

QUALITE LOGICIELLE-ETUDE DE CAS ESPORT



Karima Kouaouci

Rand Findakly

1	Contexte du projet :	2
2	Objectif :	2
3	Principales fonctionnalités demandées :	2
4	Organisation du projet et mode de travail :	3
5	Choix des technologies :	3
5.1	Technologies utilisées :	3
6	Présentation du défèrent taches	5
7	Maquette :	5
8	Page WIKI & BDD	7
9	Planning Pocker:	7
10	Base de données (MCD) :	8
11	DEVELOPPEMENT	8
12	Test Unitaire :	9
13	Test d'intégration :	12
14	Les difficultés rencontrées et les solutions apportées :	14

1 Contexte du projet :

Esport Tourmant est une structure d'eSport située dans le sud-est de Lyon. De la conception à la mise en œuvre et à l'encadrement de chaque événement, Esport Tourmant propose des solutions sur mesure pour les fans de jeux vidéo en France. Ces événements, organisés sur plusieurs jours dans des lieux spécifiques, rassemblent des centaines de joueurs participant à des tournois de jeux vidéo populaires tels que League of Legends, FIFA, Call of Duty, PES et Counter Strike.

2 Objectif :

L'objectif de l'application est de faciliter la vente de produits dérivés, à la fois en ligne et en boutique, lors des événements eSport. Elle permet aux utilisateurs de créer un compte, d'effectuer des paiements sécurisés (carte bancaire ou PayPal), et de gérer leurs livraisons. Ce système permettra une gestion fluide des opérations, une expérience utilisateur améliorée.

3 Principales fonctionnalités demandées :

1. Gestion des événements eSport :

- Création d'un module de gestion des événements permettant de définir les paramètres de chaque événement (dates, lieux, jeux, formats de tournoi, etc.)
- Développement d'un système d'inscription des joueurs (individuel ou par équipe) avec vérification des doublons
- Implémentation d'un outil de planification et d'organisation des tournois (génération des matchs, saisie des résultats, mise à jour des classements, etc.)
- Conception d'une interface de diffusion des informations et du suivi des événements (classements, résultats, etc.) accessible aux participants et au public

2. Vente de produits dérivés :

- Création d'un catalogue de produits dérivés avec les informations pertinentes (code produit, nom, famille, prix d'achat, prix de vente)
- Développement d'une plateforme de vente en ligne avec gestion des comptes utilisateurs (email, mot de passe, adresses de facturation et de livraison)
- Intégration des modes de paiement acceptés (cartes bancaires, PayPal) avec sécurisation des transactions
- Gestion des stocks et des délais de livraison en fonction de la disponibilité des produits

- Application des règles de frais de port (livraison gratuite à partir de 75€, frais variables selon le pays).
- Mise en place d'un système de vente sur place pendant les événements avec suivi des stocks et des ventes

3. Suivi et analyses :

- Conception de tableaux de bord et de rapports permettant de visualiser les ventes de produits dérivés (en ligne et sur place)
- Développement d'outils d'analyse statistique sur les ventes, les participants, les résultats des tournois, etc.
- Intégration de fonctionnalités permettant de distinguer les ventes en ligne des ventes sur place

4 Organisation du projet et mode de travail :

Nous avons opté pour une méthodologie agile, en suivant le cadre Scrum. Chaque sprint dure une semaine et commence par un sprint planning, suivi d'une revue de code à la fin du sprint, ainsi qu'un rendez-vous hebdomadaire avec le client pour faire le point.

Un daily stand-up de 10 minutes est prévu chaque jour avec l'équipe, composée de deux développeurs.

5 Choix des technologies :

Les technologies utilisées ont été choisies en fonction des compétences maîtrisées par les membres de l'équipe, afin d'assurer une productivité maximale et une meilleure qualité de développement.

5.1 Technologies utilisées :

Front-end :

- **Angular 13** (TypeScript) (framework web)
- **Bootstrap** (mise en page)
- **HTML** (structure web)
- **SCSS** (CSS amélioré)
- **Angular Materials** (composants UI)

Back-end :

- **Java** (langage backend)
- **Spring Boot** (framework backend)
- **Hibernate** (ORM Java)
- **JUnit5**

Gestion de code :

- **Git** (gestion version)
- **GitHub** (hébergement code)

Base de données :

- **MySQL** (base de données)
- **Workbench** (outil gestion)

Maquette :

- **Figma** (conception UI)

Communication :

- **Discord** (messagerie équipe)

Wiki & BDD :

- **ClickUp** (gestion projet)

Gestion des tâches :

- **Azure DevOps** (suivi tâches)
- **User story**

6 Présentation du défèrent taches

Produit_Dérivé Team

Backlog Analytics + New Work Item View as Board Column Options

Order	ID	Title	Assigned To	State
1	5	FRAIS DE PORT	...	To Do
2	4	MODE DE PAIEMENT		To Do
3	3	PRODUIT		To Do
4	2	EVENEMENT		To Do
5	1	AUTHENTIFICATION	Karima Kouao...	Doing
6	9	FRONT END	Karima Kouao...	Doing
7	10	backend		Doing

Planning

Drag and drop work items to include them in a sprint.

Produit_Dérivé Team Backlog

developpement **Current** 05/09/2024 - 06/09/2024
2 working days
No work scheduled yet

TEST 06/09/2024 - 06/09/2024
1 working days
No work scheduled yet

Sprint 1
Planned Effort: -
11

+ New Sorint

5 septembre - 6 septembre
2 work days remaining

Taskboard Backlog Capacity Analytics + New Work Item Column Options

Collapse all

13 Developpement
Unassigned
State To Do

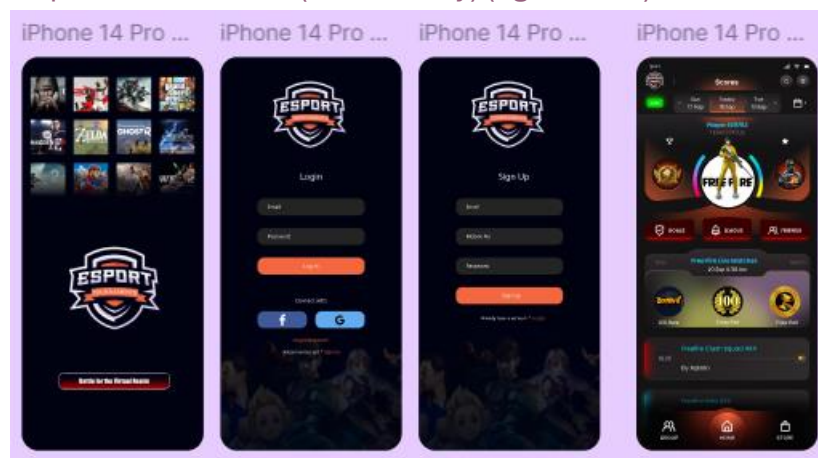
16 Tester API AVEC POSTMAN
Unassigned
State To Do

15 creation de API
Unassigned
State Doing

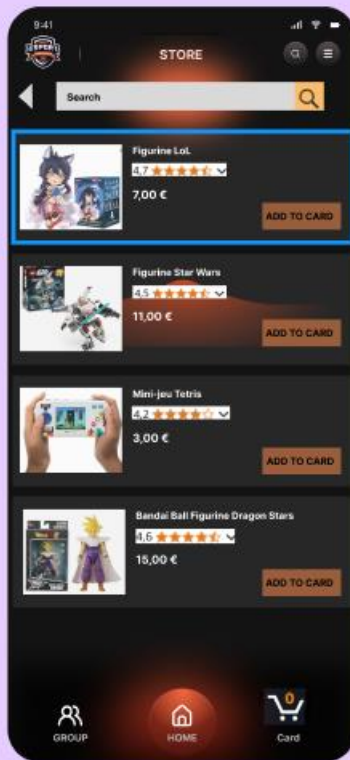
14 Tester la connexion de base de donnée
Karima Kouao
State Done

7 Maquette :

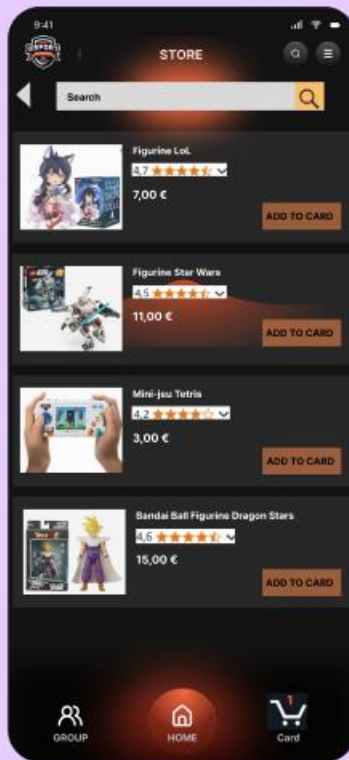
E sports tournament (Community) (figma.com)



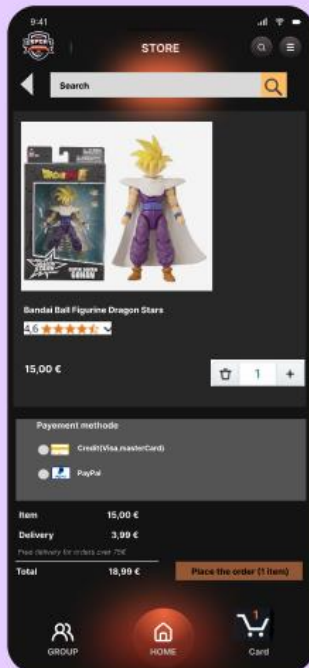
iPhone 14 Pro Max - 6



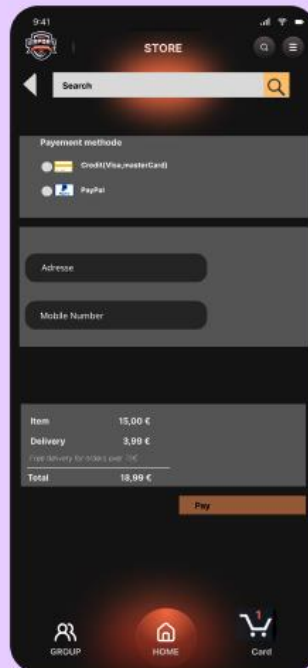
iPhone 14 Pro Max - 8



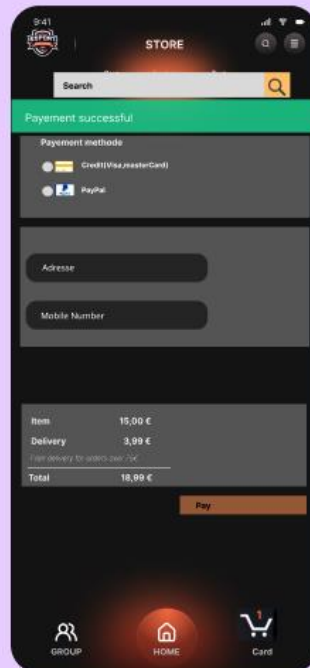
iPhone 14 Pro Max - 7



iPhone 14 Pro Max - 9



iPhone 14 Pro Max - 10



8 Page WIKI & BDD

<https://doc.clickup.com/9012238244/d/h/8cjgnx4-412/85de79115dbae35>



BDD

▼ Création de compte utilisateur

Fonctionnalité : Création de compte utilisateur sur la plateforme en ligne

Scénario : Un utilisateur crée un compte pour acheter des produits

- **Étant donné** que je suis sur la page d'inscription
- **Quand** je saisis mon email, mon mot de passe, et mes informations de facturation
- **Et** je clique sur le bouton "S'inscrire"
- **Alors** mon compte doit être créé et je dois être redirigé vers la page de connexion
- **Et** je reçois un email de confirmation pour vérifier mon compte

► Consultation des produits

► Passer une commande

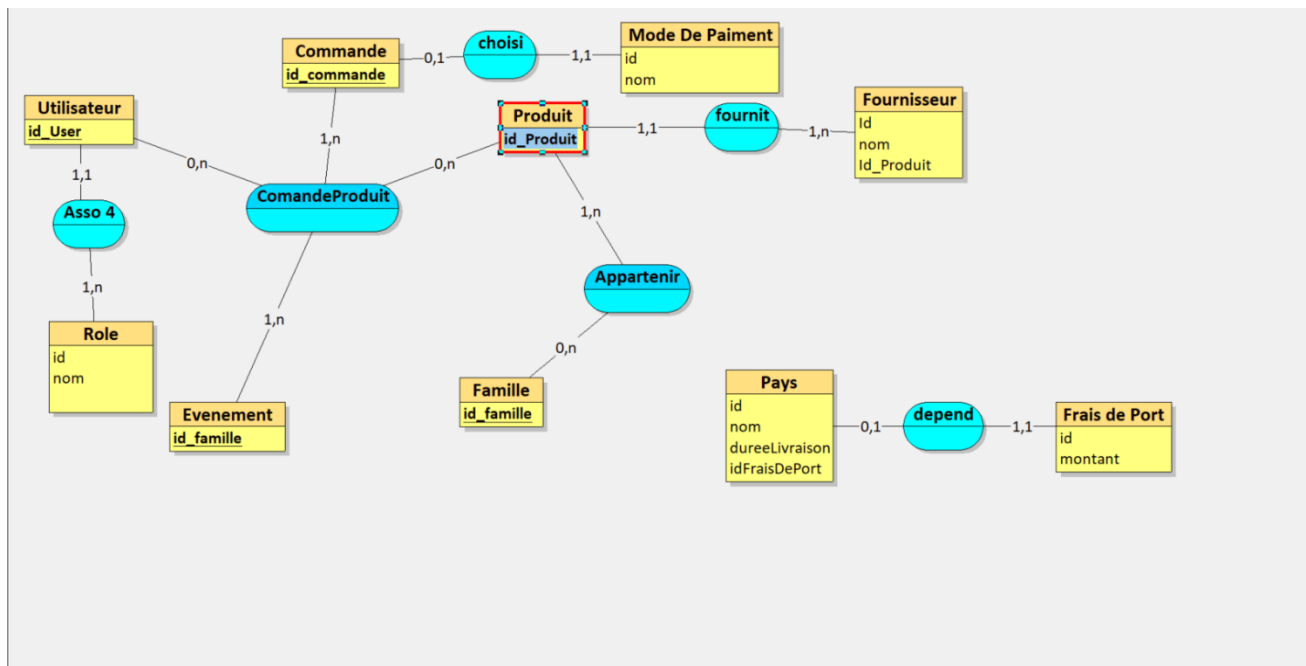
9 Planning Poker :

1	5	10
haute priorité		basse priorité

Summary	Key	Story Points	Priorité
Création du compte utilisateur(signe in)	PP-1	3	
creation du catalogue de produit	PP-2	2	2
Développer un système de vente en ligne	PP-3	7	1
Mettre en place le paiement en ligne	PP-4	3	2
Intégrer un module de gestion des frais de livraison	PP-5	2	1
Gestion des stocks et des délais de livraison	PP-6	6	1
Mise en place d'un système de vente sur place pendant les événements avec suivi des stoc	PP-7	10	1
Conception de tableaux de bord et de rapports permettant de visualiser les ventes de prod	PP-8	5	3
Développement d'outils d'analyse statistique sur les ventes, les participants, les résultats d	PP-9	5	3
Intégration de fonctionnalités permettant de distinguer les ventes en ligne des ventes sur p	PP-10	3	8
integrer un programme de fidelite	PP-11	4	10

10 Base de données (MCD) :

Modélisation de base de données : Avant de créer la base de données, il faut commencer par la modélisation, afin de mettre en place un plan de construction. Cette modélisation sera nécessaire pour déterminer les entités et les relations entre les tables. Pour cela j'ai utilisé **Looping**.



11 DEVELOPPEMENT

<https://github.com/kouaouci/Qualit-Logicielle-Produit-Deriv->

Nous avons développé la partie backend avec le framework Spring Boot, et compte tenu du temps imparti, nous avons utilisé la console pour afficher nos pages au lieu d'une interface utilisateur complète.

```

19 ConsoleController consoleController = applicationContext.getBean(ConsoleController.class);
20 //SpringApplication.run(EsportTournamentApplication.class, args);
21 consoleController.printConsoleApp();
22 }
23
24 }
25

```

```

-----
STATISTIQUES DE PRODUITS DERIVE
-----

----- MENU -----
(Pour naviguer dans le menu, entrer le numéro correspondant entre parenthèses)

(1) : Statistiques des commandes faites
(2) : Statistiques des familles de produits
(3) : Statistiques des produits
-----
Rentrer dans la page : 1

```

```

-----
PAGE (1) STATISTIQUES COMMANDES
-----
|          | SUR PLACE | EN LIGNE |          |
-----
| SOMME DE PRIX D'ACHAT | 14.0    | 0.0    |          |
| SOMME DE PRIX DE VENTE | 14.0    | 0.0    |          |
-----

(1) Retour au menu
Rentrer dans la page :

```

12 Test Unitaire :

Nous avons testé la fonctionnalité suivante :

- La méthode `findUserByEmail()` qui permet de vérifier le comportement du service utilisateur en simulant la recherche d'un utilisateur à partir de son adresse email.
- Nous utilisons `@ExtendWith(MockitoExtension.class)` pour intégrer Mockito avec JUnit 5.
- Nous ajoutons un mock pour `UtilisateurDAO` et nous injectons ce mock dans `UtilisateurService` en utilisant `@InjectMocks`. Cela simule mieux la structure réelle de votre application.

- Nous testons deux scénarios : un où l'utilisateur existe et un où il n'existe pas.
- Dans le test `testFindUserByEmail_UserExists`, nous vérifions que lorsqu'un utilisateur est trouvé, il est correctement retourné dans un `Optional`.
- Dans le test `testFindUserByEmail_UserDoesNotExist`, nous vérifions que lorsqu'aucun utilisateur n'est trouvé, une `Optional` vidéo est retournée.
- Nous utilisons `verify` pour nous assurer que la méthode `findByEmail` du DAO est appelée exactement une fois avec le bon email.
- Nous supposons qu'il `UtilisateurService` existe une méthode `findUserByEmail` qui encapsule l'appel au DAO et gère le retour d'un `Optional`. Si ce n'est pas le cas, vous devriez envisager d'ajouter cette couche de service.

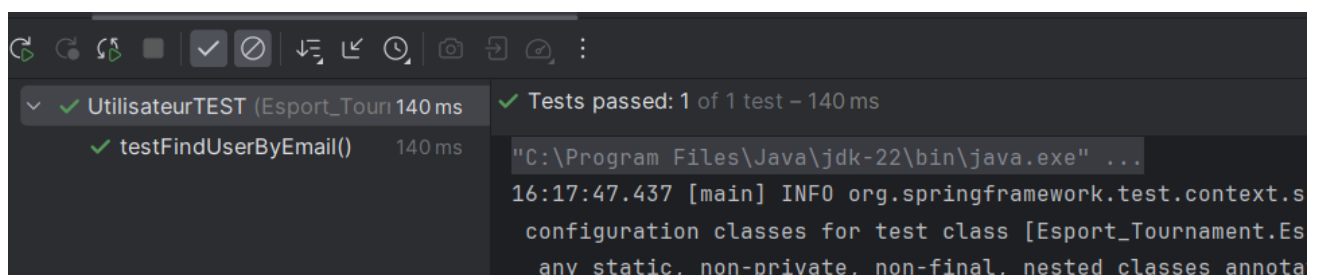
```
@Test
void testFindUserByEmail() {

    // Given
    String email = "test@example.com";
    Utilisateur expectedUtilisateur = new Utilisateur( email: "John Doe", email);

    when(utilisateurDAO.findByEmail(email)).thenReturn(expectedUtilisateur);

    // When
    Optional<Utilisateur> result = Optional.ofNullable(utilisateurDAO.findByEmail(email));

    // Then
    assertTrue(result.isPresent());
    assertEquals(expectedUtilisateur, result.get());
    verify(utilisateurDAO).findByEmail(email);
}
```



UtilisateurTEST (Esport_Tourn 140 ms) ✓ Tests passed: 1 of 1 test – 140 ms

testFindUserByEmail() 140 ms

"C:\Program Files\Java\jdk-22\bin\java.exe" ...

16:17:47.437 [main] INFO org.springframework.test.context.s
configuration classes for test class [Esport_Tournament.Es
any static, non-private, non-final, nested classes annota

```

@Test
public void testGetVenteFamille2() throws Exception {
    //given
    Famille famille = new Famille( nom: "figurine", code: "FIG");
    Famille getFamille = familleDAO.save(famille);

    Fournisseur fournisseur = new Fournisseur( nom: "F1");
    fournisseurDAO.save(fournisseur);

    ModeDePaiement modePaiement = new ModeDePaiement( nom: "carte bancaire");
    modePaiementDAO.save(modePaiement);

    FraisDePort fraisDePort = new FraisDePort( montant: 2.0);
    fraisDePortDAO.save(fraisDePort);

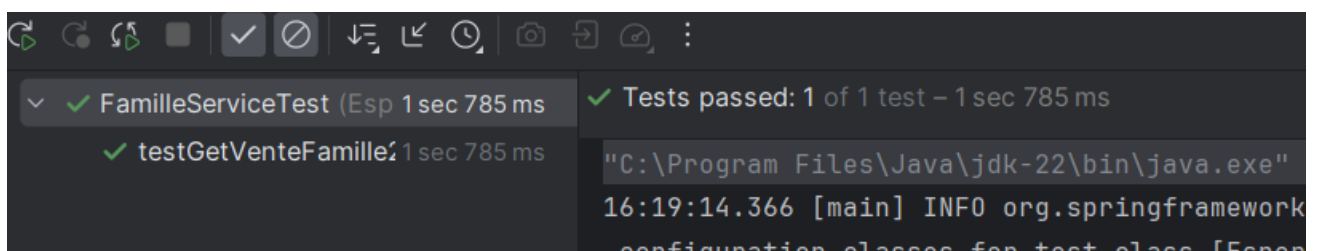
    Pays pays = new Pays( nom: "france", dureeLivraison: 1);
    pays.setFraisDePort(fraisDePort);
    paysDAO.save(pays);

    Adresse adresse;
    adresse = new Adresse( rue: "a", codePostal: "b", ville: "c");
    adresse.setPays(pays);
    adresseDAO.save(adresse);

    Role role = new Role( nom: "client");
    roleDAO.save(role);

    Utilisateur utilisateur = new Utilisateur( email: "aaa", motDePasse: "bbb");
    utilisateur.setRole(role);
    utilisateur.setPays(pays);

```



The screenshot shows the bottom toolbar of an IDE with icons for running, debugging, and testing. Below the toolbar, a test results panel displays the following information:

- Test Suite:** FamilleServiceTest (Esp 1 sec 785 ms)
- Test Result:** ✓ Tests passed: 1 of 1 test – 1 sec 785 ms
- Test Name:** ✓ testGetVenteFamille2 1 sec 785 ms
- Log Output:**

```

"C:\Program Files\Java\jdk-22\bin\java.exe"
16:19:14.366 [main] INFO org.springframework
configuration classes for test class [Espen

```

13 Test d'intégration :

La `getVentresFamille` méthode. Ce test est assez complet et couvre de nombreux aspects du système. Décomposons-le et discutons de sa structure et de son objectif :

- Le test met en place un scénario complexe avec plusieurs entités : **Famille**, **Fournisseur**, **ModeDePaiement**, **FraisDePort**, **Pays**, **Adresse**, **Role**, **Utilisateur**, **Produit**, **Commande** et **CommandeProduit**.
- Elle crée une famille (« figurine ») avec deux produits (P1 et P2).
- Il établit trois types d'ordres différents :
 - Une commande en ligne valide
 - Une commande hors ligne valide
 - Une commande en ligne invalide
- Il crée des entrées **CommandeProduit** pour associer des produits aux commandes.
- Enfin, il appelle la `getVentresFamille` méthode et vérifie les résultats.
- Ce test est utile car il :
 - Teste l'ensemble de la pile, de la couche de service jusqu'à la base de données.
 - Vérifie que la méthode calcule correctement le prix total et la quantité pour une famille.
 - Garantit que seules les commandes valides sont comptabilisées dans les calculs.

```

kouaouci
@Test
public void testGetVenteFamille() throws Exception {
    //given
    Famille famille = new Famille( nom: "figurine", code: "FIG");
    Famille getFamille = familleDAO.save(famille);

    Fournisseur fournisseur = new Fournisseur( nom: "F1");
    fournisseurDAO.save(fournisseur);

    ModeDePaiement modePaiement = new ModeDePaiement( nom: "carte bancaire");
    modePaiementDAO.save(modePaiement);

    FraisDePort fraisDePort = new FraisDePort( montant: 2.0);
    fraisDePortDAO.save(fraisDePort);

    Pays pays = new Pays( nom: "france", dureeLivraison: 1);
    pays.setFraisDePort(fraisDePort);
    paysDAO.save(pays);

    Adresse adresse;
    adresse = new Adresse( rue: "a", codePostal: "b", ville: "c");
    adresse.setPays(pays);
    adresseDAO.save(adresse);

    Role role = new Role( nom: "client");
    roleDAO.save(role);

    Utilisateur utilisateur = new Utilisateur( email: "aaa", motDePasse: "bbb");
    utilisateurDAO.save(utilisateur);
}
```

✓ FamilleServiceTest (Esp 1 sec 555 ms) ✓ testGetVenteFamille(1 sec 555 ms)	✓ Tests passed: 1 of 1 test – 1 sec 555 ms <pre> "C:\Program Files\Java\jdk-22\bin\java.exe" ... 16:28:33.249 [main] INFO org.springframework.test.context configuration classes for test class [Esport_Tournament. declare any static, non-private, non-final, nested class 16:28:33.416 [main] INFO org.springframework.boot.test.co </pre>
--	--

Tester le scénario où une famille n'a pas de ventes :

<pre> kouaouci @Test public void testGetVenteFamilleNoSales() throws Exception { //given Famille famille = new Famille(nom: "figurine", code: "FIG"); Famille getFamille = familleDAO.save(famille); Fournisseur fournisseur = new Fournisseur(nom: "F1"); fournisseurDAO.save(fournisseur); ModeDePaiement modePaiement = new ModeDePaiement(nom: "carte bancaire"); modePaiementDAO.save(modePaiement); FraisDePort fraisDePort = new FraisDePort(montant: 2.0); fraisDePortDAO.save(fraisDePort); Pays pays = new Pays(nom: "france", dureeLivraison: 1); pays.setFraisDePort(fraisDePort); paysDAO.save(pays); </pre>	
✓ FamilleServiceTest (Esport_Tournament.F 2 sec 201 ms) ✓ testGetVenteFamilleNoSales() 2 sec 201 ms	✓ Tests passed: 1 of 1 test – 2 sec 201 ms <pre> "C:\Program Files\Java\jdk-22\bin\java.exe" ... 16:30:44.384 [main] INFO org.springframework.test.context.su default configuration classes for test class [Esport_Tourna FamilleServiceTest does not declare any static, non-private 16:30:44.522 [main] INFO org.springframework.boot.test.conte @SpringBootConfiguration Esport_Tournament.Esport_Tournamen </pre>

14 Les difficultés rencontrées et les solutions apportées :

Parmi les difficultés rencontrées :

- Le manque de temps dédié à ce module, malgré son importance dans notre parcours.
- Un travail réalisé de manière superficielle en raison du délai limité.
- Un décalage dans le niveau de développement au sein de l'équipe.
- L'espacement entre les séances, entraînant une perte de certaines user stories et un manque de continuité.

Les solutions proposées incluent :

- Augmenter les heures de travail et rapprocher les séances.
- Adopter la méthode TDD (Test Driven Développement) pour rendre l'écriture des tests plus facile et fluide grâce à une pratique régulière.
- Rééquilibrer les groupes pour favoriser le partage des compétences et du savoir-faire entre les membres.