Première Partie : Stratégie de Test

1. Cahier de recettes de l'application

Fonctionnalités de l'application :

1. **Gestion des utilisateurs**

Inscription : Permet aux utilisateurs de s'inscrire avec un email, mot de passe, pseudo, date de naissance et rôle.

Connexion : Authentifie les utilisateurs avec leur email et mot de passe.

Profil utilisateur : Récupère les informations du profil utilisateur.

Modification de profil : Permet aux utilisateurs de modifier leur profil.

Suppression de compte : Permet aux utilisateurs de supprimer leur compte.

Réinitialisation de mot de passe : Envoie un email pour réinitialiser le mot de passe.

2. **Gestion des annonces**

Création d'annonce : Permet aux utilisateurs de créer une annonce avec des détails spécifiques (titre, description, catégorie, etc.).

Modification d'annonce : Permet de modifier les détails d'une annonce existante.

Suppression d'annonce : Permet de supprimer une annonce existante.

Consultation des annonces : Affiche toutes les annonces ou les annonces par catégorie, utilisateur ou type.

Recherche d'annonces similaires : Affiche des annonces similaires basées sur certains critères.

3. **Gestion des catégories**

Récupération des catégories : Permet de récupérer toutes les catégories disponibles.

Modification de catégorie : Permet de modifier une catégorie existante.

Suppression de catégorie : Permet de supprimer une catégorie.

2. Stratégie de test

Tests unitaires : Tester chaque fonction ou méthode isolément. Par exemple, tester les fonctions de contrôleur (comme register, login, createUserAd, etc.) pour s'assurer qu'elles se comportent correctement avec des entrées et sorties attendues.

Tests d'intégration : Tester comment différentes parties de l'application fonctionnent ensemble. Par exemple, s'assurer que les routes API renvoient les résultats corrects.

Tests fonctionnels ou end-to-end (E2E) : Tester des scénarios utilisateur complets de bout en bout. Par exemple, tester l'inscription, la connexion et la création d'une annonce depuis l'interface utilisateur jusqu'à la base de données, en passant par le backend.

3. Définir l'ensemble des tests nécessaires

En nous basant sur le cahier de recettes, voici l'ensemble des tests nécessaires :

Tests unitaires :

Tests pour la gestion des utilisateurs

register : Vérifie que l'inscription d'un nouvel utilisateur fonctionne comme prévu.

login : Vérifie que la connexion authentifie correctement un utilisateur.

getUserProfil : Vérifie que les informations du profil utilisateur sont correctement récupérées.

edit : Vérifie que la modification du profil utilisateur fonctionne.

delete : Vérifie que la suppression du compte utilisateur est correctement gérée.

resetPassword et setNewPassword : Vérifient le flux de réinitialisation du mot de passe.

Tests pour la gestion des annonces

createUserAd : Vérifie que la création d'une annonce fonctionne.

edit : Vérifie que la modification d'une annonce fonctionne.

delete : Vérifie que la suppression d'une annonce fonctionne.

getAll, getAllByCategory, getAllByUser, getAllByType : Vérifient que la récupération des annonces fonctionne correctement.

Tests pour la gestion des catégories

getAll : Vérifie que la récupération des catégories fonctionne.

edit : Vérifie que la modification d'une catégorie fonctionne.

delete : Vérifie que la suppression d'une catégorie fonctionne.

Tests d'intégration :

Tests des routes API

Vérifier que chaque route API renvoie les réponses correctes (ex. : codes d'état, messages d'erreur, données).

(Voir Tableau)

Tests Fonctionnels ou end-to-end (E2E) :

Scénarios utilisateur complets

Flux d'inscription et connexion : Tester qu'un utilisateur peut s'inscrire, se connecter et voir son profil.

Flux de création d'annonce : Tester qu'un utilisateur peut créer, modifier et supprimer une annonce.

Flux de réinitialisation de mot de passe : Tester qu'un utilisateur peut réinitialiser son mot de passe via un email.

4. Plan de test

Un plan de test est un document qui détaille comment chaque test sera effectué, les données utilisées, les résultats attendus et les conditions préalables.

#	Nom du Test	Objectif	Type de Test	Préconditions	Données de Test	Étapes	Résultat Attendu
1	**POST /api/register**	Vérifier l'enregistrement d'un utilisateur.	Intégration	Aucun utilisateur avec l'email fourni n'existe.	`email=coucou@gmail.com`, `password=coucou`, `pseudo=CoucouTest`, `birthdate=1900-01-01`	1. Envoyer une requête POST à `/api/register`. 2. Vérifier que la réponse est en JSON et a un statut 200.	L'utilisateur est enregistré, réponse avec JSON et statut 200.
2	**DELETE /api/user/**	Vérifier la suppression d'un utilisateur.	Intégration	L'utilisateur doit être connecté et autorisé.	`Authorization : Bearer ${randomUserToken}`	1. Envoyer une requête DELETE à `/api/user/${randomUserId}`. 2. Vérifier que la réponse est en JSON et a un statut 200.	Utilisateur supprimé, réponse avec JSON et statut 200.
3	**POST /api/login**	Vérifier la connexion d'un utilisateur.	Intégration	L'utilisateur doit exister dans le système.	`email=alizeamasse@gmail.com`, `password=test`	1. Envoyer une requête POST à `/api/login`. 2. Vérifier que la réponse est en JSON et a un statut 200.	Connexion réussie avec un token JWT, réponse avec JSON et statut 200.
4	**POST /api/users/create-annonces**	Vérifier la création d'une annonce.	Intégration	L'utilisateur doit être connecté et autorisé.	`category_id=8`, `condition_id=1`, `description=coucoutest`, `image=8`, `postal_code=33400`, `title=coucoutest`, `type_id=1`, `user_id=7`	1. Envoyer une requête POST à `/api/users/create-annonces`. 2. Vérifier que la réponse est en JSON et a un statut 200.	Annonce créée, réponse avec JSON et statut 200.

5	**PATCH /api/annonces/**	Vérifier la modification d'une annonce.	Intégration	L'annonce doit exister et l'utilisateur doit être autorisé.	`description=coucoutestModify`	1. Envoyer une requête PATCH à `/api/annonces/${AdId}`. 2. Vérifier que la réponse est en JSON et a un statut 200.	Annonce modifiée, réponse avec JSON et statut 200.
6	**DELETE /api/annonces/**	Vérifier la suppression d'une annonce.	Intégration	L'annonce doit exister et l'utilisateur doit être autorisé.	`Authorization: Bearer ${token}`	1. Envoyer une requête DELETE à `/api/annonces/${AdId}`. 2. Vérifier que la réponse est en JSON et a un statut 200.	Annonce supprimée, réponse avec JSON et statut 200.
7	**GET /api/categories**	Vérifier la récupération des catégories.	Intégration	Aucun.	`Accept: application/json`	1. Envoyer une requête GET à `/api/categories`. 2. Vérifier que la réponse est en JSON et a un statut 200.	Liste des catégories récupérée, réponse avec JSON et statut 200.
8	**GET /api/user/**	Vérifier la récupération des détails d'un utilisateur.	Intégration	L'utilisateur doit exister.	`Accept: application/json`	1. Envoyer une requête GET à `/api/user/1`. 2. Vérifier que la réponse est en JSON et a un statut 200.	Détails de l'utilisateur récupérés, réponse avec JSON et statut 200.
9	**GET /api/annonces/**	Vérifier la récupération des détails d'une annonce.	Intégration	L'annonce doit exister.	`Accept: application/json`	1. Envoyer une requête GET à `/api/annonces/1`. 2. Vérifier que la réponse est en JSON et a un statut 200.	Détails de l'annonce récupérés, réponse avec JSON et statut 200.
10	**POST /api/resetpassword**	Vérifier la réinitialisation du mot de passe.	Intégration	L'email doit être valide et exister dans le système.	`email=alizeamasse@gmail.com`	1. Envoyer une requête POST à `/api/resetpassword`. 2. Vérifier que la réponse est en JSON et a un statut 200.	Email de réinitialisation envoyé, réponse avec JSON et statut 200.

11	**PATCH /api/newpassword**	Vérifier le changement de mot de passe.	Intégration	L'utilisateur doit être connecté et autorisé.	`newpassword=testpassword`, `password=test`	1. Envoyer une requête PATCH à `/api/newpassword`. 2. Vérifier que la réponse est en JSON et a un statut 200.	Mot de passe changé, réponse avec JSON et statut 200.
12	**DELETE /api/annonces/ (invalid)**	Vérifier la suppression d'une annonce avec un token invalide échoue.	Intégration	Utiliser un token non valide.	`Authorization: Bearer mauvaisToken`	1. Envoyer une requête DELETE à `/api/annonces/27`. 2. Vérifier que la réponse a un statut 401.	Échec de la suppression, statut 401 non autorisé.
13	**userController.login - valid**	Vérifier que la connexion réussit avec les informations valides.	Test unitaire	L'utilisateur doit exister avec des informations valides.	`email=test@example.com`, `password=password123`	1. Simuler la requête avec les informations valides. 2. Appeler `userController.login`. 3. Vérifier que les tokens et les informations utilisateur sont retournés.	Connexion réussie, tokens et infos utilisateur retournés.
14	**userController.login - email not found**	Vérifier que la connexion échoue si l'email n'est pas trouvé.	Test unitaire	Aucun utilisateur avec l'email fourni n'existe.	`email=test@example.com`, `password=password123`	1. Simuler la requête avec un email non existant. 2. Appeler `userController.login`. 3. Vérifier que le statut 401 est retourné avec un message d'erreur.	Échec de la connexion, statut 401 et message d'erreur retourné.
15	**userController.login - password incorrect**	Vérifier que la connexion échoue si le mot de passe est incorrect.	Test unitaire	L'utilisateur doit exister mais avec un mot de passe différent.	`email=test@example.com`, `password=password123`	1. Simuler la requête avec un mot de passe incorrect. 2. Appeler `userController.login`. 3. Vérifier que le statut 401 est	Échec de la connexion, statut 401 et message d'erreur retourné.

						retourné avec un message d'erreur.	
16	**userController .login - handle errors**	Vérifier que les erreurs sont gérées correctement et que le statut 500 est retourné en cas d'erreur.	Test unitaire	Une erreur doit se produire dans la requête de connexion.	`email=test@example.com`, `password=password123`	1. Simuler une erreur de base de données. 2. Appeler `userController.login`. 3. Vérifier que le statut 500 est retourné avec un message d'erreur.	Erreur gérée correctement, statut 500 et message d'erreur retourné.
17	**userController .register - valid**	Vérifier que l'enregistrement réussit avec des informations valides.	Test unitaire	Les informations de l'utilisateur doivent être valides.	`email=newuser@example.com`, `password=password123`, `pseudo=newuser`, `birthdate=2000-01-01`, `role_id=2`	1. Simuler la requête avec les informations valides. 2. Appeler `userController.register`. 3. Vérifier que les tokens et les informations utilisateur sont retournés.	Enregistrement réussi, tokens et infos utilisateur retournés.
18	**userController .register - could not register**	Vérifier que l'enregistrement échoue si l'utilisateur ne peut pas être ajouté.	Test unitaire	Une erreur de base de données se produit lors de l'enregistrement.	`email=newuser@example.com`, `password=password123`, `pseudo=newuser`, `birthdate=2000-01-01`, `role_id=2`	1. Simuler une erreur de base de données lors de l'enregistrement. 2. Appeler `userController.register`. 3. Vérifier que le statut 404 est retourné avec un message d'erreur.	Échec de l'enregistrement, statut 404 et message d'erreur retourné.
19	**userController .register - handle errors**	Vérifier que les erreurs sont gérées correctement et que le statut 500 est retourné en cas d'erreur.	Test unitaire	Une erreur doit se produire dans la requête d'enregistrement.	`email=newuser@example.com`, `password=password123`, `pseudo=newuser`, `birthdate=2000-01-01`, `role_id=2`	1. Simuler une erreur lors du hachage du mot de passe. 2. Appeler `userController.register`. 3. Vérifier que le statut 500 est retourné avec	Erreur gérée correctement, statut 500 et message d'erreur retourné.

						un message d'erreur.	
20	**userController .getUserProfil - valid**	Vérifier que le profil de l'utilisateur est retourné avec succès.	Test unitaire	L'utilisateur doit exister dans le système.	`userId=1`	1. Simuler la requête pour récupérer le profil de l'utilisateur. 2. Appeler `userController.getUserProfil`. 3. Vérifier que le profil utilisateur est retourné avec succès.	Profil utilisateur retourné avec succès, réponse JSON.
21	**userController .getUserProfil - handle errors**	Vérifier que les erreurs sont gérées correctement et que le statut 500 est retourné en cas d'erreur.	Test unitaire	Une erreur doit se produire lors de la récupération du profil.	`userId=1`	1. Simuler une erreur de base de données lors de la récupération du profil. 2. Appeler `userController.getUserProfil`. 3. Vérifier que le statut 500 est retourné avec un message d'erreur.	Erreur gérée correctement , statut 500 et message d'erreur retourné.
22	**userController .delete - valid**	Vérifier que la suppression de l'utilisateur réussit si l'utilisateur est trouvé.	Test unitaire	L'utilisateur doit exister et être autorisé à supprimer.	`userId=1`	1. Simuler la requête de suppression avec un utilisateur existant. 2. Appeler `userController.delete`. 3. Vérifier que l'utilisateur est supprimé avec succès et que le statut 200 est retourné.	Utilisateur supprimé avec succès, statut 200.

23	**userController .delete - not found**	Vérifier que la suppression échoue si l'utilisateur ne peut pas être supprimé.	Test unitaire	L'utilisateur doit exister mais ne pas pouvoir être supprimé.	`userId=1`	1. Simuler la requête de suppression avec un utilisateur non supprimable. 2. Appeler `userController.delete`. 3. Vérifier que le statut 404 est retourné avec un message d'erreur.	Échec de la suppression, statut 404 et message d'erreur retourné.
24	**userController .delete - unauthorized**	Vérifier que la suppression échoue si l'utilisateur n'est pas autorisé à supprimer.	Test unitaire	L'utilisateur doit exister mais ne pas être autorisé à supprimer.	`userId=1`	1. Simuler la requête de suppression avec un utilisateur non autorisé. 2. Appeler `userController.delete`. 3. Vérifier que le statut 401 est retourné avec un message d'erreur.	Échec de la suppression, statut 401 et message d'erreur retourné.
25	**userController .delete - handle errors**	Vérifier que les erreurs sont gérées correctement et que le statut 500 est retourné en cas d'erreur.	Test unitaire	Une erreur doit se produire lors de la suppression de l'utilisateur.	`userId=1`	1. Simuler une erreur de base de données lors de la suppression. 2. Appeler `userController.delete`. 3. Vérifier que le statut 500 est retourné avec un message d'erreur.	Erreur gérée correctement , statut 500 et message d'erreur retourné.
26	**userController .resetPasswor d - valid**	Vérifier que l'email de réinitialisation de mot de passe est envoyé si l'utilisateur est trouvé.	Test unitaire	L'utilisateur doit exister dans le système.	`email=test@ex ample.com`	1. Simuler la requête de réinitialisation de mot de passe. 2. Appeler `userController.resetPasswo rd`.	Email de réinitialisation envoyé avec succès, statut 200.

						3. Vérifier que l'email de réinitialisation est envoyé et que le statut 200 est retourné.	
27	**userController.resetPassword - not found**	Vérifier que la réinitialisation échoue si l'utilisateur n'est pas trouvé.	Test unitaire	Aucun utilisateur avec l'email fourni n'existe.	`email=test@example.com`	1. Simuler la requête de réinitialisation de mot de passe avec un email non existant. 2. Appeler `userController.resetPassword`. 3. Vérifier que le statut 404 est retourné avec un message d'erreur.	Échec de la réinitialisation, statut 404 et message d'erreur retourné.
28	**userController.resetPassword - handle errors**	Vérifier que les erreurs sont gérées correctement et que le statut 500 est retourné en cas d'erreur.	Test unitaire	Une erreur doit se produire lors de la réinitialisation de mot de passe.	`email=test@example.com`	1. Simuler une erreur lors de l'envoi de l'email de réinitialisation. 2. Appeler `userController.resetPassword`. 3. Vérifier que le statut 500 est retourné avec un message d'erreur.	Erreur gérée correctement, statut 500 et message d'erreur retourné.
29	**userController.setNewPassword - valid**	Vérifier que la mise à jour du mot de passe réussit si les informations sont valides.	Test unitaire	L'utilisateur doit être connecté et fournir un nouveau mot de passe valide.	`email=test@example.com`, `newPassword=newpassword123`	1. Simuler la requête de mise à jour du mot de passe. 2. Appeler `userController.setNewPassword`. 3. Vérifier que le mot de passe est mis à jour et que le statut 200 est retourné.	Mot de passe mis à jour avec succès, statut 200.

30	**userController .setNewPassword - not updated**	Vérifier que la mise à jour échoue si le mot de passe ne peut pas être mis à jour.	Test unitaire	L'utilisateur doit être connecté mais le mot de passe ne peut pas être mis à jour.	`email=test@example.com`, `newPassword=newpassword123`	1. Simuler une erreur de mise à jour du mot de passe. 2. Appeler `userController.setNewPassword`. 3. Vérifier que le statut 404 est retourné avec un message d'erreur.	Échec de la mise à jour, statut 404 et message d'erreur retourné.
31	**userController .setNewPassword - handle errors**	Vérifier que les erreurs sont gérées correctement et que le statut 500 est retourné en cas d'erreur.	Test unitaire	Une erreur doit se produire lors de la mise à jour du mot de passe.	`email=test@example.com`, `newPassword=newpassword123`	1. Simuler une erreur lors du hachage du mot de passe. 2. Appeler `userController.setNewPassword`. 3. Vérifier que le statut 500 est retourné avec un message d'erreur.	Erreur gérée correctement, statut 500 et message d'erreur retourné.
32	**userController .getAllUsers - valid**	Vérifier que tous les utilisateurs sont retournés avec succès.	Test unitaire	Les utilisateurs doivent exister dans le système.	Aucune	1. Simuler la requête pour récupérer tous les utilisateurs. 2. Appeler `userController.getAllUsers`. 3. Vérifier que la liste des utilisateurs est retournée avec succès.	Liste des utilisateurs retournée avec succès, statut 200.
33	**userController .getAllUsers - not found**	Vérifier que la récupération échoue si aucun utilisateur n'est trouvé.	Test unitaire	Aucun utilisateur n'existe dans le système.	Aucune	1. Simuler la requête pour récupérer tous les utilisateurs avec une base de données vide. 2. Appeler `userController.getAllUsers`. 3. Vérifier que le statut 404 est retourné avec	Échec de la récupération, statut 404 et message d'erreur retourné.

						un message d'erreur.	
34	**userController .getAllUsers - handle errors**	Vérifier que les erreurs sont gérées correctement et que le statut 500 est retourné en cas d'erreur.	Test unitaire	Une erreur doit se produire lors de la récupération des utilisateurs.	Aucune	1. Simuler une erreur de base de données lors de la récupération des utilisateurs. 2. Appeler `userController.getAllUsers`. 3. Vérifier que le statut 500 est retourné avec un message d'erreur.	Erreur gérée correctement, statut 500 et message d'erreur retourné.
35	**userController .edit - valid**	Vérifier que la modification d'un utilisateur réussit si les informations sont valides.	Test unitaire	L'utilisateur doit exister et être autorisé à modifier.	`userId=1`, `pseudo=updateduser`	1. Simuler la requête de modification avec un utilisateur existant. 2. Appeler `userController.edit`. 3. Vérifier que l'utilisateur est modifié avec succès et que le statut 200 est retourné.	Utilisateur modifié avec succès, statut 200.
36	**userController .edit - not found**	Vérifier que la modification échoue si l'utilisateur ne peut pas être modifié.	Test unitaire	L'utilisateur doit exister mais ne pas pouvoir être modifié.	`userId=1`, `pseudo=updateduser`	1. Simuler la requête de modification avec un utilisateur non modifiable. 2. Appeler `userController.edit`. 3. Vérifier que le statut 404 est retourné avec un message d'erreur.	Échec de la modification, statut 404 et message d'erreur retourné.

37	**userController .edit - unauthorized**	Vérifier que la modification échoue si l'utilisateur n'est pas autorisé à modifier.	Test unitaire	L'utilisateur doit exister mais ne pas être autorisé à modifier.	`userId=1`, `pseudo=updateduser`	1. Simuler la requête de modification avec un utilisateur non autorisé. 2. Appeler `userController.edit`. 3. Vérifier que le statut 401 est retourné avec un message d'erreur.	Échec de la modification, statut 401 et message d'erreur retourné.
38	**userController .edit - handle errors**	Vérifier que les erreurs sont gérées correctement et que le statut 500 est retourné en cas d'erreur.	Test unitaire	Une erreur doit se produire lors de la modification de l'utilisateur.	`userId=1`, `pseudo=updateduser`	1. Simuler une erreur de base de données lors de la modification. 2. Appeler `userController.edit`. 3. Vérifier que le statut 500 est retourné avec un message d'erreur.	Erreur gérée correctement, statut 500 et message d'erreur retourné.
39	**userController .contactForm - valid**	Vérifier que le formulaire de contact envoie un email avec succès.	Test unitaire	Aucune	`name=Test User`, `email=test@example.com`, `message=Hello`	1. Simuler la requête de formulaire de contact. 2. Appeler `userController.contactForm`. 3. Vérifier que l'email est envoyé et que le statut 200 est retourné avec un message de succès.	Email envoyé avec succès, statut 200 et message de succès retourné.
40	**userController .contactForm - email error**	Vérifier que le formulaire de contact échoue si l'email ne peut pas être envoyé.	Test unitaire	L'email ne peut pas être envoyé pour une raison quelconque.	`name=Test User`, `email=test@example.com`, `message=Hello`	1. Simuler une erreur d'envoi d'email lors du formulaire de contact. 2. Appeler `userController.contactForm`. 3. Vérifier que le statut 500 est	Échec de l'envoi d'email, statut 500 et message d'erreur retourné.

						retourné avec un message d'erreur.	
41	**userController .getAllAvatars - valid**	Vérifier que tous les avatars sont retournés avec succès.	Test unitaire	Les avatars doivent exister dans le système.	Aucune	1. Simuler la requête pour récupérer tous les avatars. 2. Appeler `userController.getAllAvatars`. 3. Vérifier que la liste des avatars est retournée avec succès.	Liste des avatars retournée avec succès, statut 200.
42	**userController .getAllAvatars - not found**	Vérifier que la récupération échoue si aucun avatar n'est trouvé.	Test unitaire	Aucun avatar n'existe dans le système.	Aucune	1. Simuler la requête pour récupérer tous les avatars avec une base de données vide. 2. Appeler `userController.getAllAvatars`. 3. Vérifier que le statut 404 est retourné avec un message d'erreur.	Échec de la récupération, statut 404 et message d'erreur retourné.
43	**userController .getAllAvatars - handle errors**	Vérifier que les erreurs sont gérées correctement et que le statut 500 est retourné en cas d'erreur.	Test unitaire	Une erreur doit se produire lors de la récupération des avatars.	Aucune	1. Simuler une erreur de base de données lors de la récupération des avatars. 2. Appeler `userController.getAllAvatars`. 3. Vérifier que le statut 500 est retourné avec un message d'erreur.	Erreur gérée correctement , statut 500 et message d'erreur retourné.

44	**adController.getAll - valid**	Vérifier que toutes les annonces sont retournées avec succès.	Test unitaire	Les annonces doivent exister dans le système.	Aucune	1. Simuler la requête pour récupérer toutes les annonces. 2. Appeler `adController.getAll`. 3. Vérifier que la liste des annonces est retournée avec succès.	Liste des annonces retournée avec succès, statut 200.
45	**adController.getAll - not found**	Vérifier que la récupération échoue si aucune annonce n'est trouvée.	Test unitaire	Aucune annonce n'existe dans le système.	Aucune	1. Simuler la requête pour récupérer toutes les annonces avec une base de données vide. 2. Appeler `adController.getAll`. 3. Vérifier que le statut 404 est retourné avec un message d'erreur.	Échec de la récupération, statut 404 et message d'erreur retourné.
46	**adController.getAll - handle errors**	Vérifier que les erreurs sont gérées correctement et que le statut 500 est retourné en cas d'erreur.	Test unitaire	Une erreur doit se produire lors de la récupération des annonces.	Aucune	1. Simuler une erreur de base de données lors de la récupération des annonces. 2. Appeler `adController.getAll`. 3. Vérifier que le statut 500 est retourné avec un message d'erreur.	Erreur gérée correctement, statut 500 et message d'erreur retourné.
47	**adController.getAllByCategory - valid**	Vérifier que toutes les annonces pour une catégorie sont retournées avec succès.	Test unitaire	Les annonces doivent exister pour une catégorie spécifique.	`category_id=1`	1. Simuler la requête pour récupérer les annonces par catégorie. 2. Appeler `adController.getAllByCategory`. 3. Vérifier que la liste des annonces est retournée avec succès.	Liste des annonces par catégorie retournée avec succès, statut 200.

48	**adController.getAllByCategory - not found**	Vérifier que la récupération échoue si aucune annonce n'est trouvée pour une catégorie.	Test unitaire	Aucune annonce n'existe pour la catégorie spécifiée.	`category_id=1`	1. Simuler la requête pour récupérer les annonces par catégorie avec une base de données vide. 2. Appeler `adController.getAllByCategory`. 3. Vérifier que le statut 404 est retourné avec un message d'erreur.	Échec de la récupération, statut 404 et message d'erreur retourné.
49	**adController.getAllByCategory - handle errors**	Vérifier que les erreurs sont gérées correctement et que le statut 500 est retourné en cas d'erreur.	Test unitaire	Une erreur doit se produire lors de la récupération des annonces par catégorie.	`category_id=1`	1. Simuler une erreur de base de données lors de la récupération des annonces par catégorie. 2. Appeler `adController.getAllByCategory`. 3. Vérifier que le statut 500 est retourné avec un message d'erreur.	Erreur gérée correctement, statut 500 et message d'erreur retourné.
50	**adController.getUserAds - valid**	Vérifier que toutes les annonces pour un utilisateur sont retournées avec succès.	Test unitaire	Les annonces doivent exister pour un utilisateur spécifique.	`user_id=1`	1. Simuler la requête pour récupérer les annonces d'un utilisateur. 2. Appeler `adController.getUserAds`. 3. Vérifier que la liste des annonces est retournée avec succès.	Liste des annonces par utilisateur retournée avec succès, statut 200.
51	**adController.getUserAds - not found**	Vérifier que la récupération échoue si aucune annonce n'est trouvée pour un utilisateur.	Test unitaire	Aucune annonce n'existe pour l'utilisateur spécifié.	`user_id=1`	1. Simuler la requête pour récupérer les annonces d'un utilisateur avec une base de données vide. 2. Appeler `adController.getUserAds`.	Échec de la récupération, statut 404 et message d'erreur retourné.

						3. Vérifier que le statut 404 est retourné avec un message d'erreur.	
52	**adController.getUserAds - handle errors**	Vérifier que les erreurs sont gérées correctement et que le statut 500 est retourné en cas d'erreur.	Test unitaire	Une erreur doit se produire lors de la récupération des annonces d'un utilisateur.	`user_id=1`	1. Simuler une erreur de base de données lors de la récupération des annonces d'un utilisateur. 2. Appeler `adController.getUserAds`. 3. Vérifier que le statut 500 est retourné avec un message d'erreur.	Erreur gérée correctement, statut 500 et message d'erreur retourné.
53	**adController.getAllByUser - valid**	Vérifier que toutes les annonces pour un utilisateur sont retournées avec succès.	Test unitaire	Les annonces doivent exister pour un utilisateur spécifique.	`user_id=1`	1. Simuler la requête pour récupérer les annonces d'un utilisateur. 2. Appeler `adController.getAllByUser`. 3. Vérifier que la liste des annonces est retournée avec succès.	Liste des annonces par utilisateur retournée avec succès, statut 200.
54	**adController.getAllByUser - not found**	Vérifier que la récupération échoue si aucune annonce n'est trouvée pour un utilisateur.	Test unitaire	Aucune annonce n'existe pour l'utilisateur spécifié.	`user_id=1`	1. Simuler la requête pour récupérer les annonces d'un utilisateur avec une base de données vide. 2. Appeler `adController.getAllByUser`. 3. Vérifier que le statut 404 est retourné avec un message d'erreur.	Échec de la récupération, statut 404 et message d'erreur retourné.

55	**adController.getAllByUser - handle errors**	Vérifier que les erreurs sont gérées correctement et que le statut 500 est retourné en cas d'erreur.	Test unitaire	Une erreur doit se produire lors de la récupération des annonces d'un utilisateur.	`user_id=1`	1. Simuler une erreur de base de données lors de la récupération des annonces d'un utilisateur. 2. Appeler `adController.getAllByUser`. 3. Vérifier que le statut 500 est retourné avec un message d'erreur.	Erreur gérée correctement, statut 500 et message d'erreur retourné.
56	**adController.createUserAd - valid**	Vérifier que la création d'une annonce réussit avec des informations valides.	Test unitaire	Les informations de l'annonce doivent être valides.	`title=New Test Ad`, `userId=1`	1. Simuler la requête de création d'une annonce. 2. Appeler `adController.createUserAd`. 3. Vérifier que l'annonce est créée avec succès et que le statut 200 est retourné.	Annonce créée avec succès, statut 200.
57	**adController.createUserAd - could not create**	Vérifier que la création échoue si l'annonce ne peut pas être ajoutée.	Test unitaire	Une erreur de base de données se produit lors de la création de l'annonce.	`title=New Test Ad`, `userId=1`	1. Simuler une erreur de base de données lors de la création d'une annonce. 2. Appeler `adController.createUserAd`. 3. Vérifier que le statut 404 est retourné avec un message d'erreur.	Échec de la création, statut 404 et message d'erreur retourné.
58	**adController.createUserAd - handle errors**	Vérifier que les erreurs sont gérées correctement et que le statut 500 est retourné en cas d'erreur.	Test unitaire	Une erreur doit se produire lors de la création de l'annonce.	`title=New Test Ad`, `userId=1`	1. Simuler une erreur lors de la création de l'annonce. 2. Appeler `adController.createUserAd`. 3. Vérifier que le statut 500 est retourné avec un message d'erreur.	Erreur gérée correctement, statut 500 et message d'erreur retourné.

59	**adController.getAllByType - valid**	Vérifier que toutes les annonces pour un type sont retournées avec succès.	Test unitaire	Les annonces doivent exister pour un type spécifique.	`type_id=1`	1. Simuler la requête pour récupérer les annonces par type. 2. Appeler `adController.getAllByType`. 3. Vérifier que la liste des annonces est retournée avec succès.	Liste des annonces par type retournée avec succès, statut 200.
60	**adController.getAllByType - not found**	Vérifier que la récupération échoue si aucune annonce n'est trouvée pour un type.	Test unitaire	Aucune annonce n'existe pour le type spécifié.	`type_id=1`	1. Simuler la requête pour récupérer les annonces par type avec une base de données vide. 2. Appeler `adController.getAllByType`. 3. Vérifier que le statut 404 est retourné avec un message d'erreur.	Échec de la récupération, statut 404 et message d'erreur retourné.
61	**adController.getAllByType - handle errors**	Vérifier que les erreurs sont gérées correctement et que le statut 500 est retourné en cas d'erreur.	Test unitaire	Une erreur doit se produire lors de la récupération des annonces par type.	`type_id=1`	1. Simuler une erreur de base de données lors de la récupération des annonces par type. 2. Appeler `adController.getAllByType`. 3. Vérifier que le statut 500 est retourné avec un message d'erreur.	Erreur gérée correctement, statut 500 et message d'erreur retourné.
62	**adController.getOneWithSimilar - valid**	Vérifier que l'annonce et les annonces similaires sont retournées avec succès.	Test unitaire	L'annonce doit exister avec des annonces similaires.	`id=1`	1. Simuler la requête pour récupérer une annonce avec ses similaires. 2. Appeler `adController.getOneWithSimilar`. 3. Vérifier que l'annonce et ses	Annonce et similaires retournés avec succès, statut 200.

						similaires sont retournés avec succès.	
63	**adController.getOneWithSimilar - not found**	Vérifier que la récupération échoue si l'annonce n'est pas trouvée.	Test unitaire	Aucune annonce n'existe avec les critères spécifiés.	`id=1`	1. Simuler la requête pour récupérer une annonce non existante. 2. Appeler `adController.getOneWithSimilar`. 3. Vérifier que le statut 500 est retourné avec un message d'erreur.	Échec de la récupération, statut 500 et message d'erreur retourné.
64	**adController.getOneWithSimilar - handle errors**	Vérifier que les erreurs sont gérées correctement et que le statut 500 est retourné en cas d'erreur.	Test unitaire	Une erreur doit se produire lors de la récupération d'une annonce avec ses similaires.	`id=1`	1. Simuler une erreur lors de la récupération d'une annonce avec ses similaires. 2. Appeler `adController.getOneWithSimilar`. 3. Vérifier que le statut 500 est retourné avec un message d'erreur.	Erreur gérée correctement, statut 500 et message d'erreur retourné.
65	**adController.getAllByTypeAndCategory - valid**	Vérifier que toutes les annonces pour un type et une catégorie sont retournées avec succès.	Test unitaire	Les annonces doivent exister pour un type et une catégorie spécifiques.	`type_id=1`, `category_id=2`	1. Simuler la requête pour récupérer les annonces par type et catégorie. 2. Appeler `adController.getAllByTypeAndCategory`. 3. Vérifier que la liste des annonces est retournée avec succès.	Liste des annonces par type et catégorie retournée avec succès, statut 200.

66	**adController.getAllByTypeAndCategory - not found**	Vérifier que la récupération échoue si aucune annonce n'est trouvée pour un type et une catégorie.	Test unitaire	Aucune annonce n'existe pour le type et la catégorie spécifiés.	`type_id=1`, `category_id=2`	1. Simuler la requête pour récupérer les annonces par type et catégorie avec une base de données vide. 2. Appeler `adController.getAllByTypeAndCategory`. 3. Vérifier que le statut 404 est retourné avec un message d'erreur.	Échec de la récupération, statut 404 et message d'erreur retourné.
67	**adController.getAllByTypeAndCategory - handle errors**	Vérifier que les erreurs sont gérées correctement et que le statut 500 est retourné en cas d'erreur.	Test unitaire	Une erreur doit se produire lors de la récupération des annonces par type et catégorie.	`type_id=1`, `category_id=2`	1. Simuler une erreur de base de données lors de la récupération des annonces par type et catégorie. 2. Appeler `adController.getAllByTypeAndCategory`. 3. Vérifier que le statut 500 est retourné avec un message d'erreur.	Erreur gérée correctement, statut 500 et message d'erreur retourné.
68	**adController.delete - valid**	Vérifier que la suppression d'une annonce réussit si elle est trouvée et l'utilisateur est autorisé.	Test unitaire	L'annonce doit exister et l'utilisateur doit être autorisé à supprimer.	`id=1`, `userId=1`	1. Simuler la requête de suppression avec une annonce existante et un utilisateur autorisé. 2. Appeler `adController.delete`. 3. Vérifier que l'annonce est supprimée avec succès et que le statut 200 est retourné.	Annonce supprimée avec succès, statut 200.

69	**adController.delete - not found**	Vérifier que la suppression échoue si l'annonce ne peut pas être supprimée.	Test unitaire	L'annonce doit exister mais ne pas pouvoir être supprimée.	`id=1`, `userId=1`	1. Simuler la requête de suppression avec une annonce non supprimable. 2. Appeler `adController.delete`. 3. Vérifier que le statut 404 est retourné avec un message d'erreur.	Échec de la suppression, statut 404 et message d'erreur retourné.
70	**adController.delete - unauthorized**	Vérifier que la suppression échoue si l'utilisateur n'est pas autorisé à supprimer l'annonce.	Test unitaire	L'annonce doit exister mais l'utilisateur ne doit pas être autorisé à supprimer.	`id=1`, `userId=1`	1. Simuler la requête de suppression avec un utilisateur non autorisé. 2. Appeler `adController.delete`. 3. Vérifier que le statut 401 est retourné avec un message d'erreur.	Échec de la suppression, statut 401 et message d'erreur retourné.
71	**adController.delete - handle errors**	Vérifier que les erreurs sont gérées correctement et que le statut 500 est retourné en cas d'erreur.	Test unitaire	Une erreur doit se produire lors de la suppression de l'annonce.	`id=1`, `userId=1`	1. Simuler une erreur de base de données lors de la suppression. 2. Appeler `adController.delete`. 3. Vérifier que le statut 500 est retourné avec un message d'erreur.	Erreur gérée correctement, statut 500 et message d'erreur retourné.
72	**adController.edit - valid**	Vérifier que la modification d'une annonce réussit si elle est trouvée et l'utilisateur est autorisé.	Test unitaire	L'annonce doit exister et l'utilisateur doit être autorisé à modifier.	`id=1`, `userId=1`, `title=Edited Ad`	1. Simuler la requête de modification avec une annonce existante et un utilisateur autorisé. 2. Appeler `adController.edit`. 3. Vérifier que l'annonce est modifiée avec succès et que le statut 200 est retourné.	Annonce modifiée avec succès, statut 200.

73	**adController.edit - not found**	Vérifier que la modification échoue si l'annonce ne peut pas être modifiée.	Test unitaire	L'annonce doit exister mais ne pas pouvoir être modifiée.	`id=1`, `userId=1`, `title=Edited Ad`	1. Simuler la requête de modification avec une annonce non modifiable. 2. Appeler `adController.edit`. 3. Vérifier que le statut 404 est retourné avec un message d'erreur.	Échec de la modification, statut 404 et message d'erreur retourné.
74	**adController.edit - unauthorized**	Vérifier que la modification échoue si l'utilisateur n'est pas autorisé à modifier l'annonce.	Test unitaire	L'annonce doit exister mais l'utilisateur ne doit pas être autorisé à modifier.	`id=1`, `userId=1`, `title=Edited Ad`	1. Simuler la requête de modification avec un utilisateur non autorisé. 2. Appeler `adController.edit`. 3. Vérifier que le statut 401 est retourné avec un message d'erreur.	Échec de la modification, statut 401 et message d'erreur retourné.
75	**adController.edit - handle errors**	Vérifier que les erreurs sont gérées correctement et que le statut 500 est retourné en cas d'erreur.	Test unitaire	Une erreur doit se produire lors de la modification de l'annonce.	`id=1`, `userId=1`, `title=Edited Ad`	1. Simuler une erreur de base de données lors de la modification. 2. Appeler `adController.edit`. 3. Vérifier que le statut 500 est retourné avec un message d'erreur.	Erreur gérée correctement, statut 500 et message d'erreur retourné.
76	**categoryController.getAll - valid**	Vérifier que toutes les catégories sont retournées avec succès.	Test unitaire	Les catégories doivent exister dans le système.	Aucune	1. Simuler la requête pour récupérer toutes les catégories. 2. Appeler `categoryController.getAll`. 3. Vérifier que la liste des catégories est retournée avec succès.	Liste des catégories retournée avec succès, statut 200.

77	**categoryController.getAll - not found**	Vérifier que la récupération échoue si aucune catégorie n'est trouvée.	Test unitaire	Aucune catégorie n'existe dans le système.	Aucune	1. Simuler la requête pour récupérer toutes les catégories avec une base de données vide. 2. Appeler `categoryController.getAll`. 3. Vérifier que le statut 404 est retourné avec un message d'erreur.	Échec de la récupération, statut 404 et message d'erreur retourné.
78	**categoryController.getAll - handle errors**	Vérifier que les erreurs sont gérées correctement et que le statut 500 est retourné en cas d'erreur.	Test unitaire	Une erreur doit se produire lors de la récupération des catégories.	Aucune	1. Simuler une erreur de base de données lors de la récupération des catégories. 2. Appeler `categoryController.getAll`. 3. Vérifier que le statut 500 est retourné avec un message d'erreur.	Erreur gérée correctement, statut 500 et message d'erreur retourné.
79	**categoryController.edit - valid**	Vérifier que la modification d'une catégorie réussit si elle est trouvée.	Test unitaire	La catégorie doit exister dans le système.	`id=1`, `name=Edited Category`	1. Simuler la requête de modification d'une catégorie. 2. Appeler `categoryController.edit`. 3. Vérifier que la catégorie est modifiée avec succès et que le statut 200 est retourné.	Catégorie modifiée avec succès, statut 200.
80	**categoryController.edit - not found**	Vérifier que la modification échoue si la catégorie ne peut pas être modifiée.	Test unitaire	La catégorie doit exister mais ne pas pouvoir être modifiée.	`id=1`	1. Simuler la requête de modification d'une catégorie non modifiable. 2. Appeler `categoryController.edit`. 3. Vérifier que le statut 404 est retourné avec un message d'erreur.	Échec de la modification, statut 404 et message d'erreur retourné.

81	**categoryController.edit - handle errors**	Vérifier que les erreurs sont gérées correctement et que le statut 500 est retourné en cas d'erreur.	Test unitaire	Une erreur doit se produire lors de la modification de la catégorie.	`id=1`, `name=Edited Category`	1. Simuler une erreur de base de données lors de la modification. 2. Appeler `categoryController.edit`. 3. Vérifier que le statut 500 est retourné avec un message d'erreur.	Erreur gérée correctement, statut 500 et message d'erreur retourné.
82	**categoryController.delete - valid**	Vérifier que la suppression d'une catégorie réussit si elle est trouvée.	Test unitaire	La catégorie doit exister dans le système.	`id=1`	1. Simuler la requête de suppression d'une catégorie. 2. Appeler `categoryController.delete`. 3. Vérifier que la catégorie est supprimée avec succès et que le statut 200 est retourné.	Catégorie supprimée avec succès, statut 200.
83	**categoryController.delete - not found**	Vérifier que la suppression échoue si la catégorie ne peut pas être supprimée.	Test unitaire	La catégorie doit exister mais ne pas pouvoir être supprimée.	`id=1`	1. Simuler la requête de suppression d'une catégorie non supprimable. 2. Appeler `categoryController.delete`. 3. Vérifier que le statut 404 est retourné avec un message d'erreur.	Échec de la suppression, statut 404 et message d'erreur retourné.
84	**categoryController.delete - handle errors**	Vérifier que les erreurs sont gérées correctement et que le statut 500 est retourné en cas d'erreur.	Test unitaire	Une erreur doit se produire lors de la suppression de la catégorie.	`id=1`	1. Simuler une erreur de base de données lors de la suppression. 2. Appeler `categoryController.delete`. 3. Vérifier que le statut 500 est retourné avec un message d'erreur.	Erreur gérée correctement, statut 500 et message d'erreur retourné.

85	**Signup Page - display all basic elements**	Vérifier que tous les éléments de base de la page d'inscription sont affichés correctement.	E2E	La page d'inscription doit être accessible.	Aucune	1. Accéder à la page d'inscription. 2. Vérifier la présence des titres et des champs de formulaire. 3. Vérifier que le bouton d'inscription est affiché correctement.	Tous les éléments de base de la page d'inscription sont affichés correctement.
86	**Signup Page - email validation**	Vérifier que le formulaire affiche une erreur lorsque l'email est invalide.	E2E	La page d'inscription doit être accessible.	`email=invalid@email`, `password=validPassword123!`, `passwordConfirm=validPassword123!`, `username=testuser`, `birthdate=2000-01-01`	1. Accéder à la page d'inscription. 2. Remplir le formulaire avec un email invalide et d'autres champs valides. 3. Soumettre le formulaire. 4. Vérifier qu'une erreur "Email non valide" est affichée.	Une erreur "Email non valide" est affichée lorsque l'email est invalide.
87	**Signup Page - password matching**	Vérifier que le formulaire affiche une erreur lorsque les mots de passe ne correspondent pas.	E2E	La page d'inscription doit être accessible.	`email=test@example.com`, `password=validPassword123!`, `passwordConfirm=differentPassword!`, `username=testuser`, `birthdate=2000-01-01`	1. Accéder à la page d'inscription. 2. Remplir le formulaire avec des mots de passe qui ne correspondent pas. 3. Soumettre le formulaire. 4. Vérifier qu'une erreur "Les mots de passe ne sont pas identiques" est affichée.	Une erreur "Les mots de passe ne sont pas identiques" est affichée lorsque les mots de passe ne correspondent pas.

88	**Signup Page - password visibility toggle**	Vérifier que la visibilité des mots de passe peut être basculée.	E2E	La page d'inscription doit être accessible.	Aucune	1. Accéder à la page d'inscription. 2. Vérifier que les champs de mot de passe sont de type "password". 3. Cliquer sur l'icône de bascule de la visibilité des mots de passe. 4. Vérifier que les champs de mot de passe sont de type "text". 5. Cliquer de nouveau sur l'icône. 6. Vérifier que les champs de mot de passe reviennent au type "password".	La visibilité des mots de passe peut être basculée entre "password" et "text".
89	**Signup Page - loading indicator**	Vérifier qu'un indicateur de chargement est affiché lors de la soumission du formulaire.	E2E	La page d'inscription doit être accessible.	`email=validEmailExemple@example.com`, `password=validPassword123!Exemple`, `passwordConfirm=validPassword123!Exemple`, `username=testuser`, `birthdate=2000-01-01`	1. Accéder à la page d'inscription. 2. Remplir le formulaire avec des informations valides. 3. Soumettre le formulaire. 4. Vérifier qu'un indicateur de chargement "Chargement en cours..." est affiché.	Un indicateur de chargement est affiché pendant le processus de soumission du formulaire.

90	**Signup Page - successful signup**	Vérifier que l'utilisateur est redirigé vers la page d'accueil après une inscription réussie.	E2E	La page d'inscription doit être accessible.	`email=validEmailExemple@example.com`, `password=validPassword123!Exemple`, `passwordConfirm=validPassword123!Exemple`, `username=testuser`, `birthdate=2000-01-01`	1. Accéder à la page d'inscription. 2. Remplir le formulaire avec des informations valides. 3. Soumettre le formulaire. 4. Vérifier que l'utilisateur est redirigé vers la page d'accueil.	L'utilisateur est redirigé vers la page d'accueil après une inscription réussie.
91	**Login Page - display all basic elements**	Vérifier que tous les éléments de base de la page de connexion sont affichés correctement.	E2E	La page de connexion doit être accessible.	Aucune	1. Accéder à la page de connexion. 2. Vérifier la présence des titres et des champs de formulaire. 3. Vérifier que le bouton de connexion et le lien de réinitialisation du mot de passe sont affichés correctement.	Tous les éléments de base de la page de connexion sont affichés correctement.
92	**Login Page - email validation**	Vérifier que le formulaire affiche une erreur lorsque l'email est invalide.	E2E	La page de connexion doit être accessible.	`email=emaili@nvalid`, `password=validPassword123!`	1. Accéder à la page de connexion. 2. Remplir le formulaire avec un email invalide et un mot de passe valide. 3. Soumettre le formulaire. 4. Vérifier qu'une erreur est affichée.	Une erreur est affichée lorsque l'email est invalide.

93	**Login Page - password visibility toggle**	Vérifier que la visibilité du mot de passe peut être basculée.	E2E	La page de connexion doit être accessible.	Aucune	1. Accéder à la page de connexion. 2. Vérifier que le champ de mot de passe est de type "password". 3. Cliquer sur l'icône de bascule de la visibilité du mot de passe. 4. Vérifier que le champ de mot de passe est de type "text". 5. Cliquer de nouveau sur l'icône. 6. Vérifier que le champ de mot de passe revient au type "password".	La visibilité du mot de passe peut être basculée entre "password" et "text".
94	**Login Page - password reset link**	Vérifier que l'utilisateur est redirigé vers la page de réinitialisation du mot de passe.	E2E	La page de connexion doit être accessible.	Aucune	1. Accéder à la page de connexion. 2. Cliquer sur le lien de réinitialisation du mot de passe. 3. Vérifier que l'utilisateur est redirigé vers la page de réinitialisation du mot de passe.	L'utilisateur est redirigé vers la page de réinitialisation du mot de passe.

95	**Login Page - loading indicator**	Vérifier qu'un indicateur de chargement est affiché lors de la soumission du formulaire.	E2E	La page de connexion doit être accessible.	`email=alizeamasse@gmail.com`, `password=test`	1. Accéder à la page de connexion. 2. Remplir le formulaire avec des informations valides. 3. Soumettre le formulaire. 4. Vérifier qu'un indicateur de chargement "Chargement en cours..." est affiché.	Un indicateur de chargement est affiché pendant le processus de soumission du formulaire.
96	**Login Page - redirect to registration**	Vérifier que l'utilisateur est redirigé vers la page d'inscription lorsqu'il clique sur le bouton d'inscription.	E2E	La page de connexion doit être accessible.	Aucune	1. Accéder à la page de connexion. 2. Cliquer sur le bouton d'inscription. 3. Vérifier que l'utilisateur est redirigé vers la page d'inscription.	L'utilisateur est redirigé vers la page d'inscription.
97	**Login Page - successful login**	Vérifier que l'utilisateur est redirigé vers la page d'accueil après une connexion réussie.	E2E	La page de connexion doit être accessible.	`email=alizeamasse@gmail.com`, `password=test`	1. Accéder à la page de connexion. 2. Remplir le formulaire avec des informations valides. 3. Soumettre le formulaire. 4. Vérifier que l'utilisateur est redirigé vers la page d'accueil.	L'utilisateur est redirigé vers la page d'accueil après une connexion réussie.

98	**Forgotten Password Page - display form elements**	Vérifier que tous les éléments du formulaire de la page "Mot de passe oublié" sont affichés correctement.	E2E	La page "Mot de passe oublié" doit être accessible.	Aucune	1. Accéder à la page "Mot de passe oublié". 2. Vérifier la présence des éléments du formulaire tels que le titre, le label pour l'email, le champ de saisie de l'email, et le bouton de soumission.	Tous les éléments du formulaire de la page "Mot de passe oublié" sont affichés correctement.
99	**Forgotten Password Page - submit form with valid email**	Vérifier que le formulaire est soumis avec succès lorsqu'un email valide est fourni.	E2E	La page "Mot de passe oublié" doit être accessible.	`email=alizeamasse@gmail.com`	1. Accéder à la page "Mot de passe oublié". 2. Intercepter l'appel API de réinitialisation de mot de passe. 3. Remplir le formulaire avec un email valide et le soumettre. 4. Attendre la réponse de l'API et vérifier que le code de statut est 200. 5. Vérifier qu'un message de confirmation est affiché.	Le formulaire est soumis avec succès, statut 200, et un message de confirmation est affiché.

100	**Forgotten Password Page - display error for invalid email**	Vérifier que le formulaire affiche une erreur lorsque l'email est invalide.	E2E	La page "Mot de passe oublié" doit être accessible.	`email=inval@idemail`	1. Accéder à la page "Mot de passe oublié". 2. Intercepter l'appel API de réinitialisation de mot de passe et forcer une réponse d'erreur. 3. Remplir le formulaire avec un email invalide et le soumettre. 4. Attendre la réponse de l'API. 5. Vérifier qu'un message d'erreur est affiché.	Une erreur est affichée lorsque l'email est invalide, indiquant que l'utilisateur n'a pas été trouvé.
101	**Forgotten Password Page - loading message on submit**	Vérifier qu'un message de chargement est affiché lorsque le formulaire est soumis.	E2E	La page "Mot de passe oublié" doit être accessible.	`email=alizeamasse@gmail.com`	1. Accéder à la page "Mot de passe oublié". 2. Intercepter l'appel API de réinitialisation de mot de passe et retarder la réponse. 3. Remplir le formulaire avec un email valide et le soumettre. 4. Vérifier qu'un message de chargement "Chargement en cours..." est affiché. 5. Attendre la réponse de l'API.	Un message de chargement est affiché pendant le processus de soumission du formulaire.

102	**Add New Post Page - display error for invalid postal code**	Vérifier que le formulaire affiche une erreur lorsque le code postal est invalide.	E2E	La page "Ajouter une Nouvelle Annonce" doit être accessible.	`title=Titre de Test pour une Nouvelle Annonce`, `category=1`, `condition=1`, `postal_code=123`, `description=Description de Test pour une Nouvelle Annonce`	1. Accéder à la page "Ajouter une Nouvelle Annonce". 2. Remplir le formulaire avec un code postal invalide. 3. Soumettre le formulaire. 4. Vérifier qu'un message d'erreur "Code postal invalide" est affiché.	Une erreur "Code postal invalide" est affichée lorsque le code postal est invalide.
103	**Add New Post Page - toggle postal code field visibility**	Vérifier que le champ du code postal est visible ou masqué en fonction de la sélection de la condition.	E2E	La page "Ajouter une Nouvelle Annonce" doit être accessible.	`condition=1`, `condition=2`	1. Accéder à la page "Ajouter une Nouvelle Annonce". 2. Sélectionner la condition "1". 3. Vérifier que le champ du code postal est visible. 4. Sélectionner la condition "2". 5. Vérifier que le champ du code postal n'est pas visible.	Le champ du code postal est visible pour la condition "1" et masqué pour la condition "2".

104	**Add New Post Page - submit form successfully**	Vérifier que l'utilisateur peut remplir et soumettre le formulaire avec succès.	E2E	La page "Ajouter une Nouvelle Annonce" doit être accessible.	`title=Titre de Test pour une Nouvelle Annonce`, `category=1`, `condition=1`, `postal_code=13004`, `description=Description de Test pour une Nouvelle Annonce`	1. Accéder à la page "Ajouter une Nouvelle Annonce". 2. Intercepter la requête POST pour créer une annonce. 3. Remplir le formulaire avec des informations valides. 4. Soumettre le formulaire. 5. Attendre la réponse de la requête POST et vérifier que la nouvelle annonce a été créée avec succès. 6. Vérifier l'URL et rediriger si nécessaire. 7. Valider que l'utilisateur est redirigé vers l'URL correcte de la nouvelle annonce ou gérer la redirection si elle échoue.	Le formulaire est soumis avec succès, l'annonce est créée, et l'utilisateur est redirigé vers la page de la nouvelle annonce.