|  |  |  |
| --- | --- | --- |
|  |  |  |
| *Nom de naissance* |  | Messai |
| *Nom d’usage* |  | Ellouzi |
| *Prénom* |  | sondes |
| *Adresse* |  | 4 rue jean de solier |
|  |  |

|  |  |
| --- | --- |
| **Titre professionnel visé** | |
|  | |
| *Cliquez ici pour entrer l’intitulé du titre professionnel visé.* | |
|  | |
| **Modalité d’accès :** | |
|  | |
|  | Parcours de formation |
|  | Validation des Acquis de l’Expérience (VAE) |
|  | |

|  |  |
| --- | --- |
| **Présentation du dossier** | |
|  | |
|  | |
| Le dossier professionnel (DP) constitue un élément du système de validation du titre professionnel. **Ce titre est délivré par le Ministère chargé de l’emploi.**  Le DP appartient au candidat. Il le conserve, l’actualise durant son parcours et le présente **obligatoirement à chaque session d’examen**.  Pour rédiger le DP, le candidat peut être aidé par un formateur ou par un accompagnateur VAE.  Il est consulté par le jury au moment de la session d’examen.  **Pour prendre sa décision, le jury dispose :**   1. des résultats de la mise en situation professionnelle complétés, éventuellement, du questionnaire professionnel ou de l’entretien professionnel ou de l’entretien technique ou du questionnement à partir de productions. 2. du **Dossier Professionnel** (DP) dans lequel le candidat a consigné les preuves de sa pratique professionnelle. 3. des résultats des évaluations passées en cours de formation lorsque le candidat évalué est issu d’un parcours de formation 4. de l’entretien final (dans le cadre de la session titre).   *[Arrêté du 22 décembre 2015, relatif aux conditions de délivrance des titres professionnels*  *du ministère chargé de l’Emploi]*  **Ce dossier comporte :**   * pour chaque activité-type du titre visé, un à trois exemples de pratique professionnelle ; * un tableau à renseigner si le candidat souhaite porter à la connaissance du jury la détention d’un titre, d’un diplôme, d’un certificat de qualification professionnelle (CQP) ou des attestations de formation ; * une déclaration sur l’honneur à compléter et à signer ; * des documents illustrant la pratique professionnelle du candidat (facultatif) * des annexes, si nécessaire. | |
| *Pour compléter ce dossier, le candidat dispose d’un site web en accès libre sur le site.* | |
|  | [**http://travail-emploi.gouv.fr/titres-professionnels**](http://travail-emploi.gouv.fr/titres-professionnels) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Sommaire** | | | |
| **Exemples de pratique professionnelle** | | |
| **Concevoir et développer des composants d’interface utilisateur en intégrant les recommandations de sécurité** | | **p.** | **5** |
|  |  Maquetter une application p. | p. |  |
|  |  Développer la partie frontend d’une interface web p. | p. |  |
|  |  Développer la partie backend d’une interface web p | p. |  |
|  |  |  |  |
| **Concevoir et développer la persistance des données en intégrant les recommandations de sécurité** | | **p.** | **35** |
|  |  Concevoir une base de données p. | p. |  |
|  |  Mettre en place une base de données p. | p. |  |
|  |  |  |  |
| **Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité** | | **p.** | **49** |
|  |  Concevoir une application UML p. | p. |  |
|  |
|  |  Développer une application mobile Mario Pizza p | p. |  |
|  |  |  |  |
| **Gestion de projet** | | **p.** | **45** |
|  |  Gestion de Projet p. | p. |  |
|  |
|  |
|  |  |  |  |
| **Titres, diplômes, CQP, attestations de formation** *(facultatif)* | | **p.** |  |
| **Déclaration sur l’honneur** | | **p.** |  |
| **Documents illustrant la pratique professionnelle** *(facultatif)* | | **p.** |  |
| **Annexes** *(Si le RC le prévoit)* | | **p.** |  |

**Exemples de pratique**

**professionnelle**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **Activité-type** | **1** | **Concevoir et développer des composants d’interface utilisateur en intégrant les recommandations de sécurité** | | **1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :** | | | | | | |
| 1. Cahier charge et évolution   Bien que la solution actuelle ait été efficace jusqu'à présent, diverses évolutions et améliorations sont désormais nécessaires en raison du contexte de l'entreprise.  Nous avons identifié les fonctionnalités suivantes à développer :   1. Gestion de la TVA   L’application doit permettre de comptabiliser le montant de TVA à déclarer entre deux déclarations et afficher ce montant sur le tableau de bord. Seules les factures payées sont concernées et c’est la date à laquelle la facture a été effectivement payée qui doit être prise en compte. Maquettage Le maquettage d'une application est le processus de création de schémas visuels rudimentaires qui représentent l'interface utilisateur d'une application, y compris la disposition des éléments, la navigation et l'apparence générale. Il sert à planifier la structure et le design avant le développement, permettant aux concepteurs et aux développeurs de mieux comprendre les besoins et les fonctionnalités de l'application. Les maquettes sont généralement des prototypes non fonctionnels qui aident à visualiser et à valider les idées avant la création de la version finale de l'application. Gestion de facturesZONING Liste des factures :  A white rectangular object with lines  Description automatically generated with medium confidence  Crée une nouvelle facture  Facture1  Facture 2  Facture3  Facture4  A screenshot of a computer  Description automatically generated  A screenshot of a computer  Description automatically generated  A blue and white list with black text  Description automatically generated  A screenshot of a computer  Description automatically generated  Ajouter/Modifier facture :    Client  Bon de commande  Payée le  Émise le  Échéance le  Détail de la facturation  Créé  Annulez         Emettre facture pour non-paiement :  WIREFRAM A screenshot of a computer screen  Description automatically generated Structure du projetEmission d’une factureIntégration dans le Backend : ➢Pom.xml :  Ces dépendances sont utilisées dans un projet Java basé sur Spring Boot pour la gestion des e-mails. La première dépendance, "spring-boot-starter-mail", inclut les fonctionnalités de base pour envoyer des e-mails avec Spring Boot. La deuxième dépendance, "javax.mail:mail:1.4.3", est une bibliothèque Java standard pour la manipulation des e-mails, et elle est utilisée en conjonction avec Spring Boot pour les opérations plus avancées, telles que la construction de messages personnalisés et l'accès aux pièces jointes. Ces dépendances facilitent le développement d'applications qui nécessitent des fonctionnalités d'envoi et de réception d'e-mails. La version 1.4.3 de javax.mail est spécifiée pour garantir la compatibilité avec Spring Boot.    ➢Src/main/application.properties :  Ces paramètres de configuration sont destinés à permettre à une application Spring d'envoyer des e-mails via le serveur SMTP de Gmail (smtp.gmail.com). Les informations d'identification (nom d'utilisateur et mot de passe) sont fournies pour authentifier l'application, et le port 587 est spécifié pour la communication. De plus, l'option "starttls" est activée pour assurer une communication sécurisée avec le serveur SMTP de Gmail.   Couche DTO J’ai ajouté un Data Transfer Object (DTO) appelé **InvoiceMailDto** au sein de la couche service, ce qui nous permet de mieux gérer les données que nous envoyons par e-mail.  A screenshot of a computer  Description automatically generated Couche Repository et services J'ai enrichi notre application en introduisant un tout nouveau service d'envoi d'e-mails.  Cette interface déclare une méthode `send` pour envoyer un e-mail avec une pièce jointe, prenant comme arguments un objet `InvoiceMailDto`,  A screen shot of a computer program  Description automatically generated  Ce code représente une implémentation d'un service d'envoi d'e-mails avec des pièces jointes. Il gére l'envoi d'e-mails, en injectant `JavaMailSender`. La méthode `send` crée un e-mail avec une pièce jointe à partir des informations fournies dans l'objet `InvoiceMailDto`, puis envoie cet e-mail. En cas d'erreur, des exceptions sont gérées pour la notification.  A screen shot of a computer program  Description automatically generated  `InvoiceMailer` utilisée pour envoyer des e-mails avec des pièces jointes. Il utilise la bibliothèque Spring pour gérer l'envoi d'e-mails, en injectant `JavaMailSender`. La méthode `send` crée un e-mail avec une pièce jointe à partir des informations fournies dans l'objet `InvoiceMailDto` et envoie cet e-mail. Notez qu'il semble y avoir une erreur concernant la déclaration de la variable `mailSender`, car elle est déclarée en tant que statique.    A screen shot of a computer program  Description automatically generated Couche contrôleur De plus, j'ai effectué une mise à jour importante dans le contrôleur InvoiceController. Plus précisément, j'ai intégré un appel à ce service d'envoi d'e-mails au sein de la méthode "publishInvoice," qui est responsable d'émettre une facture. Cette modification nous permet d'automatiser l'envoi d'e-mails à chaque fois qu'une facture est émise.    A screen shot of a computer screen  Description automatically generated Intégration dans le frontendCouche components Afin de développer cette fonctionnalité j’ai commencé par créer un composant React qui crée une modal de suppression. Dans ce composant :   1. J’ai ajouté l’importations de différents modules nécessaires.    1. InvoiceService service qui gère les opérations liées aux factures.    2. InvoiceMail et Invoice sont des modèles de données utilisés pour représenter les informations d'une facture et d'un courrier électronique, respectivement.   A computer screen with green text  Description automatically generated   1. J’ai déclaré une interface props qui définit les propriétés attendues par le composant.  * show: Un booléen indiquant si la modal doit être affichée ou non. * setShowModal: Une fonction pour mettre à jour l'état de la modal. * invoice: Une instance de la classe Invoice (ou undefined si non spécifié). * updateInvoiceList: Une fonction pour mettre à jour la liste des factures.   A black screen with red and white text  Description automatically generated   1. J’ai déclaré le composant PublishModal, Il reçoit des propriétés (props) conformes à l'interface définie précédemment.   A black background with white text  Description automatically generated  Le composant PublishModal est défini ici. Il reçoit des propriétés (props) conformes à l'interface définie précédemment.   1. J’ai ajouté le hook useState pour gérer l'état de l'ouverture/fermeture de la modal, et useEffect pour mettre à jour l'état open lorsque la propriété show change:   A black screen with white text  Description automatically generated   1. J’ai utilisé useFormik pour gérer le formulaire à l'intérieur de la modal. J’ai initialisé les valeurs du formulaire avec l'ID de la facture , le destinataire , le sujet et le contenue du mail, et j’ai définit une fonction onSubmit avec un appel au service externe (**InvoiceService.publishInvoice**) pour supprimer une facture qui sera exécutée lorsque le formulaire est soumis   A screen shot of a computer program  Description automatically generated  A computer screen shot of a program code  Description automatically generated   1. J’ai ajouté le composant rend une modal avec un formulaire avec des champs pour l'adresse e-mail du destinataire, le sujet et le corps du courrier électronique. L'utilisateur peut annuler l'action ou envoyer la facture par courrier électronique en cliquant sur les boutons correspondants.   A screen shot of a computer program  Description automatically generated Couche views  1. J’ai ajouté l’importations de différents modules nécessaires.      1. À l'intérieur du composant Invoices, j’ai ajouté les états à l'aide du Hook useState de React. Ces états incluent :    1. invoices, qui est un tableau de factures (Invoice[]).    2. showPublishModal, un booléen qui contrôle l'affichage du modèle de publication.    3. invoiceToPublish, qui stockera une facture à publier.   A screenshot of a computer program  Description automatically generated   1. J’ai ajouté une fonction **publishInvoice** qui prend un paramètre id (supposément l'ID de la facture à publier). Cette fonction est utilisée pour préparer une facture spécifique à la publication en mettant à jour l'état **invoiceToPublish** avec la facture correspondante trouvée dans le tableau invoices. De plus, elle définit l'état **showPublishModal** sur true, ce qui affiche le modèle de publication.   A computer screen with white text  Description automatically generated   1. J’ai ajouté la déclaration du modale « PublishModal »   A computer screen with white text  Description automatically generated Emission d’une facture impayéeIntégration dans le BackendCouche DTO N/A Couche Repository et services Service `InvoiceService` : Vous avez modifié le service `InvoiceService` pour y ajouter un appel à l'API "unpaid". Cela signifie que le service est désormais capable de communiquer avec la méthode "Unpaid" du contrôleur `InvoiceController`. Couche contrôleur Contrôleur `InvoiceController` : j’ajoute une nouvelle méthode appelée "Unpaid" dans le contrôleur `InvoiceController`. Cette méthode semble faire appel à un service de messagerie avec en entrée un objet `InvoiceMailDto`. Cette méthode pourrait être utilisée pour gérer des factures impayées.  A screenshot of a computer program  Description automatically generated Intégration dans le frontendCouche components Afin de développer cette fonctionnalité j’ai commencé par créer un composant React qui crée une modal de suppression. Dans ce composant :   1. J’ai ajouté l’importations de composants et de bibliothèques : le code importe divers composants et bibliothèques nécessaires à la construction de la modal, y compris des composants MUI (Material-UI) tels que Modal, Backdrop, Fade, Button, Typography, et d'autres éléments   A computer screen shot of a program code  Description automatically generated   1. J’ai déclaré une interface props qui définit les propriétés attendues par le composant.  * show: Un booléen indiquant si la modal doit être affichée ou non. * setShowModal: Une fonction pour mettre à jour l'état de la modal. * invoice: Une instance de la classe Invoice (ou undefined si non spécifié). * updateInvoiceList: Une fonction pour mettre à jour la liste des factures.   A black background with red and white text  Description automatically generated   1. J’ai déclaré le composant DeleteModal, Il reçoit des propriétés (props) conformes à l'interface définie précédemment.   A black background with white text  Description automatically generated  Le composant UnpaidModal est défini ici. Il reçoit des propriétés (props) conformes à l'interface définie précédemment.   1. J’ai ajouté les Hooks React : tels que useTheme() pour obtenir le thème actuel, et useTranslation() pour la gestion de la traduction dans l'application.   A black background with white text  Description automatically generated   1. J’ai ajouté le hook useState pour gérer l'état de l'ouverture/fermeture de la modal, et useEffect pour mettre à jour l'état open lorsque la propriété show change:   A black screen with white text  Description automatically generated   1. J’ai utilisé useFormik pour gérer le formulaire à l'intérieur de la modal. J’ai initialisé les valeurs du formulaire avec l'ID de la facture , le destinataire , le sujet et le contenue du mail, et   A computer screen with white text and blue text  Description automatically generated   1. J’ai définit une fonction onSubmit avec un appel au service externe (**InvoiceService.unpaidInvoice**) pour supprimer une facture qui sera exécutée lorsque le formulaire est soumis   A computer screen shot of a program  Description automatically generated   1. J’ai ajouté le composant rend une modal utilisant les composants MUI (Modal, Backdrop, Fade). Le contenu de la modal comprend un titre, des boutons "Annuler" et "Confirmer", et il est géré par le formulaire défini précédemment.   A screen shot of a computer program  Description automatically generated  49 Couche views  1. J’ai ajouté l’importations de différents modules nécessaires.      1. À l'intérieur du composant Invoices, j’ai ajouté les états à l'aide du Hook useState de React. Ces états incluent :    1. invoices, qui est un tableau de factures (Invoice[]).    2. showUnpaidModal, un booléen qui contrôle l'affichage du modèle de publication.    3. invoiceToUnpaid, qui stockera une facture à publier.   A screen shot of a computer  Description automatically generated   1. J’ai ajouté une fonction **unpaidInvoice** qui prend un paramètre id (supposément l'ID de la facture à publier). Cette fonction est utilisée pour préparer une facture spécifique à envoyer en mettant à jour l'état **invoiceToUnpaid** avec la facture correspondante trouvée dans le tableau invoices. De plus, elle définit l'état **showUnpaidModal** sur true, ce qui affiche le modèle de publication.   A computer code with white text  Description automatically generated   1. J’ai ajouté la déclaration du modale « UnpaidModal » et j’ai ajouté une grid actions.   A computer screen shot of a program  Description automatically generated Suppression d’une factureIntégration dans le frontendCouche components Afin de développer cette fonctionnalité j’ai commencé par créer un composant React qui crée une modal de suppression. Dans ce composant :   1. J’ai ajouté l’importations de composants et de bibliothèques : le code importe divers composants et bibliothèques nécessaires à la construction de la modal, y compris des composants MUI (Material-UI) tels que Modal, Backdrop, Fade, Button, Typography, et d'autres éléments   A computer screen shot of a program code  Description automatically generated   1. J’ai déclaré une interface props qui définit les propriétés attendues par le composant.   A black background with red and white text  Description automatically generated   1. J’ai déclaré le composant DeleteModal, Il reçoit des propriétés (props) conformes à l'interface définie précédemment.   A black background with white text  Description automatically generated  Le composant DeleteModal est défini ici. Il reçoit des propriétés (props) conformes à l'interface définie précédemment.   1. J’ai ajouté les Hooks React : tels que useTheme() pour obtenir le thème actuel, et useTranslation() pour la gestion de la traduction dans l'application.   A black background with white text  Description automatically generated 57   1. J’ai ajouté le hook useState pour gérer l'état de l'ouverture/fermeture de la modal, et useEffect pour mettre à jour l'état open lorsque la propriété show change:   A black screen with white text  Description automatically generated   1. J’ai utilisé useFormik pour gérer le formulaire à l'intérieur de la modal. J’ai initialisé les valeurs du formulaire avec l'ID de la facture et la facture elle-même, et j’ai définit une fonction onSubmit avec un appel au service externe (**InvoiceService.deleteInvoice**) pour supprimer une facture qui sera exécutée lorsque le formulaire est soumis   A computer screen with white text and blue text  Description automatically generated   1. J’ai ajouté le composant rend une modal utilisant les composants MUI (Modal, Backdrop, Fade). Le contenu de la modal comprend un titre, des boutons "Annuler" et "Confirmer", et il est géré par le formulaire défini précédemment.   A screen shot of a computer program  Description automatically generated Couche views  1. J’ai ajouté l’importations de différents modules nécessaires.      1. À l'intérieur du composant Invoices, j’ai ajouté les états à l'aide du Hook useState de React. Ces états incluent :    1. showDeletedModal, un booléen qui contrôle l'affichage du modèle de suppression.    2. invoiceToDeleted, qui stockera une facture à publier.   A computer screen with white text  Description automatically generated   1. J’ai ajouté une fonction **deleteInvoice** qui prend un paramètre id (supposément l'ID de la facture à supprimer).   Cette fonction est utilisée pour préparer une facture spécifique à envoyer en mettant à jour l'état **invoiceToDelete** avec la facture correspondante trouvée dans le tableau invoices. De plus, elle définit l'état **showDeleteModal** sur true, ce qui affiche le modèle de publication.  A computer screen with white text  Description automatically generated   1. J’ai ajouté la déclaration du modale « **Delete**Modal ».   A computer screen shot of a program code  Description automatically generated Démonstration et testsGestion TVAEmission d’une facture A screenshot of a computer  Description automatically generated  A screenshot of a computer  Description automatically generated Emission d’une facture impayée A screenshot of a computer  Description automatically generated  A screenshot of a computer  Description automatically generated Suppression d’une facture A screenshot of a computer  Description automatically generated | | | |
|  | | | |
|  | | |
|  | | | |
| **2. Précisez les moyens utilisés :** | | | |
| 1. Pour créer les maquettes, j'utilise l'outil Figma, qui me permet de visualiser et concevoir l'interface de manière détaillée. 2. Mon choix s'est porté sur le Framework Spring pour développer l'architecture de l'application, garantissant ainsi une structure solide et modulable. 3. Mon éditeur de prédilection est Visual Studio Code et Eclipse, auquel j'ajoute différentes extensions pour une documentation exhaustive du code et une gestion efficace des espaces de noms des classes. | | | |
|  | | | |
|  | | | |
|  | | | |
|  | | | |
| **3. Avec qui avez-vous travaillé ?** | | | |
|  | | | |
| J’ai travaillé seule | | | |
|  | | | |
|  | | |
|  | | | |
| **4. Contexte** | | | |
|  | |  | |
| Nom de l’entreprise, organisme ou association | | *Idformation* | |
| Chantier, atelier, service | Amélioration d'une application de gestion de micro-entreprise | | |
| Période d’exerciceDu : *18/09/2023*au :*22/09/2023* | | | |
|  | | |
|  | | |
| **5. Informations complémentaires** *(facultatif)* | | |
|  | | | |
| Rien | | | |
|  | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Activité-type** | **2** | **Concevoir et développer la persistance des données en intégrant les recommandations de sécurité** | | | |
|  | | | | | |
|  | | | | | |
| **1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :** | | | | | |
| 1. Après avoir procédé à l'analyse approfondie de mon application, je commence à réfléchir aux tables et aux champs qui constitueront la base de données, ainsi qu'aux relations entre ces tables. Je construis en utilisant la méthode Merise :    1. Le dictionnaire de données    2. Le Modèle Conceptuel de Données (MCD)    3. Le Modèle logique de Données    4. Le Modèle Logique de Données (MLD) afin d'identifier clairement les clés primaires et les clés étrangères. 2. Pour concrétiser ces schémas, je crée la base de données dans mon système de gestion BD. 3. Pour garantir la sécurité des informations, je stocke les mots de passe de manière cryptée dans la base de données, en utilisant l'algorithme BCRYPT. 4. J'ai testé le modèle de données en effectuant des saisies automatisées de jeux de données (fixtures) dans la base, me permettant ainsi de vérifier son bon fonctionnement et son intégrité. | | | | | |
|  | | | | | |
| Le script :  ----  CREATE TABLE `COMMANDES` (  `id\_commande` int NOT NULL AUTO\_INCREMENT,  `type\_commande` varchar(10) NOT NULL,  `etat\_commande` varchar(10) NOT NULL,  `total\_commande` DECIMAL(6,2) NOT NULL,  `id\_utilisateur` int NOT NULL AUTO\_INCREMENT,  PRIMARY KEY (`id\_commande`)  ) ;  ----  CREATE TABLE `COMPOSER` (  `id\_commande` int NOT NULL AUTO\_INCREMENT,  `id\_pizza` int NOT NULL AUTO\_INCREMENT,  `quantite` int NOT NULL,  `prix` DECIMAL(6,2) NOT NULL,  PRIMARY KEY (`id\_commande`, `id\_pizza`)  ) ;  ----  CREATE TABLE `PIZZA` (  `id\_pizza` int NOT NULL AUTO\_INCREMENT,  `nom\_pizza` varchar(100) NOT NULL,  `description\_pizza` varchar(255) NOT NULL,  `prix\_pizza` DECIMAL(6,2) NOT NULL,  `image\_pizza` BLOB NOT NULL,  PRIMARY KEY (`id\_pizza`)  ) ;  ----  CREATE TABLE `ROLES` (  `id\_role` int NOT NULL AUTO\_INCREMENT,  `nom\_role` varchar(100) NOT NULL,  PRIMARY KEY (`id\_role`)  ) ;  ----  CREATE TABLE `UTILISATEURS` (  `id\_utilisateur` int NOT NULL AUTO\_INCREMENT,  `nom\_utilisateur` varchar(100) NOT NULL,  `prenom\_utilisateur` varchar(100) NOT NULL,  `adresse\_utilisateur` varchar(MAX) NOT NULL,  `telephone\_utilisateur` nvarchar(12) NOT NULL,  `motpasse\_utilisateur` nvarchar(100) NOT NULL,  `id\_role` int NOT NULL AUTO\_INCREMENT,  PRIMARY KEY (`id\_utilisateur`)  ) ;  ----  ALTER TABLE `COMMANDES` ADD FOREIGN KEY (`id\_utilisateur`) REFERENCES `UTILISATEURS` (`id\_utilisateur`);  ALTER TABLE `COMPOSER` ADD FOREIGN KEY (`id\_pizza`) REFERENCES `PIZZA` (`id\_pizza`);  ALTER TABLE `COMPOSER` ADD FOREIGN KEY (`id\_commande`) REFERENCES `COMMANDES` (`id\_commande`);  ALTER TABLE `UTILISATEURS` ADD FOREIGN KEY (`id\_role`) REFERENCES `ROLES` (`id\_role`); | | | | | |
|  | | | | |
|  | | | | | |
| **2. Précisez les moyens utilisés :** | | | | | |
|  | | | | | |
| DBeaver, Lucidechard. | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
| **3. Avec qui avez-vous travaillé ?** | | | | | |
|  | | | | | |
| seul | | | | | |
|  | | | | | |
|  | | | | |
|  | | | | | |
| **4. Contexte** | | | | | |
|  | | | |  | |
| Nom de l’entreprise, organisme ou association | | | | *idformation* | |
| Chantier, atelier, service | | | Mario pizza | | |
| Période d’exerciceDu : *14/08/2023*au :*18/08/2023* | | | | | |
|  | | | | |
|  | | | | |
| **5. Informations complémentaires** *(facultatif)* | | | | |
|  | | | | | |
| Rien | | | | | |
|  | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Activité-type** | **3** | **Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité** | | | |
|  | | | | | |
|  | | | | | |
| **1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :** | | | | | |
| 1. Je m’organise et je réalise un planning et un tableau Trello que je mets régulièrement à jour pour suivre l’avancée du projet et voir si je respecte bien les délais. 2. J’utilise une méthode agile dans la gestion de projet et j’adopte une démarche itérative. 3. J’utilise un système de versioning GIT.   Voici un exemple :  Gestion du projet  L’application Gestion micro-entreprise est réalisé dans le cadre de projet fin de formation au sein de IdFormation.  Le but est de concevoir et développer une application d’une micro-entreprise Pour ceci, nous avons pu mettre en place des bonnes pratiques rencontrées au cours de la formation.  AGILE/SCRUM et Trello  Étant donné que notre période de stage était limitée à un mois, nous avions besoin de flexibilité, ce que l’approche AGILE/SCRUM nous a offert.  Il s'agit d'un cadre de gestion de projet qui segmente les projets en plusieurs phases dynamiques appelées “sprints”.  À la fin de chaque sprint, nous discutions des axes d'amélioration, des éléments bloquants et des facteurs ralentisseurs (“parachute”) à considérer pour le sprint suivant. Nous tenions également des “dailies” – de courtes réunions nous permettant de faire le point sur les tâches réalisées et celles prévues pour la journée. De plus, en cas de problème, nous n'attendions pas la fin du sprint pour le signaler. Ce processus favorisait une compréhension mutuelle du projet et un partage efficace des informations.  Les sprints constituaient aussi des moments propices pour revoir les tickets ou en créer de nouveaux. Ces tickets représentaient des demandes à traiter, segmentant ainsi le projet en plusieurs sous-parties. Nous utilisions Trello pour les gérer.  Chaque ticket est structuré en trois parties : la demande initiale (celle du client), la description et les “besoins” ou solutions potentielles pour sa mise en œuvre.    Le ticket est décomposé́ en trois parties : la demande initiale (celle du client), la description et les “besoins” ou solutions potentielles pour réaliser le ticket. | | | | | |
| ➢GIT  Nous avions établi une organisation précise pour Git, notamment en ce qui concerne la nomenclature des branches. Une branche était dédiée à chaque ticket sur lequel nous travaillions.  Lorsque je développais une fonctionnalité, je fusionnais la branche associée avec une branche de "pré-production". Plusieurs branches "pré-production" étaient présentes, chacune correspondant au numéro du sprint en cours.  Finalement, nous disposions d'une branche "production" (la branche principale) qui accueillait les fusions jugées "finales". En d'autres mots, une fois que toutes les fusions d'un sprint étaient opérationnelles et compatibles, je les intégrais à cette branche principale. | | | | | |
|  | | | | |
|  | | | | | |
| **2. Précisez les moyens utilisés :** | | | | | |
|  | | | | | |
| J’ai utilisé :   * Trello pour l’organisation des tickets * Git * Discord | | | | | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
| **3. Avec qui avez-vous travaillé ?** | | | | | |
|  | | | | | |
| Sondes, Hugo, yacine | | | | | |
|  | | | | | |
|  | | | | |
|  | | | | | |
| **4. Contexte** | | | | | |
|  | | | |  | |
| Nom de l’entreprise, organisme ou association | | | | *Idformation* | |
| Chantier, atelier, service | | | Amélioration d'une application de gestion de micro-entreprise | | |
| Période d’exerciceDu : Cliquez ici *25/09/2023*au :*Cliquez ici 02/10/2023* | | | | | |
|  | | | | |
|  | | | | |
| **5. Informations complémentaires** *(facultatif)* | | | | |
|  | | | | | |
| Rien | | | | | |
|  | | | | | |

|  |  |  |
| --- | --- | --- |
| **Activité-type** | **2** | **Concevoir et développer des composants d’interface utilisateur en intégrant les recommandations de sécurité** |
|  | | |
|  | | |
| **1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :** | | |
| Le sujet "Évaluation CCP3 - Application mobile de réservation de commande de pizzas" porte sur la conception d'une application mobile pour la pizzeria Mario's Pizza.   1. Après avoir fait l'analyse de mon application, je commence à penser aux composants et aux fonctionnalités que va contenir l'application React Native. 2. Je crée l'application React Native en configurant les composants et en définissant les états nécessaires. Je travaille sur la logique de l'application en utilisant JavaScript et les bibliothèques React Native appropriées pour gérer les interactions utilisateur et les réponses aux événements. 3. Je développe les différentes vues et composants de l'application en utilisant React Native, en assurant la cohérence du design et la réactivité de l'interface utilisateur. Une fois les composants créés, je teste l'application sur Android Studio, un émulateur Android, pour m'assurer de son bon fonctionnement sur la plateforme Android. 4. Pour garantir la sécurité des données, je mets en place des méthodes de stockage sécurisées en utilisant React Native Secure Storage ou d'autres solutions appropriées pour crypter et protéger les informations sensibles. 5. J'ai testé l'application sur Android Studio en simulant différents scénarios d'utilisation, en m'assurant que tous les composants interagissent correctement et que l'application répond de manière optimale aux diverses actions de l'utilisateur. 6. Dans le processus de développement, j'ai utilisé des outils spécifiques à React Native pour optimiser les performances de l'application, assurer sa compatibilité avec Android et garantir une expérience utilisateur fluide et agréable sur les appareils Android.   Voici une description structurée en utilisant le modèle MVC :   1. Modèles (Model) :   La classe `JwtResponse` est une représentation d'un objet de réponse contenant un JSON Web Token (JWT). Ce jeton JWT est utilisé pour l'authentification et l'autorisation dans les applications. La classe a deux attributs : `token` (qui stocke le jeton JWT) et `tokenHeader` (qui stocke l'en-tête du jeton). Elle est conçue pour encapsuler les réponses reçues lors d'opérations d'authentification, facilitant ainsi la manipulation et la gestion des jetons JWT dans une application.  export default class JwtResponse {  token: string;  tokenHeader: string;  constructor(  token: string,  tokenHeader: string  ) {  this.token = token;  this.tokenHeader = tokenHeader;  }  }  La classe `LoginRequest` représente une demande de connexion et possède deux attributs : `username` (nom d'utilisateur) et `password` (mot de passe). Elle est conçue pour encapsuler les informations d'identification d'un utilisateur lorsqu'il tente de se connecter à une application.  export default class LoginRequest {  username: string;  password: string;  constructor(  username: string,  password: string  ) {  this.username = username;  this.password = password;  }  }  La classe `LoginResponse` représente une réponse à une demande de connexion et comprend les éléments suivants :  1. `jwt` (de type string) : Cet attribut contient un jeton JWT (JSON Web Token), qui est généralement utilisé pour l'authentification et l'autorisation de l'utilisateur.  2. `expiration` (de type string) : Cet attribut indique la date d'expiration du jeton JWT.  3. `user` (de type User) : Cet attribut est un objet de type `User` qui représente les informations de l'utilisateur connecté, telles que son nom, prénom, adresse, etc.  La classe `LoginResponse` est conçue pour encapsuler les données renvoyées en réponse à une demande de connexion, fournissant ainsi le jeton JWT, sa date d'expiration et les détails de l'utilisateur connecté. Cela facilite la gestion de la réponse de connexion dans une application, permettant d'accéder aux informations d'identification et aux données de l'utilisateur connecté.  import User from "./User";  export default class LoginResponse {  jwt: string;  expiration: string;  user: User;  constructor(jwt: string, expiration: string, user: User) {  this.jwt = jwt;  this.expiration = expiration;  this.user = user;  }  }  La classe `Order` représente une commande et contient trois attributs : `totalAmount` (montant total de la commande), `user` (identifiant de l'utilisateur) et `orderlineDTOS` (une liste d'objets de type `OrderLine` représentant les lignes de commande). Cette classe encapsule les détails d'une commande, y compris les produits commandés et le montant total.  import OrderLine from "./OrderLine";  export default class Order {  totalAmount: number;  user: string;  orderlineDTOS: Array<OrderLine>;  constructor(  totalAmount: number,  user: string,  orderlineDTOS: Array<OrderLine>  ) {  this.totalAmount = totalAmount;  this.orderlineDTOS = orderlineDTOS;  this.user = user;  }  }  Le code définit une classe `Pizza` pour représenter des informations sur une pizza, y compris son identifiant, nom, description, image et prix. Cette classe est conçue pour stocker les détails spécifiques à une pizza dans une application, par exemple, une application de commande de pizzas.  import { Double } from "react-native/Libraries/Types/CodegenTypes";  export default class Pizza {  id: number;  name: string;  description: string;  image: string;  price: Double;  constructor(  id: number,  name: string,  description: string,  image: string,  price: Double  ) {  this.id = id;  this.name = name;  this.description = description;  this.image = image;  this.price = price;  }  }  La classe `User` représente un modèle utilisateur avec des attributs tels que l'identifiant, le numéro de téléphone, le mot de passe, le prénom, le nom et l'adresse. Elle est utilisée pour stocker les informations d'un utilisateur dans une application, par exemple, pour gérer les utilisateurs inscrits.  export default class User {  id: number;  phonenumber: string;  password: string;  firstname: string;  lastname: string;  address: string;  constructor(  id: number,  phonenumber: string,  password: string,  firstname: string,  lastname: string,  address: string  ) {  this.id = id;  this.phonenumber = phonenumber;  this.password = password;  this.firstname = firstname;  this.lastname = lastname;  this.address = address;  }  }  En Pages  import {  RefreshControl,  ScrollView,  StyleSheet,  Text,  TextInput,  View,  } from "react-native";  import React, { useCallback, useState } from "react";  import Header from "../components/Header/Header";  import CustomButton from "../components/CustomButton/CustomButton";  import CustomInput from "../components/CustomInput/CustomInput";  import { useNavigation } from "@react-navigation/native";  import { NativeStackNavigationProp } from "@react-navigation/native-stack";  import { useForm } from "react-hook-form";  import AuthenticationService from "../service/AuthenticationService";  import { RouteParams } from "../Routes/RootNavigator";  const InscriptionScreen = () => {  const navigation = useNavigation<NativeStackNavigationProp<RouteParams>>();  // message juste apres avoir clic sur valider lors de l'inscription, message de chargement  const [chargement, setChargement] = useState("");  const {  control,  handleSubmit,  watch,  formState: { errors },  } = useForm();  // contient la valeur du mot de passe  const mdp = watch("password");  // test la connexion de l'utilisateur, si tout est bon, permet a l'utilisateur de se connecter  const login = () => {  AuthenticationService.login(watch("phonenumber"), watch("password")).then(  (isAuthenticated: any) => {  if (!isAuthenticated) {  console.log("ça Marche Pas");  return;  }  navigation.navigate("Bienvenue");  }  );  };  // Ajoute l'utilisateur a la base de donnée + connecte l'utilisateur a l'application  const Creation = (data: any) => {  setChargement("Chargement. . .");  AuthenticationService.addUser(data).then(login);  };  return (  <View style={styles.container}>  <Header name="Créez votre compte client " />  {/\* FORMULAIRE \*/}  <ScrollView style={styles.scroll} showsVerticalScrollIndicator={false}>  {/\* NOM \*/}  <Text style={styles.text}>Nom</Text>  <CustomInput  name="lastname"  placeholder="Votre nom"  control={control}  rules={{  required: "Votre nom est obligatoire",  minLength: {  value: 2,  message: "Le nom doit contenir au minimum 2 caractères",  },  maxLength: { value: 120, message: "Le nom est trop long" },  }}  />  {/\* PRENOM \*/}  <Text style={styles.text}>Prénom</Text>  <CustomInput  name="firstname"  placeholder="Votre prénom"  control={control}  rules={{  required: "Votre prénom est obligatoire",  minLength: {  value: 2,  message: "Le prénom doit contenir au minimum 2 caractères",  },  maxLength: { value: 120, message: "Le prenom est trop long" },  }}  />  {/\* MOT DE PASSE \*/}  <Text style={styles.text}>Mot de passe</Text>  <CustomInput  name="password"  secureTextEntry  type="mdp"  placeholder="Votre mot de passe"  control={control}  rules={{  required: "Le mot de passe est obligatoire",  minLength: {  value: 6,  message: "Le mot de passe dois faire 6 caractères minimum",  },  maxLength: { value: 120, message: "Le mot de passe est trop long" },  }}  />  {/\* CONFIRME MOT DE PASSE \*/}  <Text style={styles.text}>Confirmation</Text>  <CustomInput  name="confirmPassword"  secureTextEntry  type="mdp"  placeholder="Confirmation du mot de passe"  control={control}  rules={{  required: "Confirmer le mot de passe est obligatoire",  minLength: {  value: 6,  message: "Le mot de passe dois faire 6 caractères minimum",  },  maxLength: { value: 120, message: "Le mot de passe est trop long" },  validate: (value: string) =>  value === mdp ? true : "Le mot de passe ne correspond pas",  }}  />  {/\* NUMERO TELEPHONE \*/}  <Text style={styles.text}>Téléphone</Text>  <CustomInput  name="phonenumber"  type="number"  placeholder="Téléphone"  control={control}  rules={{  required: "Le numéro de téléphone est obligatoire",  minLength: {  value: 10,  message: "Le numéro dois faire 10 caractères minimum",  },  maxLength: { value: 20, message: "Le numéro est trop long" },  }}  />  {/\* ADRESSE \*/}  <Text style={styles.text}>Adresse</Text>  <CustomInput  name="address"  type="adresse"  placeholder="Votre Adresse pour la livraison"  control={control}  rules={{  required: "L'adresse est obligatoire",  maxLength: { value: 120, message: "L'adresse est trop longue" },  }}  />  {/\* BOUTON VALIDER \*/}  <CustomButton  text="Créer"  backgroundColor="#FFC921"  color="#341f97"  align="flex-end"  onPress={handleSubmit((data) => Creation(data))}  />  <Text style={{ color: "#FFC921", fontSize: 20 }}>{chargement}</Text>  </ScrollView>  </View>  );  };  export default InscriptionScreen;  const styles = StyleSheet.create({  container: {  backgroundColor: "#3b438b",  height: "100%",  },  text: {  margin: 10,  color: "#FFC921",  fontWeight: "bold",  fontSize: 17,  },  scroll: {  width: "80%",  marginLeft: "5%",  marginTop: "5%",  },  });  Ce code représente une page d'inscription dans une application mobile. Il permet aux utilisateurs de créer un compte en fournissant leurs informations, y compris nom, prénom, mot de passe, numéro de téléphone et adresse. Une fois les informations saisies, l'utilisateur peut appuyer sur le bouton "Créer" pour enregistrer son compte. Le code utilise React Native et des composants personnalisés pour gérer le formulaire d'inscription et la navigation entre les écrans.  import { View, Text, StyleSheet, LogBox } from "react-native";  import React, { useState } from "react";  import { useNavigation } from "@react-navigation/native";  import { RouteParams } from "../Routes/RootNavigator";  import { NativeStackNavigationProp } from "@react-navigation/native-stack";  import CustomInput from "../components/CustomInput/CustomInput";  import CustomButton from "../components/CustomButton/CustomButton";  import Header from "../components/Header/Header";  import { useForm } from "react-hook-form";  import AuthenticationService from "../service/AuthenticationService";  const LoginPage = () => {  const navigation = useNavigation<NativeStackNavigationProp<RouteParams>>();  const {  control,  handleSubmit,  formState: { errors },  watch,  } = useForm();  // Pour message de connexion  const [message, setMessage] = useState<string>(" ");  // Connecte l'utilisateur et l'envoi a la liste des pizzas  const goPizza = (data: any) => {  console.log(data);  LogBox.ignoreAllLogs();  // tentative Connexion  setMessage(" Tentative de connexion");  AuthenticationService.login(watch("identifiant"), watch("mdp")).then(  (isAuthenticated: any) => {  if (!isAuthenticated) {  setMessage("Identifiant ou mot de passe incorrect.");  console.log("Erreur à la connexion");  return;  }  setMessage("");  navigation.navigate("Pizza");  }  );  };  return (  <>  <Header />  <View style={styles.container}>  {/\* FORMULAIRE \*/}  <View style={styles.login}>  {/\* IDENTIFIANT \*/}  <Text style={styles.text}>Identifiant</Text>  <CustomInput  name="identifiant"  placeholder="Téléphone"  type="number"  control={control}  rules={{  required: "L'identifiant est obligatoire",  minLength: {  value: 10,  message: "L'identifiant dois faire 10 caractères minimum",  },  }}  />  {/\* MOT DE PASSE \*/}  <Text style={styles.text}>Mot de passe</Text>  <CustomInput  name="mdp"  secureTextEntry  type="mdp"  placeholder="Mot de passe"  control={control}  rules={{  required: "Le mot de passe est obligatoire",  minLength: {  value: 6,  message: "Le mot de passe dois faire 6 caractères minimum",  },  }}  />  {/\* PAS DE COMPTE \*/}  <Text style={styles.new} onPress={() => navigation.navigate("New")}>  créer un compte  </Text>  {/\* CONNEXION \*/}  <CustomButton  text="Connexion"  backgroundColor="#FFC921"  color="#341f97"  align="flex-end"  onPress={handleSubmit(goPizza)}  />  <Text style={{ color: "#FFC921", margin: 10 }}>{message}</Text>  </View>  </View>  </>  );  };  const styles = StyleSheet.create({  container: {  height: "100%",  alignItems: "center",  backgroundColor: "#3b438b",  width: "100%",  },  text: {  margin: 10,  color: "#FFC921",  fontWeight: "bold",  fontSize: 20,  },  login: {  marginTop: 60,  width: "60%",  },  new: {  margin: 10,  color: "#FFC921",  alignSelf: "flex-end",  textDecorationLine: "underline",  },  });  export default LoginPage;  Ce code représente une page de connexion dans une application mobile, permettant aux utilisateurs de saisir leur identifiant et mot de passe. Lorsque les informations sont soumises, l'application tente de connecter l'utilisateur, affichant un message de connexion. Si la connexion est réussie, l'utilisateur est dirigé vers une liste de pizzas. Il utilise React Native avec des composants personnalisés pour gérer le formulaire de connexion et la navigation entre les écrans.  import { StyleSheet, View } from "react-native";  import React, { useCallback, useEffect, useState } from "react";  import Header from "../components/Header/Header";  import Pizza from "../models/Pizza";  import PizzaService from "../service/PizzaService";  import Footer from "../components/Footer/Footer";  import { CommonActions, useNavigation } from "@react-navigation/native";  import { NativeStackNavigationProp } from "@react-navigation/native-stack";  import { RouteParams } from "../Routes/RootNavigator";  import PizzaList from "../components/PizzaList/PizzaList";  import OrderLine from "../models/OrderLine";  import Order from "../models/Order";  const OrderScreen = () => {  const navigation = useNavigation<NativeStackNavigationProp<RouteParams>>();  // Pizza et Liste de pizza  const [pizzas, setPizzas] = useState<Pizza[]>([]);  useEffect(() => {  PizzaService.getPizzas().then((pizzas) => setPizzas(pizzas));  }, []);  // parametre pour refresh la page  const wait = (timeout: number) => {  return new Promise((resolve) => setTimeout(resolve, timeout));  };  const [refreshing, setRefreshing] = useState(false);  const onRefresh = useCallback(() => {  setRefreshing(true);  const resetAction = CommonActions.reset({  index: 0,  routes: [{ name: "Pizza" }],  });  navigation.dispatch(resetAction);  wait(2000).then(() => setRefreshing(false));  }, []);  // Contient les lignes de commandes  const [order, setOrder] = useState<OrderLine[]>([]);  // Permet d'obtenir les commandes des pizzas, les id et leurs quantités  const updateValue = (id: number, quantity: number) => {  let newOrder: OrderLine[] = [];  order  .filter((pizza: OrderLine) => pizza.id !== id)  .map((pizza) => newOrder.push(pizza));  newOrder.push(new OrderLine(id, quantity));  setOrder(newOrder);  };  console.log(order);  return (  <>  <Header name="Passez votre commande" />  <View style={styles.container}>  <View>  {/\* Liste des pizzas \*/}  <PizzaList  pizzas={pizzas}  refreshing={refreshing}  onRefresh={onRefresh}  updateValue={updateValue}  />  </View>  {/\* FOOTER \*/}  <Footer pizzas={pizzas} order={order} />  </View>  </>  );  };  export default OrderScreen;  const styles = StyleSheet.create({  container: {  flex: 1,  alignItems: "center",  justifyContent: "center",  },  });  Ce code représente l'écran de commande d'une application mobile où les utilisateurs peuvent choisir des pizzas, ajouter des quantités et passer leur commande. Il utilise React Native, récupère les données des pizzas via un service, permet la mise à jour des quantités de pizzas sélectionnées et affiche un résumé de la commande dans un footer personnalisé.  import { StyleSheet, Text, View } from "react-native";  import React from "react";  import { useNavigation } from "@react-navigation/native";  import { NativeStackNavigationProp } from "@react-navigation/native-stack";  import { RouteParams } from "../Routes/RootNavigator";  import Header from "../components/Header/Header";  const Welcome = () => {  const navigation = useNavigation<NativeStackNavigationProp<RouteParams>>();  // Bouton pour aller vers la liste des pizzas  const GoPizza = () => {  navigation.navigate("Pizza");  };  return (  <>  <Header />  <View style={styles.container}>  <View style={styles.login}>  <Text style={styles.text}>Félicitation !</Text>  <Text style={styles.text}>Vous êtes prêt à commander .</Text>  <Text style={styles.new} onPress={GoPizza}>  Passez votre première commande  </Text>  </View>  </View>  </>  );  };  export default Welcome;  const styles = StyleSheet.create({  container: {  height: "100%",  alignItems: "center",  backgroundColor: "#3b438b",  width: "100%",  },  text: {  marginBottom: 50,  color: "#FFC921",  fontWeight: "bold",  fontSize: 50,  textAlign: "center",  },  login: {  marginTop: 60,  },  new: {  marginTop: "45%",  color: "#FFC921",  textDecorationLine: "underline",  fontSize: 30,  textAlign: "center",  },  });  Ce code représente un écran de bienvenue dans une application mobile où les utilisateurs sont félicités pour leur inscription. Il affiche un message de félicitations et propose un bouton pour accéder à la liste des pizzas. Il utilise React Native pour la création de l'interface utilisateur.  Pour les routes  import React from 'react'  import { createNativeStackNavigator } from '@react-navigation/native-stack';  import { NavigationContainer } from '@react-navigation/native';  import LoginPage from '../pages/LoginScreen';  import New from '../pages/InscriptionScreen';  import Bienvenue from '../pages/Welcome';  import Final from '../pages/Final';  import OrderScreen from '../pages/OrderScreen';  export type RouteParams = {  LoginPage: undefined;  New: undefined;  Bienvenue: undefined;  Pizza: undefined;  Final: undefined;  }  const Stack = createNativeStackNavigator<RouteParams>();  const Navigation = () => {  return (  <NavigationContainer>  <Stack.Navigator screenOptions={{headerShown: false}}>  <Stack.Screen name="LoginPage" component={LoginPage} options={{animation: 'slide\_from\_left'}}/>  <Stack.Screen name='Pizza' component={OrderScreen} />  <Stack.Screen name="New" component={New} options={{ animation: 'slide\_from\_right'}} />  <Stack.Screen name="Bienvenue" component={Bienvenue} />  <Stack.Screen name="Final" component={Final}/>  </Stack.Navigator>  </NavigationContainer>  );  }  export default Navigation  Ce code définit la structure de navigation d'une application mobile en utilisant React Navigation. Il configure une pile de navigation avec plusieurs écrans, notamment la page de connexion, l'écran de commande, l'inscription, la page de bienvenue et l'écran final. L'application démarre avec la page de connexion et permet la navigation entre ces écrans.  import LocalStorage from "../data/LocalStorage";  import LoginRequest from "../models/Loginrequest";  import User from "../models/User";  import LoginResponse from "../models/loginResponse";  class AuthenticationService {  static jwt: string;  // sert à accéder au back de l'application et permet de récuperer les informations présentes  // grace aux informations données dans le login. Ces informations sont stockées dans le body  // d'une requête de type POST, response est renvoyé sous forme de json.  static async call(login: LoginRequest): Promise<LoginResponse | undefined> {  const response = await fetch("http://192.168.0.20:8080/auth/signin", {  method: "POST",  body: JSON.stringify(login),  headers: { "Content-Type": "application/json" },  });  if (response.ok) {  return await response.json();  }  }  // sert de point d'entree, prend en paramètre un username et un password, promet de renvoyer un boolean  // dans un futur plus ou moins proche. On attend la réponse avant d'aller plus loin. On fait appelle  // à la methode call qui prend en paramètre un objet LoginRequest (qui prend en parametre ces username/password)  // elle sont assignées à response. On peut stocker le contenu de la réponse dans le localStorage, pour cela  // on a besoin d'une clée (string) et de la donnée (également un string, qui reprends la donnée de response)  // dans notre cas. On en profite pour ajoute un "boolean" pour le rechargement de la page.  static async login(username: string, password: string): Promise<boolean> {  await this.call(new LoginRequest(username, password)).then((response) => {  if (response !== undefined) {  LocalStorage.setToken(response.jwt);  LocalStorage.setExpiration(response.expiration);  LocalStorage.setRole(JSON.stringify(response.user));  LocalStorage.setId(JSON.stringify(response.user.id));  LocalStorage.setReloaded("false");  }  });  // On retourne la promesse de type boolean (le resolve) par la methode isAuthenticated au bout de  // 1000 ms  return new Promise((resolve) => {  setTimeout(() => {  resolve(this.isAuthenticated());  }, 1000);  });  }  //  //  static isAuthenticated(): boolean {  const expiration = LocalStorage.getExpiration();  const jwt = LocalStorage.getToken();  if (  expiration !== null &&  Date.parse(expiration) > Date.now() &&  jwt !== null  ) {  return jwt !== undefined;  } else {  LocalStorage.setToken("");  LocalStorage.setExpiration("");  LocalStorage.setRole("");  if (LocalStorage.getReloaded() !== "true") {  LocalStorage.setReloaded("true");  }  return false;  }  }  static async addUser(user: String): Promise<User> {  console.log(user);  return fetch(`http://192.168.0.20:8080/auth/signup`, {  method: "POST",  body: JSON.stringify(user),  headers: {  "Content-Type": "application/json",  Authorization: AuthenticationService.jwt,  },  })  .then((response) => response.json())  .catch((error) => this.handleError(error));  }  static handleError(error: Error): void {  console.error(error);  }  static getJwt(): any {  this.isAuthenticated();  return LocalStorage.getToken();  }  }  export default AuthenticationService;  Ce code représente un service d'authentification dans une application mobile. Il permet de se connecter, de vérifier l'authenticité de l'utilisateur, d'ajouter un utilisateur et de gérer le jeton d'authentification. Le service utilise des appels API pour interagir avec le back-end de l'application.  import Order from "../models/Order";  import AuthenticationService from "./AuthenticationService";  export default class OrderService {  // Récupère la méthode pour lier avec le JWT l'utilisateur et ça commande  static async userOrder(order: Order): Promise<any> {  console.log(order);  return fetch("http://192.168.0.20:8080/api/orders/", {  method: "POST",  body: JSON.stringify(order),  headers: {  "Content-Type": "application/json",  Authorization: AuthenticationService.getJwt(),  },  })  .then((response) => response.status)  .catch((error) => this.handleError(error));  }  static handleError(error: Error): void {  console.error(error);  }  }  Ce code représente un service pour passer des commandes d'utilisateurs dans une application mobile. Il envoie une commande au back-end de l'application en utilisant une requête POST avec un jeton d'authentification JWT pour authentifier l'utilisateur.  import Pizza from "../models/Pizza";  import User from "../models/User";  import Authentication from "./AuthenticationService";  export default class PizzaService {  static pizzas: Pizza[];  // Récupère toutes les pizzas de la base de données  static async getPizzas(): Promise<Pizza[]> {  return fetch("http://192.168.0.20:8080/api/pizzas", {  headers: { Authorization: Authentication.getJwt() },  })  .then((response) => response.json())  .catch((error) => this.handleError(error));  }  static isEmpty(data: Object): boolean {  return Object.keys(data).length === 0;  }  static handleError(error: Error): void {  console.error(error);  }  }  Ce code représente un service pour récupérer la liste des pizzas depuis une base de données via une requête HTTP, en utilisant un jeton d'authentification JWT pour l'authentification.  Dans helpers  const images: any = {  "margherita.jpg": require('../assets/img/pizzas/margherita.jpg'),  "4fromages.jpg" : require('../assets/img/pizzas/4fromages.jpg'),  "chevre-miel.jpg" : require('../assets/img/pizzas/chevre-miel.jpg'),  "chicken-bbq.jpg" : require('../assets/img/pizzas/chicken-bbq.jpg'),  "hawaienne.jpg" : require('../assets/img/pizzas/hawaienne.jpg'),  "nordique.jpg" : require('../assets/img/pizzas/nordique.jpg'),  "orientale.jpg" : require('../assets/img/pizzas/orientale.jpg'),  "pepperoni.jpg" : require('../assets/img/pizzas/pepperoni.jpg'),  "reine.jpg" : require('../assets/img/pizzas/reine.jpg'),  "texan-bbq.jpg" : require('../assets/img/pizzas/texan-bbq.jpg'),  "vegetarienne.jpg" : require('../assets/img/pizzas/vegetarienne.jpg'),  "campagnarde.jpg" : require('../assets/img/pizzas/campagnarde.jpg'),  }    export default images;  Ce code crée un objet `images` qui associe des noms de fichiers d'images de pizzas à leurs chemins de fichiers correspondants dans un projet React Native. Il est utilisé pour charger ces images dans l'application. Et finalement les composant  Ce composant React Native, `CardPizza`, affiche des informations sur une pizza, y compris son nom, sa description, une image, sa quantité et son prix. Les styles sont importés depuis un fichier externe, et l'image de la pizza est chargée à partir d'un objet `images` basé sur le nom de fichier. Le composant prend en charge la mise à jour de la quantité de pizzas à commander.  import { StyleSheet, Text, View, Image } from "react-native";  import React from "react";  import images from "../../helpers/images";  import Quantity from "../Quantity/Quantity";  import styles from "./cardPizzaStyle "; // Assurez-vous que le chemin d'importation est correct  function CardPizza(props: any) {  const { pizza, updateValue } = props;  const prix = pizza.price;  return (  <>  <View style={styles.card}>  {/\* INFORMATION PIZZA \*/}  <View style={styles.pizzaInfo}>  <View style={styles.image}>  <Image  source={images[pizza.image]}  style={{ height: "100%", width: "100%" }}  resizeMode="cover"  />  </View>  {/\* description Pizza \*/}  <View style={styles.contenu}>  <Text style={styles.nom}>{pizza.name}</Text>  <Text style={styles.info}>{pizza.description}</Text>  </View>  </View>  {/\* PRIX + COMMANDE \*/}  <View style={styles.prixQuantite}>  <View style={styles.boxPrixQuantite}>  <Text style={styles.text}>Quantité : </Text>  <Quantity id={pizza.id} updateValue={updateValue} />  </View>  <View style={styles.boxPrixQuantite}>  <Text style={styles.totalPrice}> {prix.toFixed(2)} € </Text>  </View>  </View>  </View>  </>  );  }  export default CardPizza; // Assurez-vous que le fichier de styles exporte par défaut votre composant  /\*  const styles = StyleSheet.create({  card: {  // CSS de la carte  height: 155,  width: "95%",  marginVertical: 10,  borderWidth: 1,  padding: 1,  alignSelf: "center",  },  pizzaInfo: {  // comporte les informations sur la pizza  flexDirection: "row",  height: "70%",  width: "100%",  },  image: {  // image de la pizza  height: "100%",  width: "30%",  },  contenu: {  // nom et description de la pizza  height: "100%",  width: "70%",  },  nom: {  marginLeft: 5,  fontWeight: "bold",  fontSize: 18,  },  info: {  margin: 5,  },  prixQuantite: {  // quantité + prix  height: "30%",  width: "100%",  flexDirection: "row",  alignItems: "center",  justifyContent: "space-between",  },  boxPrixQuantite: {  marginHorizontal: 10,  flexDirection: "row",  },  text: {  marginHorizontal: 5,  marginTop: 4.5,  marginRight: 5,  },  totalPrice: {  color: "green",  fontWeight: "bold",  },  });  \*/  Ce module définit un ensemble de styles CSS sous forme d'objets JavaScript pour le composant `CardPizza` ou d'autres composants similaires. Les styles sont exportés en tant qu'objet JavaScript pour être utilisés dans le composant.  import { StyleSheet } from "react-native";  const styles = StyleSheet.create({  card: {  // CSS de la carte  height: 155,  width: "95%",  marginVertical: 10,  borderWidth: 1,  padding: 1,  alignSelf: "center",  },  pizzaInfo: {  // comporte les informations sur la pizza  flexDirection: "row",  height: "70%",  width: "100%",  },  image: {  // image de la pizza  height: "100%",  width: "30%",  },  contenu: {  // nom et description de la pizza  height: "100%",  width: "70%",  },  nom: {  marginLeft: 5,  fontWeight: "bold",  fontSize: 18,  },  info: {  margin: 5,  },  prixQuantite: {  // quantité + prix  height: "30%",  width: "100%",  flexDirection: "row",  alignItems: "center",  justifyContent: "space-between",  },  boxPrixQuantite: {  marginHorizontal: 10,  flexDirection: "row",  },  text: {  marginHorizontal: 5,  marginTop: 4.5,  marginRight: 5,  },  totalPrice: {  color: "green",  fontWeight: "bold",  },  });  export default styles; | | |
|  | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **2. Précisez les moyens utilisés :** | | | |
|  | | | |
| J’ai développé l’application avec :   * Lucidchart * L’environnement de développement Visual Code * Android Studio * React native avec différents modules tel que : Axios | | | |
|  | | | |
|  | | | |
|  | | | |
| **3. Avec qui avez-vous travaillé ?** | | | |
|  | | | |
| J’ai travaillé seule | | | |
|  | | | |
|  | | |
|  | | | |
| **4. Contexte** | | | |
|  | |  | |
| Nom de l’entreprise, organisme ou association **** | | *idformation* | |
| Chantier, atelier, service | Mario pizza | | |
| Période d’exerciceDu : *07/08/2023*au :*11/08/2023* | | | |
|  | | |
|  | | |
| **5. Informations complémentaires** *(facultatif)* | | |
| Rien | | | |
|  | | | |
|  | | | |
|  | | | |

|  |  |  |
| --- | --- | --- |
| **Titres, diplômes, CQP, attestations de formation** | | |
|  | | |
| *(facultatif)* | | |
| **Intitulé** | | **Autorité ou organisme** | **Date** |
| **Bac Scientifique** | | **Lycée technique Rades** | 2011 |
| **Licence 3** | | **Ecole Nationale d’ingénieurs de Carthage** | 2014 |
|  | |  |  |
|  | |  |  |
|  | |  |  |
|  | |  |  |
|  | |  |  |
|  | |  |  |
|  | |  |  |

|  |
| --- |
| **Déclaration sur l’honneur** |
|  |
|  |

Sondes Ellouzi

Je soussigné(e) [prénom et nom] ,

déclare sur l’honneur que les renseignements fournis dans ce dossier sont exacts et que je suis l’auteur(e) des réalisations jointes.

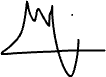
20/10/2023

LILLE

Fait à le

pour faire valoir ce que de droit.

Signature :



|  |
| --- |
| **Documents illustrant la pratique professionnelle** |
|  |
| *(facultatif)* |
| **Intitulé** |
| Cliquez ici pour taper du texte. |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

|  |
| --- |
| **Annexes** |
|  |
| *(Si le RC le prévoit)* |