResultSet rsPC = stmt.executeQuery();

        while (rsPC.next()) {
            // Les Détails
            String DetailsPCSQL = "SELECT ing.ing_va_nom, ing.ing_n_prix_uni "
                + "FROM Elements_plat_compose apc "
                + "INNER JOIN Ingredients ing ON apc.ing_n_id =
ing.ing_i_id "
                + "WHERE apc.plat_n_id = ?";
            PreparedStatement stmt2 =
db.getConnection().prepareStatement(DetailsPCSQL);
            stmt2.setInt(1, rsPC.getInt("plat_i_id"));
            ResultSet rsPCDetail = stmt2.executeQuery();

            List<String> DetailsPC = new ArrayList<>();
            while (rsPCDetail.next()) {
                DetailsPC.add(rsPCDetail.getString("ing_va_nom"));
            }
            Produit newPC = new Produit(rsPC.getString("pdt_va_nom"),
DetailsPC.toArray(new String[0]), rsPC.getDouble("plat_n_prix"));
            produitsCompose.add(newPC);
        }
    } catch (SQLException e) {

```

```

        System.err.println("Erreur lors de l'exécution de la requête pour le détail
de la commande n°" + idCommande + " : " + e.getMessage());
    } finally {
        db.closeConnexion();
    }
    return produitsCompose;
}

public String getMailUser(Commande commande) {
    BDD db = new BDD();
    db.openConnexion();
    int idCommande = commande.getIdCommande();
    String UserSQL = "SELECT DISTINCT u.email "
        + "FROM users u "
        + "INNER JOIN Commandes c ON u.usr_i_id = c.usr_n_id "
        + "WHERE c.cmd_i_id = ?;";

    try {
        PreparedStatement stmt = db.getConnexion().prepareStatement(UserSQL);
        stmt.setInt(1, idCommande);
        ResultSet rs = stmt.executeQuery();
        if (rs.next()) {
            return rs.getString("email");
        } else {
            System.err.println("Aucun utilisateur trouvé pour la commande n°" +
idCommande);
            return null;
        }
    } catch (SQLException e) {
        System.err.println("Erreur lors de l'exécution de la requête pour le Mail
de la commande n°" + idCommande + " : " + e.getMessage());
        return null;
    } finally {
        db.closeConnexion();
    }
}
}

```

## A8. Commandes.java

```
public class Commande {

    private int idCommande;
    private String codeEtat; // ("C" -> Crée , "T" -> Traité , "F" -> Finis)
    private String numUnique;
    private Timestamp createdAt;
    private Timestamp updatedAt;

    public Commande(int idCommande, String codeEtat, String numUnique,
Timestamp createAt, Timestamp updateAt) {
        this.idCommande = idCommande;
        this.codeEtat = codeEtat;
        this.numUnique = numUnique;
        this.updatedAt = updateAt;
        this.createdAt = createAt;
    }

    public int getIdCommande() {
        return this.idCommande;
    }

    public String getCodeEtat() {
        return this.codeEtat;
    }

    public String getNumUnique() {
        return this.numUnique;
    }

    public Timestamp getUpdatedAt() {
        return this.updatedAt;
    }

    public Timestamp getCreatedAt() {
        return this.createdAt;
    }

}
```

## A9. Produit.java

```
package fastSushi;

public class Produit {

    private String nom;
    private String[] details;
    private double prix;

    public Produit(String nom, String[] details, double prix) {
        this.nom = nom;
        this.details = details;
        this.prix = prix;
    }

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    public String[] getDetails() {
        return details;
    }

    public void setDetails(String[] details) {
        this.details = details;
    }

    public double getPrix() {
        return prix;
    }

    public void setPrix(double prix) {
        this.prix = prix;
    }
}
```