****

**BENNOUI Feriel**

**ACAD C HEDIBEL Amira**

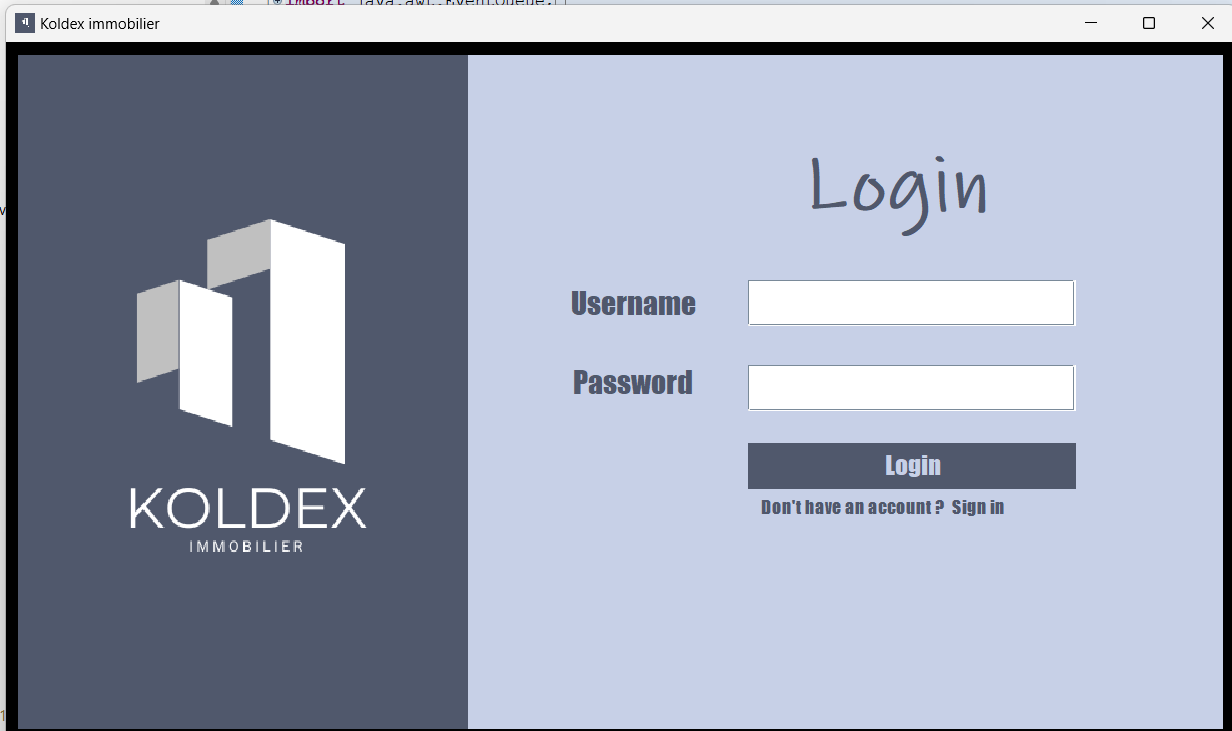
**BOUCHERIFI NASSIM**

**BENMOUSSA ZINEDINE**

**Plateforme Immobilière : Conception et Développement**

**Introduction:**

Notre application Java de gestion des biens immobiliers, développée sur l'environnement de développement intégré (IDE) Eclipse avec des interfaces créées à l'aide de WindowBuilder, offre une solution complète et intuitive pour faciliter la recherche, l'achat et la location de biens immobiliers en ligne. En mettant en relation directe les clients avec des agents immobiliers enregistrés, notre application simplifie et sécurise tout le processus immobilier, depuis la recherche de propriétés jusqu'à la finalisation des transactions et la génération de contrats.



*Éléments essentiels:*

-Les utilisateurs ont la possibilité de réaliser des recherches avancées afin de trouver des propriétés qui correspondent précisément à leurs préférences. L'emplacement, le type de bien (appartement, maison, local commercial, etc.), le budget, la superficie, le nombre de chambres, etc., sont autant de critères de recherche. Grâce à l'interface conviviale, il est facile de trier et de filtrer les résultats.

-Chaque propriété présentée sur l'application est accompagnée de descriptions détaillées et visuelles, comprenant les principales caractéristiques, les commodités disponibles, etc. Les images de grande qualité, les aident à prendre des décisions éclairées. Quand un utilisateur découvre une propriété qui suscite son intérêt, il a la possibilité de se mettre en relation directement avec des agents immobiliers grâce à l'application. Les agents ont la possibilité de fournir des renseignements supplémentaires, de planifier des visites physiques, et de répondre à toutes les interrogations des clients potentiels.

-La gestion des transactions en ligne est facilitée par l'utilisation de l'application qui permet de gérer toutes les transactions en ligne. La plateforme offre aux utilisateurs la possibilité de soumettre des offres, de négocier les conditions et de conclure des accords directement. La centralisation de tous les échanges et des négociations facilite le suivi et la gestion des transactions.

-Lorsque les parties impliquées ont conclu un accord, que ce soit pour la vente ou la location d'une propriété, notre application propose la possibilité de générer automatiquement des contrats. Cette fonctionnalité vise à rendre le processus de documentation plus simple et plus rapide en produisant des contrats complets qui comprennent toutes les clauses et les termes négociés au préalable. Les utilisateurs ont la possibilité de préciser les informations particulières de l'accord, comme le prix de vente ou de location, les modalités de paiement, les dates de début et de fin du contrat, ainsi que toutes les conditions particulières établies entre les parties.

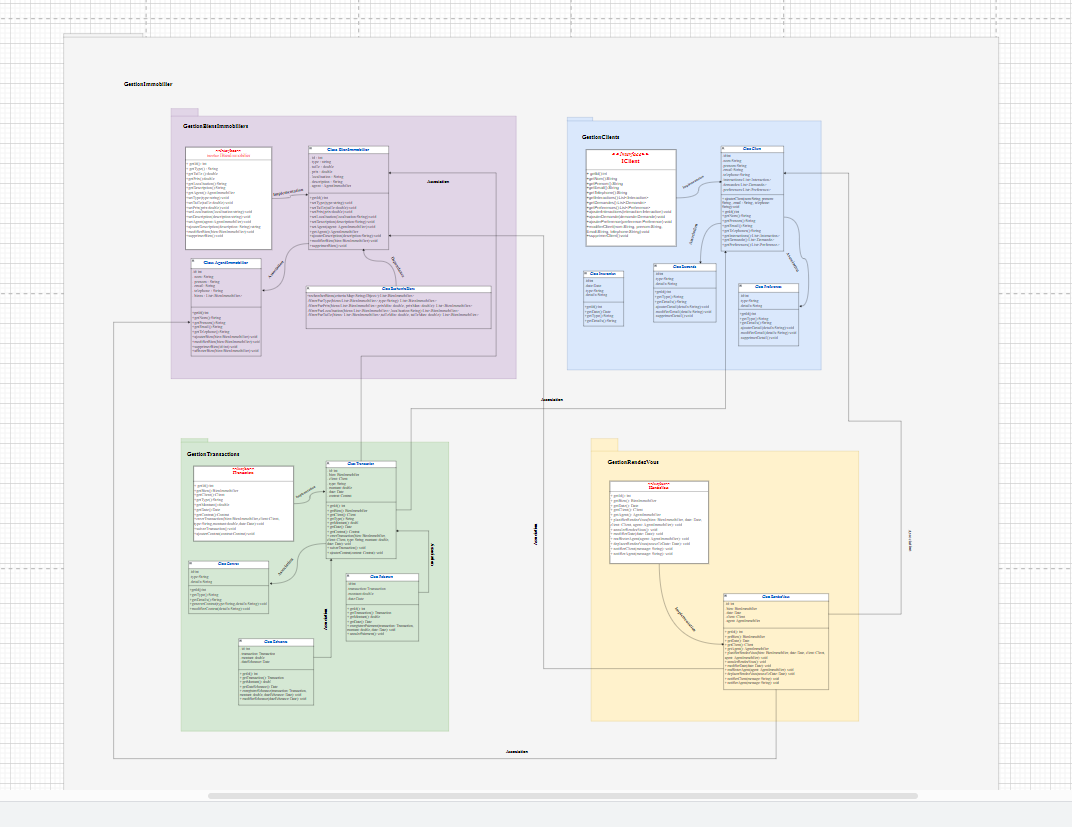
-La sécurité et la fiabilité des paiements pour les achats ou les locations de biens immobiliers sont assurées par notre application. Les utilisateurs peuvent effectuer des paiements directement grâce à l'application, en utilisant diverses méthodes de paiement sécurisées telles que les cartes de crédit, les virements bancaires ou les portefeuilles électroniques. Toutes les tâches sont réalisées en toute sécurité.

**Explorons maintenant les détails de la conception et du développement de l'application.**

**1/ CONCEPTION :**

***a/Appliquer draw.io à la modélisation UML:***

Le rapport présente une modélisation UML utilisant draw.io, une application en ligne qui permet de générer des diagrammes UML . L'interface conviviale de cette plateforme propose une gamme de formes et de modèles préétablis, ce qui simplifie la conception de diagrammes complexes tout en préservant la clarté et la lisibilité. La conception de la modélisation UML utilisant draw.io a été élaborée de manière à ce qu'elle soit claire et compréhensible à la fois pour les développeurs et les parties prenantes du projet. La position et l'annotation de chaque classe, interface et relation ont été minutieusement effectuées afin de refléter de manière précise les structures et les interactions du système de gestion des biens immobiliers.

* 

***b/ Utilisation des Classes et Méthodes dans le Contexte de l'Application:***

On va résumer les différents packages , classes et méthodes fournies :

**Package GestionBiensImmobiliers :**

**Interface IBienImmobilier** : Cette interface définit les méthodes permettant d'accéder et de manipuler les informations relatives à un bien immobilier.

**getId(): int:** Renvoie l'identifiant unique du bien immobilier.

**getType(): String**: Renvoie le type du bien immobilier (appartement, maison, terrain, etc.).

**getTaille(): double**: Renvoie la taille du bien immobilier en mètres carrés.

**getPrix(): double:** Renvoie le prix du bien immobilier.

**getLocalisation(): String:** Renvoie la localisation du bien immobilier.

**getDescription(): String:** Renvoie la description du bien immobilier.

**getAgent(): AgentImmobilier :** Renvoie l'agent immobilier en charge du bien. **ajouterDescription(description: String): void:** Ajoute une description au bien immobilier.

**setType(type: String): void:** Définit le type du bien immobilier.

**setTaille(taille: double): void**: Définit la taille du bien immobilier.

**setPrix(prix: double): void:** Définit le prix du bien immobilier. **setLocalisation(localisation: String): void:** Définit la localisation du bien immobilier.

**setDescription(description: String): void:** Définit la description du bien immobilier.

**setAgent(agent: AgentImmobilier): void:** Définit l'agent immobilier en charge du bien.

**modifierBien(bien: BienImmobilier): void:** Modifie les informations d'un bien immobilier existant.

**supprimerBien(): void:** Supprime le bien immobilier de la base de données.

**Classe BienImmobilier:** Représente un bien immobilier avec ses attributs et méthodes pour gérer ces informations.

**id: int:** Identifiant unique du bien immobilier.

**type: String**: Type du bien immobilier.

**taille: double:** Taille du bien immobilier.

**prix: double**: Prix du bien immobilier.

**localisation: String:** Localisation du bien immobilier.

**description: String:** Description du bien immobilier.

**agent: AgentImmobilier:** Agent immobilier en charge du bien. **ajouterDescription(description: String): void:** Ajoute une description au bien immobilier.

Méthodes getters et setters pour accéder et modifier les attributs du bien immobilier. **modifierBien(bien: BienImmobilier): void:** Modifie les informations d'un bien immobilier existant.

**supprimerBien(): void:** Supprime le bien immobilier de la base de données.

**Classe AgentImmobilier :** Représente un agent immobilier avec ses attributs et méthodes pour gérer ces informations.

**id: int:** Identifiant unique de l'agent immobilier.

**nom: String:** Nom de l'agent immobilier.

**prenom: String:** Prénom de l'agent immobilier.

**email: String:** Adresse email de l'agent immobilier.

**telephone: String:** Numéro de téléphone de l'agent immobilier.

**biens: List<BienImmobilier>:** Liste des biens immobiliers gérés par l'agent. Méthodes pour ajouter, modifier et supprimer des biens immobiliers, ainsi que pour accéder à ses informations.

**Classe RechercheBiens :** Fournit des méthodes pour rechercher des biens immobiliers en fonction de différents critères tels que le type, le prix, la localisation et la taille.

**rechercherBiens(criteria: Map<String, Object>): List<BienImmobilier>:** Recherche des biens immobiliers en fonction des critères spécifiés dans la carte.

**Package GestionClients :**

**Interface IClient :** Définit les méthodes permettant d'accéder et de manipuler les informations relatives à un client.

**getId(): int:** Renvoie l'identifiant unique du client.

**getNom(): String:** Renvoie le nom du client.

**getPrenom(): String:** Renvoie le prénom du client.

**getEmail(): String:** Renvoie l'email du client.

**getTelephone(): String:** Renvoie le numéro de téléphone du client.

**getInteractions(): List<Interaction>:** Renvoie la liste des interactions du client. **getDemandes(): List<Demande>:** Renvoie la liste des demandes du client. **getPreferences(): List<Preference>:** Renvoie la liste des préférences du client. **ajouterInteraction(interaction: Interaction): void:** Ajoute une interaction au client.

**ajouterDemande(demande: Demande): void:** Ajoute une demande au client. **ajouterPreference(preference: Preference): void:** Ajoute une préférence au client. **modifierClient(nom: String, prenom: String, email: String, telephone: String): void:** Modifie les informations du client.

**supprimerClient(): void:** Supprime le client de la base de données.

**Classe Client :** Représente un client avec ses attributs et méthodes pour gérer ces informations.

**id: int:** Identifiant unique du client.

**nom: String:** Nom du client.

**prenom: String:** Prénom du client.

**email: String:** Adresse email du client.

**telephone: String:** Numéro de téléphone du client.

**interactions: List<Interaction>:** Liste des interactions du client.

**demandes: List<Demande>:** Liste des demandes du client.

**preferences: List<Preference>:** Liste des préférences du client.

Méthodes pour ajouter, modifier et supprimer des interactions, des demandes et des préférences, ainsi que pour accéder à ses informations.

**Classes Interaction, Demande, Preference** Représentent respectivement une interaction avec un client, une demande émise par un client et une préférence exprimée par un client. Attributs pour stocker les informations pertinentes. Méthodes pour accéder et modifier ces informations.

**Package GestionTransactions:**

**Interface ITransaction** : Définit les méthodes permettant d'accéder et de manipuler les informations relatives à une transaction immobilière.

**getId(): int**: Renvoie l'identifiant unique de la transaction.

**getBien(): BienImmobilier:** Renvoie le bien immobilier lié à la transaction. **getClient(): Client:** Renvoie le client associé à la transaction.

**getType(): String:** Renvoie le type de la transaction.

**getMontant(): double:** Renvoie le montant de la transaction.

**getDate(): Date:** Renvoie la date de la transaction.

**getContrat(): Contrat:** Renvoie le contrat associé à la transaction. **creerTransaction(bien: BienImmobilier, client: Client, type: String, montant: double, date: Date): void:** Crée une nouvelle transaction.

**suivreTransaction(): void**: Permet de suivre l'état de la transaction. **ajouterContrat(contrat: Contrat): void:** Associe un contrat à la transaction. Classes Transaction, Contrat, Paiement, Echeance Représentent respectivement une transaction immobilière, un contrat généré dans le cadre d'une transaction, un paiement associé à une transaction et une échéance de paiement.

**Package GestionRendezVous :**

**Interface IRendezVous :**

Définit les méthodes permettant de gérer les rendez-vous entre les clients, les agents immobiliers et les biens immobiliers.

**getId(): int:** Renvoie l'identifiant unique du rendez-vous.

**getBien(): BienImmobilier:** Renvoie le bien immobilier associé au rendez-vous.

**getDate(): Date:** Renvoie la date du rendez-vous.

**getClient(): Client:** Renvoie le client associé au rendez-vous.

**getAgent(): AgentImmobilier:** Renvoie l'agent immobilier associé au rendez-vous.

**planifierRendezVous(bien: BienImmobilier, date: Date, client: Client, agent: AgentImmobilier): void:** Planifie un rendez-vous entre un client, un agent immobilier et un bien immobilier.

**annulerRendezVous(): void:** Annule le rendez-vous.

**modifierDate(date: Date): void**: Modifie la date du rendez-vous.

**reaffecterAgent(agent: AgentImmobilier): void:** Réaffecte un autre agent immobilier au rendez-vous.

**deplacerRendezVous(nouvelleDate: Date): void:** Déplace le rendez-vous à une nouvelle date.

**notifierClient(message: String): void:** Notifie le client avec un message spécifié.

**notifierAgent(message: String): void:** Notifie l'agent immobilier avec un message spécifié.

**Classe RendezVous :** Représente un rendez-vous entre un client, un agent immobilier et un bien immobilier.

**id: int:** Identifiant unique du rendez-vous.

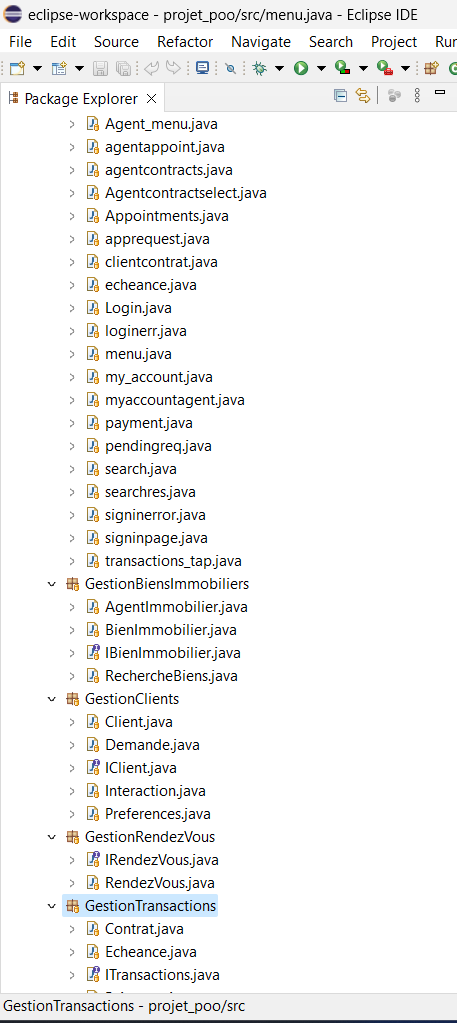
**bien: BienImmobilier:** Bien immobilier associé au rendez-vous.

**date: Date:** Date du rendez-vous.

**client: Client:** Client associé au rendez-vous.

**agent: AgentImmobilier:** Agent immobilier associé au rendez-vous.

Méthodes pour planifier, annuler, modifier et déplacer le rendez-vous, ainsi que pour notifier le client et l'agent immobilier.

****

**Conclusion:** Ce système de gestion immobilière permet de gérer de manière efficace et centralisée toutes les interactions liées aux biens immobiliers, aux clients, aux transactions et aux rendez-vous. Chaque classe et interface est conçue pour fournir des fonctionnalités spécifiques et faciliter la gestion globale du système.

**Maintenant apres avoir finie de mettre le plan permettant de mettre notre application sur pied attaquant nous au developpement**

**2/Developpement :**

***a-Utilisation de l'approche UML de conception :***

Le code de l'application a été organisé par notre développeur en utilisant la structure de conception UML fournie comme guide.

La structure UML comprend chaque package qui regroupe différentes fonctionnalités, comme la gestion des biens immobiliers, des clients, des transactions et des rendez-vous. Des entités et des fonctionnalités spécifiques de l'application sont représentées par des classes et des interfaces dans chaque package, comme les biens immobiliers, les clients, les transactions, etc.

Les classes correspondantes mettent en œuvre les méthodes définies dans les interfaces afin de fournir les fonctionnalités nécessaires à l'application.

***b-Intégration d'interfaces graphiques avec WindowBuilder :***

Tout d'abord, notre programmeur a ajouté l'extension WindowBuilder à Eclipse pour faciliter la conception d'interfaces visuelles.

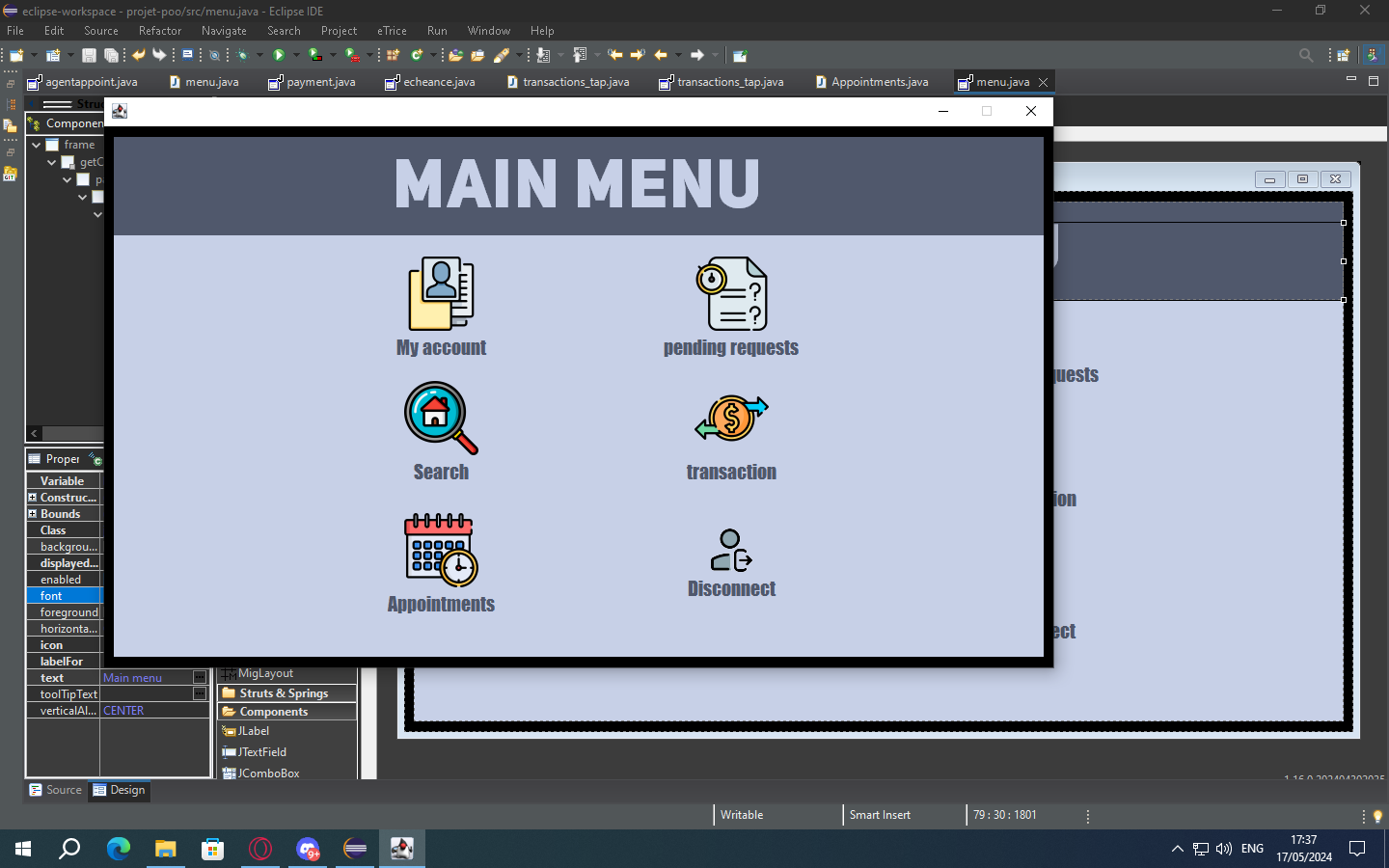
L'extension WindowBuilder d'Eclipse permet de créer des interfaces utilisateur visuelles en utilisant Swing et SWT.Après avoir ajouté WindowBuilder à notre projet Java, notre développeur a développé de nouvelles classes d'interfaces graphiques.

En général, on élargit ces classes à partir des classes JFrame ou JPanel, en fonction des exigences de l'interface utilisateur.Lorsque notre développeur a ouvert une classe d'interface graphique dans l'éditeur Eclipse, il a choisi de se diriger vers l'onglet Design au sein de WindowBuilder.

Dès lors, il a eu la possibilité de déplacer et de déplacer divers éléments graphiques à partir de la palette dans la fenêtre de conception, comme des boutons, des champs de texte, des menus déroulants, et ainsi de suite.

Cela offre la possibilité de concevoir visuellement l'interface utilisateur de manière intuitive, sans avoir à rédiger manuellement chaque ligne de code.Après avoir créé l'interface graphique, notre développeur a inclus des actions propres à chaque composant, telles que des méthodes Java pour gérer les événements de clic sur un bouton, de sélection d'un élément dans un menu, etc.

Cette association est généralement réalisée en cliquant deux fois sur un composant dans la fenêtre de conception et en ajoutant le code Java correspondant dans la méthode générée.Une fois que l'interface graphique a été développée et liée aux fonctionnalités de l'application, notre développeur a réalisé des tests pour vérifier que l'interface répondait aux attentes. On a apporté des ajustements en fonction des exigences pour garantir une expérience utilisateur fluide et intuitive.

* + 

***c-Implémentation des fonctionnalités de l'application :***

En se basant sur la conception UML, notre développeur a mis en place chaque fonctionnalité de l'application en créant les classes Java appropriées et en rédigeant le code requis pour les réaliser. Par exemple, dans le domaine de la gestion des biens immobiliers, diverses techniques ont été mises en place, telles que l'ajout de biens, la modification de biens, la suppression de biens, etc., en utilisant les interfaces et les classes définies dans le package **GestionBiensImmobiliers.**

Les interfaces graphiques développées avec **WindowBuilder** sont reliées aux méthodes mises en place, de manière à ce que lorsque l'utilisateur interagit avec l'interface graphique, les actions correspondantes sont activées dans le code Java respectivement.

***d-Vérification des conditions et manipulation des informations***

Toutes les conditions définies dans le UML ont été prises en compte par notre développeur dans le code Java pour assurer le bon déroulement de l'application. Par exemple, lorsque l'application crée une nouvelle transaction immobilière, elle vérifie les conditions telles que la disponibilité du bien, la demande du client, etc., avant de procéder à la transaction.

**Conclusion :**

En résumé, notre développeur a utilisé l'approche UML pour organiser le code de l'application, intégré WindowBuilder pour simplifier la conception des interfaces graphiques, et implémenté les fonctionnalités essentielles en suivant le modèle UML. Cela a permis de garantir une gestion efficace des biens immobiliers, des clients, des transactions et des rendez-vous, tout en assurant un fonctionnement sûr de l'application.

***REDACTION :***

***a- Le PowerPoint :***

Nous avons mis en place une approche afin de présenter notre application immobilière de manière persuasive et bien expliquée .

Dans un premier temps, nous avons élaboré un Powerpoint complet, comprenant des diapositives minutieusement élaborées, afin de mettre en évidence l'utilité et les avantages spécifiques de notre application.

Toutes les diapositives ont été conçues afin de répondre à une question essentielle : pourquoi notre application est-elle la meilleure option disponible .

Les principales caractéristiques de notre application ont été exposées en détail, mettant en évidence ses fonctionnalités essentielles comme la création automatique de contrats, les paiements sécurisés et la recherche facile de biens immobiliers. Chaque caractéristique a été décrite de manière claire, ce qui facilite la compréhension de quant à l'étendue des bénéfices proposés par notre application qui est la solution.

* 

***b- Le rapport :***

De plus, nous avons rédigé un rapport complet et détaillé décrivant le processus complet de développement de notre application.

Ce rapport fournit une vue d'ensemble complète du cheminement suivi, des ressources mobilisées et des défis surmontés pour que notre application voie le jour.

***Conclusion :*** En conclusion, la rédaction du PowerPoint et du rapport a bien mis en avant l'application et son développement, offrant une vue claire de son utilité et des étapes nécessaires à sa création.

***CONCLUSION GLOBALE :***

En résumé, ce rapport a retracé l'évolution de notre application de gestion immobilière depuis sa création. Initialement, la modélisation UML a été employée afin de guider la structure du code et la mise en place des fonctionnalités. En incorporant WindowBuilder, la création d'interfaces graphiques intuitives a été simplifiée. Ensuite, nous avons étudié de manière approfondie chaque aspect de l'application, mettant en évidence son utilité et ses principales caractéristiques. En fin de compte, la création du PowerPoint et du rapport a rendu notre application persuasive, mettant en avant son potentiel et les ressources nécessaires pour sa mise en œuvre. Dans l'ensemble, ce rapport reflète notre volonté de proposer une solution immobilière solide et originale, prête à répondre aux besoins de nos utilisateurs.