Documentazione UninaSwap

Com’è stato realizzato l’applicativo UninaSwap?

L’applicativo è stato realizzato in linguaggio Java (con JRE 23), abbiamo utilizzato il framework JavaFX per l’implementazione dell’interfaccia utente (GUI) e per la personalizzazione dei componenti GUI è stato utilizzato un foglio di stile CSS (Cascading Style Sheets).

Dal punto di vista architetturale è stato adottato un modello basato sull’euristica BCE (Boundary-Control-Entity) integrato con il pattern DAO (Data Access Object) con lo scopo di creare una suddivisione delle responsabilità nei vari livelli dell’applicativo:

- Interfaccia grafica e logica di presentazione (Boundary)
- Logica di business (Control)
- Logica di persistenza (DAO)
- Dominio (Entity)

Per la gestione dei dati è stato utilizzato il RDBMS (Relational Database Management System) PostgreSQL e la logica di connessione è stata incapsulata nella classe ConnessioneDB, la quale mediante un file `db.properties` che contiene le credenziali di accesso al DBMS (che abbiamo incluso nel .gitignore) permette la connessione al DBMS e l’esecuzione di query.

Inoltre, in diverse classi abbiamo utilizzato un design pattern Singleton, ovvero viene istanziato un unico oggetto di quella classe in modo da riutilizzarlo per tutta la durata del programma.

Abbiamo utilizzato questo design pattern per un motivo ben preciso:

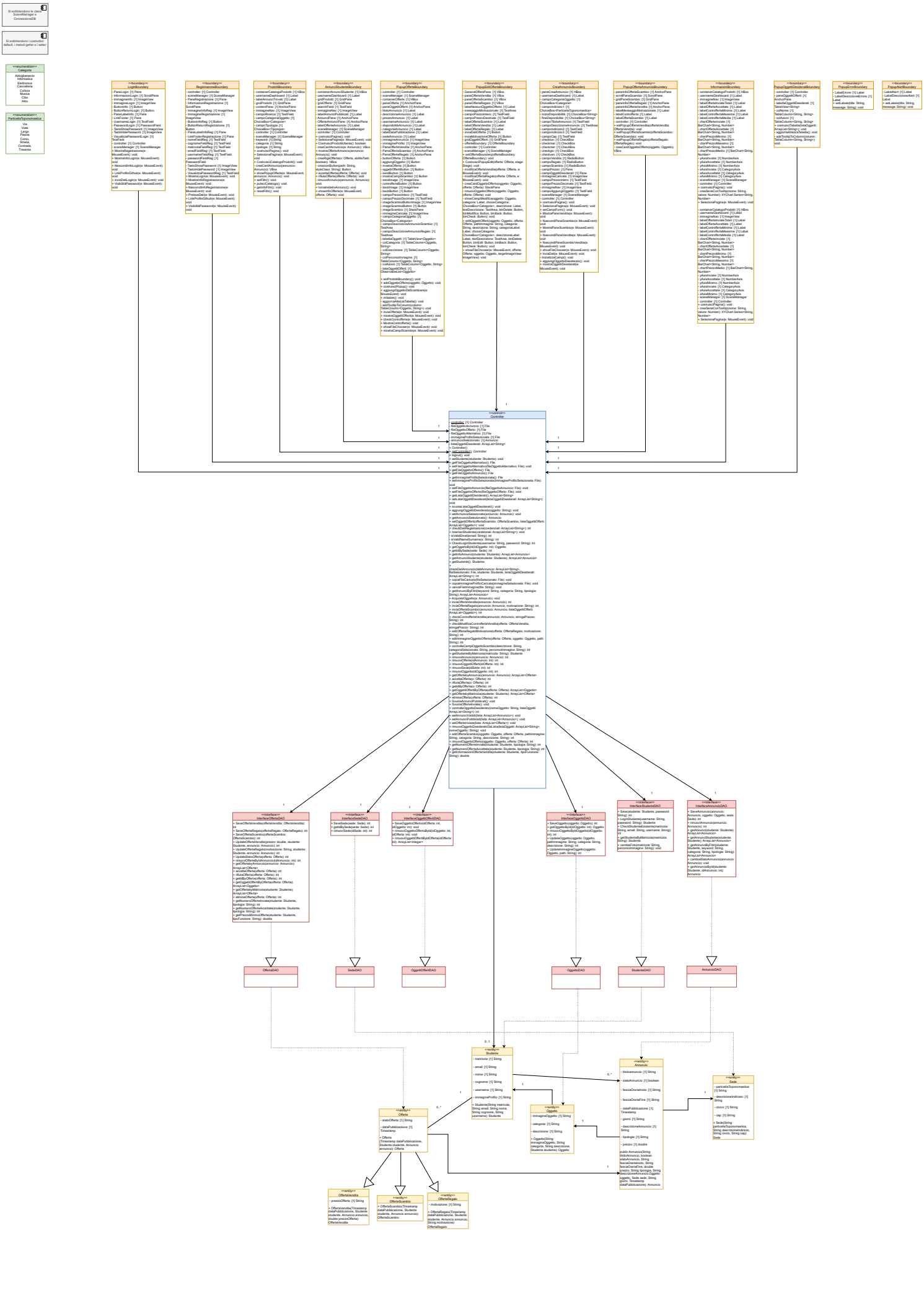
- Per ConnessioneDB questa scelta ci ha permesso di risparmiare tempo e memoria poichè le operazioni di reperimento e di scrittura dei dati sul db sono molto frequenti perciò si effettuano molte connnessioni
- Per Controller il discorso è analogo perchè un controller resta unico per tutta la durata del programma, quindi l’unicità dell’istanza è utile per rappresentare in maniera corretta la sessione relativa all’utente loggato
- Per SceneManager è stato utile renderlo Singleton perchè è un servizio statico che deve solo permettere il cambio da una scena all’altra dell’interfaccia grafica

Struttura del programma

Nome sezione	Dove si può andare	Cosa si può fare
Login	Registrazione	Si possono visualizzare le informazioni di login ed effettuare il login inserendo le credenziali negli appositi campi di input

Registrazione	Login	Si possono visualizzare le informazioni di registrazione ed effettuare la registrazione inserendo i dati negli appositi campi di input
Prodotti	Crea annuncio I tuoi annunci Offerte Informazioni Profilo	Si possono visualizzare gli annunci attivi degli altri studenti, filtrarli per keyword, tipologia e categoria Inoltre si possono visualizzare le informazioni di ciasun annuncio ed effettuare un offerta ad ognuno di essi se non è stato già fatto in precedenza
Crea annuncio	Prodotti I tuoi annunci Offerte Informazioni Profilo	Si può creare un annuncio inserendo i dati negli appositi campi di input
I tuoi annunci	Prodotti Crea annuncio Offerte Informazioni Profilo	Si possono visualizzare gli annunci dello studente loggato, è possibile eliminarli e visualizzare le offerte ricevute, ed è possibile anche visualizzare le informazioni delle offerte ricevute e accettarle/rifiutarle
Offerte	Prodotti Crea annuncio I tuoi annunci Informazioni Profilo	Si possono visualizzare le offerte inviate ed è possibile modificarle/eliminarle se non accettate
Informazioni	Prodotti Crea annuncio I tuoi annunci Offerte Profilo	Si possono visualizzare i grafici riguardo il numero totale di offerte inviate suddivise per tipologia, il numero di offerte accettate per tipologia e per offerte accettate, il costo minimo, massimo e medio di offerte di vendita accettate
Profilo	Prodotti Crea annuncio I tuoi annunci Offerte Informazioni Login	Si possono visualizzare le informazioni personali e si può fare il logout

Class diagram


Si sottintendono le classi SceneManager e ConnessioneDB
Si sottintendono i costruttori default, i metodi getter e i setter
<<enumeration>>
Categorie
Abbigliamento
Informatica
Libri
Cancelleria
Cultura
Musica
Cibo
Altro
<<enumeration>>
ParticelleToponomastica
Via
Viale
Largo
Piazza
Corso,
Strada,
Contrada,
Traversa
<<boundary>>
LoginBoundary
- PaneLogin: [1] Pane
- InformazioniLogin: [1] ScrollPane
- ImmaginiInfo: [3] ImageView
- ImmaginiLogin: [1] ImageView
- ButtonInfo: [1] Button
- ButtonReturnLogin: [1] Button
- PaneLabelInfo: [1] Pane
- LabelInfo: [1] Label
- UsernameLogin: [1] TextField
- PasswordLogin: [1] PasswordField
- TestoShowPassword: [1] ImageView
- TestoHidePassword: [1] ImageView
- VisualizzaPasswordLogin: [1] TextField
- LoginButton: [1] Controller
- sceneManager: [1] SceneManager
+ MostraRegistrazione(e: MouseEvent): void
+ MostraInfoLogin(e: MouseEvent): void
+ NascondiInfoLogin(e: MouseEvent): void
+ LinkProfiloGithub(e: MouseEvent): void
+ InviaDatiLogin(e: MouseEvent): void
+ VisibilitàPassword(e: MouseEvent): void
<<boundary>>
RegistrazioneBoundary
- controller: [1] Controller
- sceneManager: [1] SceneManager
- PaneRegistrazione: [1] Pane
- InformazioniRegistrazione: [1] ScrollPane
- ImmaginiInfoReg: [1] ImageView
- ImmaginiRegistrazione: [1] ImageView
- ButtonInfoReg: [1] Button
- ButtonReturnRegistrazione: [1] Button
- PaneLabelInfoReg: [1] Pane
- LinkFotoRegistrazione: [1] Pane
- nomeFieldReg: [1] TextField
- cognomeFieldReg: [1] TextField
- matricolaFieldReg: [1] TextField
- emailFieldReg: [1] TextField
- usernameFieldReg: [1] TextField
- passwordFieldReg: [1] PasswordField
- TestoShowPassword: [1] ImageView
- TestoHidePassword: [1] ImageView
- VisualizzaPasswordReg: [1] TextField
+ MostraLogin(e: MouseEvent): void
+ MostraInfoRegistrazione(e: MouseEvent): void
+ NascondiInfoRegistrazione(e: MouseEvent): void
+ PreleviaDatiIn(e: MouseEvent): void
+ LinkProfiloGithub(e: MouseEvent): void
+ VisibilitàPassword(e: MouseEvent): void
<<boundary>>
ProdottiBoundary
- containerCatalogoProdotti: [1] HBox
- usernameDashboard: [1] Label
- labelAnnunciTrovati: [1] Label
- gridProdotti: [1] GridPane
- contentPane: [1] AnchorPane
- immagineNav: [1] ImageView
- campoRicerca: [3] TextField
- ricercaCategoria: [1] ComboBox<Categorie>
- campoTipologia: [1] ChoiceBox<Tipologie>
- controller: [1] Controller
- sceneManager: [1] SceneManager
- keyword: [1] String
- categoria: [1] String
- tipologia: [1] String
+ costruisciPagina(): void
+ SelezionaPagina(e: MouseEvent): void
+ CostruisciCatalogoProdotti(): void
+ creaCardAnnuncio(annuncio: Annuncio): void
+ showPopupOfferte(e: MouseEvent, annuncio: Annuncio): void
+ setFitH(): void
+ pulisciCatalogo(): void
+ getIdSFiltri(): void
+ resetFiltri(): void
<<boundary>>
AnnunciStudenteBoundary
- containerAnnunciStudente: [1] HBox
- usernameDashboard: [1] Label
- gridProdotti: [1] GridPane
- gridOfferte: [1] GridPane
- searchField: [1] TextField
- immagineNav: [1] ImageView
- labelAnnunciPubblicati: [1] Label
- labelOfferteRicevute: [1] Label
- OfferteAnnunciPane: [1] AnchorPane
- labelOfferteAnnuncio: [1] Label
- sceneManager: [1] SceneManager
- controller: [1] Controller
+ costruisciPagina(): void
+ SelezionaPagina(e: MouseEvent): void
+ CostruisciAnnunciStudente(): boolean
+ creaCardAnnuncio(a: Annuncio): VBox
+ mostraOfferteAnnuncio(annuncio: Annuncio): void
+ creaRigaOfferta(o: Offerta, abilitaTasti: boolean): HBox
+ creaIconBottone(path: String, tooltip: String): Button
+ accettaOfferta(offerta: Offerta): void
+ rifiutaOfferta(offerta: Offerta): void
+ rimuoviAnnuncio(annuncio: Annuncio): void
+ modificaInteroAnnuncio(): void
+ chiudiInfoOfferta(e: MouseEvent, offerta: Offerta): void
<<boundary>>
PopupOfferteBoundary
- controller: [1] Controller
- sceneManager: [1] SceneManager
- containerOfferte: [1] HBox
- paneOfferte: [1] AnchorPane
- paneOggettiOfferti: [2] AnchorPane
- titoloAnnuncio: [1] Label
- descrizioneAnnuncio: [1] Label
- prezzoAnnuncio: [1] Label
- usernameAnnuncio: [1] Label
- disponibilitàAnnuncio: [1] Label
- categoriaAnnuncio: [1] Label
- labelDataPubblicazione: [1] Label
- nextAnnuncio: [1] Label
- immagineAnnuncio: [6] ImageView
- paneOfferteRicevute: [1] VBox
- PaneOfferteVendita: [1] AnchorPane
- PaneOfferteScambio: [1] AnchorPane
- PaneOfferteRegalo: [1] AnchorPane
- buttonOfferta: [1] Button
- aggiungiOggetto: [1] Button
- mostraOfferte: [1] Button
- sendButton: [1] Button
- sendBullon: [1] Button
- mostraCampiScambio: [1] Button
- sendImage: [1] ImageView
- controlloreBullon: [1] Button
- backImage: [1] ImageView
- backBullon: [1] Button
- campoPrezzoIntero: [1] TextField
- campoPrezzoDecimale: [1] TextField
- imageScambioButtonImage: [1] ImageView
- imageScambioBullon: [1] Button
- imageScambio: [1] StackPane
- imageScambioImage: [1] ImageView
- campoCategoriaOggetto: [1] ChoiceBox<Categorie>
- campoDescrizioneAnnuncioScambio: [1] TextArea
- campoDescrizioneAnnuncioRegalo: [1] TextArea
- labelNoOggetti: [1] TableView<Oggetto>
- colCategorie: [1] TableColumn<Oggetto, String>
- colDescrizione: [1] TableColumn<Oggetto, String>
- colPrimoscimmagine: [1] TableColumn<Oggetto, String>
- colOggettiOfferti: [1] ObservableList<Oggetto>
+ setProdottiBoundary(): void
+ caricaOggettiOfferti(o: Oggetto): void
+ costruisciPagina(): void
+ aggiungiOggettoDaScambiareja(e: MouseEvent): void
+ initialize(): void
+ aggiornaAnnunciTabelle(): void
- addTooltipToColumn(column: TableColumn<Oggetto, String>): void
+ InviaOfferta(e: MouseEvent): void
+ mostraOggettiOfferti(e: MouseEvent): void
+ checkControlRefacta: MouseEvent): void
+ MostraControlRefacta(): void
+ showFileChooser(e: MouseEvent): void
+ mostraCampiScambio(e: MouseEvent): void
<<boundary>>
PopupGUIOfferteBoundary
- GeneraOfferePane: [1] VBox
- paneOfferteVendita: [1] VBox
- paneOfferteScambio: [1] VBox
- paneOfferteRegalo: [1] VBox
- labelAnnuncioOggettoOfferto: [1] Label
- messaggioMotivazionale: [1] TextArea
- campoPrezzoIntero: [1] TextField
- campoPrezzoDecimale: [1] TextField
- labelOfferteScambio: [1] Label
- labelOfferteVendita: [1] Label
- labelOfferteRegalo: [1] Label
- invalidDataOfferta: [1] Button
- invalidMotivazioneOfferta: [1] Button
- gridOggettiOfferti: [1] GridPane
- controller: [1] PopupGUIOfferteBoundary
- controller: [1] Controller
- sceneManager: [1] SceneManager
+ setOfferteBoundary(offerteBoundary: OfferteBoundary): void
+ Costruisci(e: MouseEvent, offerta: Offerta, stage: Stage): void
+ GeneraOfferteVendita(offerta: Offerta, e: MouseEvent): void
+ modificaOfferteRegalo(offerta: Offerta, e: MouseEvent): void
+ creaCardOggettoOfferto(oggetto: Oggetto, offerta: Offerta): void
+ rimuoviOggettoOfferto(oggetto: Oggetto, offerta: Offerta): void
+ showCampiModificaOggetto: Oggetto, categoria: Label, choiceCategoria: ChoiceBox<Categorie>, descrizione: Label, textDescrizione: TextArea, btnDelete: Button, btnSalva: Button, btnAnnulla: Button, btnCheck: Button): void
+ editOggettiOfferti(oggetto: Oggetto, offerta: Offerta, pathImmagine: String, categoria: String, descrizione: String, categoriaLabel: Label, choiceCategoria: ChoiceBox<Categorie>, descrizioneLabel: Label, textDescrizione: TextArea, btnDelete: Button, btnSalva: Button, btnAnnulla: Button, btnCheck: Button): void
+ showFileChooser(e: MouseEvent, offerta: Offerta, oggetto: Oggetto, targetImageView: ImageView): void
<<boundary>>
CreaAnnuncioBoundary
- paneCreaAnnuncio: [1] VBox
- usernameDashboard: [1] Label
- campoCategoriaOggetto: [1] ChoiceBox<Categorie>
- campoIndirizzo1: [1] ChoiceBox<ParticelleToponomastica>
- mzioDisponibilità: [1] ChoiceBox<String>
- campoIndirizzo2: [1] TextField
- campoTitoloAnnuncio: [1] TextField
- campoDescrizioneAnnuncio: [1] TextArea
- campoIndirizzo3: [1] TextField
- campoIndirizzo3: [1] TextField
- campoCap: [1] TextField
- checkbox1: [1] CheckBox
- checkbox2: [1] CheckBox
- checkbox3: [1] CheckBox
- checkgio: [1] CheckBox
- checkven: [1] CheckBox
- campoVendita: [1] RadioButton
- campoRegalo: [1] RadioButton
- campoScambio: [1] RadioButton
- campoGratis: [1] RadioButton
- campoOggettiDesiderati: [1] Pane
- immaginiCaricate: [1] ImageView
- campoPrezzoIntero: [1] TextField
- campoPrezzoDecimale: [1] TextField
- campoCittà: [1] TextField
- campoAggiungiOggetto: [1] TextField
- sceneManager: [1] SceneManager
- controller: [1] Controller
+ costruisciPagina(): void
+ SelezionaPagina(e: MouseEvent): void
+ setCampiForm(): void
+ MostraPaneVendita(e: MouseEvent): void
+ NascondiPaneScambio(e: MouseEvent): void
+ MostraPaneScambio(e: MouseEvent): void
+ NascondiPaneVendita(e: MouseEvent): void
+ NascondiPaneScambioVendita(e: MouseEvent): void
+ showFileChooser(e: MouseEvent): void
+ InviaDati(e: MouseEvent): void
+ initializeCampi(): void
+ aggiungiOggettoDesiderato(s): void
+ aggiungiOggettiDesiderati(e: MouseEvent): void
<<boundary>>
PopupOfferteAnnunciBoundary
- scrollPaneScambio: [1] ScrollPane
- gridPaneScambio: [1] GridPane
- paneInfoOfferteRegalo: [1] AnchorPane
- paneInfoOfferteVendita: [1] AnchorPane
- labelMessaggioMotivazionale: [1] Label
- labelControlRefa: [1] Label
- controller: [1] Controller
- controller: [1] Controller
+ setPopupOfferteVendita(offerteVenditaOfferteVendita): void
+ setPopupOfferteScambio(offerteScambio: OfferteScambio): void
+ setPopupOfferteRegalo(offerteRegalo: OfferteRegalo): void
+ creaCardOggettoOfferto(oggetto: Oggetto): VBox
<<boundary>>
InformazioniBoundary
- containerCatalogoProdotti: [1] HBox
- usernameDashboard: [1] Label
- immagineNav: [1] ImageView
- labelOfferteInviate Totali: [1] Label
- labelOfferteAccettate: [1] Label
- labelControlRefaMinima: [1] Label
- labelControlRefaMassima: [1] Label
- labelOfferteInviate Totali: [1] Label
- chartOfferteInviate: [1] BarChart<String, Number>
- chartOfferteAccettate: [1] BarChart<String, Number>
- chartPrezzoMinimo: [1] BarChart<String, Number>
- chartPrezzoMedio: [1] BarChart<String, Number>
- yAxisInviate: [1] NumberAxis
- yAxisAccettate: [1] NumberAxis
- yAxisMinimo: [1] NumberAxis
- yAxisMedio: [1] NumberAxis
- xAxisAccettate: [1] CategoryAxis
- xAxisMinimo: [1] CategoryAxis
- sceneManager: [1] SceneManager
- controller: [1] Controller
+ costruisciPagina(): void
+ creaSerieConTooltip(nome: String, valore: Number): XYChart.Series<String, Number>
+ SelezionaPagina(e: MouseEvent): void
- containerCatalogoProdotti: [1] HBox
- usernameDashboard: [1] Label
- immagineNav: [1] ImageView
- labelOfferteInviate Totali: [1] Label
- labelOfferteAccettate: [1] Label
- labelControlRefaMinima: [1] Label
- labelControlRefaMassima: [1] Label
- labelControlRefaMedia: [1] Label
- chartOfferteInviate: [1] BarChart<String, Number>
- chartOfferteAccettate: [1] BarChart<String, Number>
- chartPrezzoMinimo: [1] BarChart<String, Number>
- chartPrezzoMassimo: [1] BarChart<String, Number>
- chartPrezzoMedio: [1] BarChart<String, Number>
- yAxisInviate: [1] NumberAxis
- yAxisAccettate: [1] NumberAxis
- yAxisMinimo: [1] NumberAxis
- xAxisInviate: [1] CategoryAxis
- xAxisAccettate: [1] CategoryAxis
- xAxisMinimo: [1] CategoryAxis
- sceneManager: [1] SceneManager
- controller: [1] Controller
+ costruisciPagina(): void
+ creaSerieConTooltip(nome: String, valore: Number): XYChart.Series<String, Number>
+ SelezionaPagina(e: MouseEvent): void
<<boundary>>
PopupDettDesideratiBoundary
- controller: [1] Controller
- paneOggettiOfferti: [1] AnchorPane
- labelloOggettiDesiderati: [1] TableView<String>
- colNome: [1] TableColumn<String, String>
- colDescrizione: [1] TableColumn<String, String>
+ costruisciTabelleListaOggetti: ArrayList<String>): void
+ aggiornaVetrinaTabelle(): void
- addTooltipToColumn(column: TableColumn<String, String>): void
<<boundary>>
PopupConfBoundary
- LabelErrore: [1] Label
- LabelDescrizioneErrore: [1] Label
- setLabelsAttr(title: String, message: String): void
<<boundary>>
PopupAlertBoundary
- LabelAlert: [1] Label
- LabelDescrizioneAlert: [1] Label
- setLabelsAttr(title: String, message: String): void
<<Controller>>
Controller
- controller: [1] Controller
- fileOggettoAnnuncio: [1] File
- fileOggettoScambio: [1] File
- fileOggettoAlternativo: [1] File
- immagineProfiloSelezionata: [1] File
- annunciSelezionato: [1] Annuncio
- listaOggettiDesiderati: ArrayList<String>
- DBConnection: [1] ConnessioneDB
- oggettoDAO: [1] OggettoDAO
- loginId: [1] void
+ setStudente(studente: Studente): void
+ getFileOggettoAlternativo(): File
+ setFileOggettoAlternativo(fileOggettoAlternativo: File): void
+ getFileOggettoOfferto(): File
+ setFileOggettoOfferto(): File
+ getImmagineProfiloSelezionata(): File
+ setImmagineProfiloSelezionata(immagineProfiloSelezionata: File): void
+ setFileOggettoAnnuncio(fileOggettoAnnuncio: File): void
+ setFileOggettoOfferto(fileOggettoOfferto: File): void
+ getListaOggettiDesiderati(): ArrayList<String>
+ setListaOggettiDesiderati(listaOggettiDesiderati: ArrayList<String>): void
+ svuotaListaOggettiDesiderati(): void
+ aggiungiOggettoDesiderato(oggetto: String): void
+ setAnnuncioSelezionato(annuncio: Annuncio): void
+ getAnnuncioSelezionato(): Annuncio
+ setOggettiOfferti(offerteScambio: OfferteScambio, listaOggettiOfferti: ArrayList<Oggetto>): void
+ checkDatiRegistrazione(credenziali: ArrayList<String>): int
+ inserisciStudente(credenziali: ArrayList<String>): void
+ isValidEmail(email: String): int
+ isValidNameSurname(s): String): int
+ checkDatiLogin(username: String, password: String): int
+ getOggettiByEditOggetto: int): Oggetto
+ getIdBySede(sede: Sede): int
+ getInfoAnnunci(studente: Studente): ArrayList<Annuncio>
+ getAnnunciStudente(studente: Studente): ArrayList<Annuncio>
+ getStudente(): Studente
+ creaDescrizioneAnnuncio(dataAnnuncio: ArrayList<String>, fileSelezionato: File, studente: Studente, listaOggettiDesiderati: ArrayList<String>): int
+ copiaFileCaricatoSelezionato(o: File): void
+ copiaImmagineProfiloCaricata(immagineSelezionata: File): void
+ caricaFileImmagine(file: String): void
+ getAnnunciByFiltri(keyword: String, categoria: String, tipologia: String): ArrayList<Annuncio>
+ AcquistaOggetto(a): Annuncio): void
+ invioOfferteVendita(annuncio: Annuncio): int
+ invioOfferteRegalo(annuncio: Annuncio, motivazione: String): int
+ invioOfferteScambio(annuncio: Annuncio, listaOggettiOfferti: ArrayList<Oggetto>): int
+ checkControlRefaVendita(annuncio: Annuncio, stringaPrezzo: String): int
+ checkMotivoControlRefa(annuncio: Annuncio, motivazione: String): int
+ checkMotivoControlRefaVendita(offerta: OfferteVendita, stringaPrezzo: String): int
+ editOfferteRegaloMotivazione(offerta: OfferteRegalo, motivazione: String): int
+ eliminaImmagineOggettoOfferto(offerta: Offerta, oggetto: Oggetto, path: String): int
+ controllaCampiOggettoScambio(descrizione: String, categoriaSelezionata: String, percorsoImmagine: String): int
+ getStudenteByMatricola(matricola: String): Studente
+ rimuoviAnnuncio(annuncio: Annuncio): int
+ rimuoviOfferte(idAnnuncio: int): int
+ rimuoviOggettiOfferti(idOfferta: int): int
+ rimuoviSede(idSede: int): int
+ rimuoviOggettoById(idOggetto: int): int
+ getOfferteByAnnuncio(annuncio: Annuncio): ArrayList<Offerta>
+ accettaOfferta(o: Offerta): int
+ rifiutaOfferta(o: Offerta): int
+ getIdByOfferta(o: Offerta): int
+ getOfferteRicevute(offerta: Offerta): ArrayList<Oggetto>
+ getOfferteMotivate(studente: Studente): ArrayList<Offerta>
+ eliminaOfferta(offerta: Offerta): int
+ SvuotaAnnunciPubblicati(): void
+ SvuotaOfferteInviate(): void
+ setOggettoAnnuncio(idAnnuncio, nomeOggetto: String, listaOggetti: ArrayList<String>): int
+ setAnnunciVisibili(lista: ArrayList<Annuncio>): void
+ setAnnunciPubblicati(lista: ArrayList<Annuncio>): void
+ setOfferteInviate(lista: ArrayList<Offerta>): void
+ rimuoviOggettiDesideratiDaLista(listaOggetti: ArrayList<String>, nomeOggetto: String): void
+ invioOfferteScambio(annuncio: Oggetto, offerte: Offerta, pathImmagine: String, categoria: String, descrizione: String): int
+ rimuoviOggettoOfferto(oggetto: Oggetto, offerta: Offerta): int
+ getNumeroOfferteInviate(studente: Studente, tipologia: String): int
+ getNumeroOfferteAccettate(studente: Studente, tipologia: String): int
+ getInformazioniOfferteInviate(studente: Studente, tipoFunzione: String): double
<<interface>>
InterfaceOfferteDAO
+ SaveOfferteVendita(offerteVendita: OfferteVendita): int
+ SaveOfferteRegalo(offerteRegalo: OfferteRegalo): int
+ SaveOfferteScambio(offerteScambio: OfferteScambio): int
+ UpdateOfferteVendita(prezzo: double, studente: Studente, annuncio: Annuncio): int
+ UpdateOfferteRegalo(motivazione: String, studente: Studente, annuncio: Annuncio): int
+ UpdateStatoOfferta(offerta: Offerta): int
+ rimuoviOfferteByIdAnnuncio(idAnnuncio: int): int
+ getOfferteByAnnuncio(annuncio: Annuncio): ArrayList<Offerta>
+ accettaOfferta(offerta: Offerta): int
+ rifiutaOfferta(offerta: Offerta): int
+ getIdByOfferta(offerta: Offerta): int
+ getOggettiOfferti(offerta: Offerta): ArrayList<Oggetto>
+ eliminaOfferta(offerta: Offerta): int
+ getNumeroOfferteInviate(studente: Studente): ArrayList<Offerta>
+ getNumeroOfferteAccettate(studente: Studente, tipologia: String): int
+ getInformazioniOfferteInviate(studente: Studente, tipoFunzione: String): double
<<interface>>
InterfaceSedeDAO
+ SaveSede(sede: Sede): int
+ getIdBySede(sede: Sede): int
+ rimuoviSede(idSede: int): int
<<interface>>
InterfaceOggettiOffertiDAO
+ SaveOggettoOfferto(idOfferta: int, idOggetto: int): void
+ rimuoviOggettoOfferto(idByIdOggetto: int, idOfferta: int): void
+ rimuoviOggettiOfferti(idOfferta): ArrayList<Integer>
<<interface>>
InterfaceOggettoDAO
+ SaveOggetto(oggetto: Oggetto): int
+ getOggettoById(idOggetto: int): Oggetto
+ rimuoviOggettoById(idOggetto): int
+ updateOggetto(oggetto: Oggetto, pathImmagine: String, categoria: String, descrizione: String): int
+ UpdateImmagineOggetto(oggetto: Oggetto, pathImmagine: String): int
<<interface>>
InterfaceStudentiDAO
+ SaveStudente: Studente, password: String): int
+ LoginStudente(username: String, password: String): Studente
+ CheckStudenteEsistente(matricola: String, email: String, username: String): int
+ getStudenteByMatricola(matricola: String): Studente
+ cambiaFotoProfilo(matricola: String, percorsoImmagine: String): void
<<interface>>
InterfaceAnnunciDAO
+ SaveAnnuncio(annuncio: Annuncio, oggetto: Oggetto, sede: Sede): int
+ rimuoviAnnuncio(annuncio: Annuncio): int
+ getAnnunciStudente(studente: Studente): ArrayList<Annuncio>
+ getAnnunciStudenteById(studente: Studente): ArrayList<Annuncio>
+ getAnnunciByFiltri(keyword: String, categoria: String, tipologia: String, idStudente: int): ArrayList<Annuncio>
+ cambiaStatoAnnuncio(annuncio: Annuncio): void
+ getAnnunciById(studente: Studente, idAnnuncio: int): Annuncio
OfferteDAO
SediDAO
OggettiOffertiDAO
OggettiDAO
StudentiDAO
AnnunciDAO
<<entity>>
Studenti
- matricola: [1] String
- email: [1] String
- nome: [1] String
- cognome: [1] String
- username: [1] String
- immagineProfilo: [1] String
+ Studente(String matricola, String email, String nome, String cognome, String username): Studente
<<entity>>
Annunci
- titoloannuncio: [1] String
- statoAnnuncio: [1] boolean
- fasciaOrariaInizio: [1] String
- fasciaOrariaFine: [1] String
- dataPubblicazione: [1] Timestamp
- giorni: [3] String
- descrizioneAnnuncio: [1] String
- tipologia: [1] String
- prezzo: [1] double
public AnnunciciString titoloAnnuncio, boolean statoAnnuncio, String fasciaOrariaInizio, String fasciaOrariaFine, double prezzo, String tipologia, String descrizioneAnnuncio,Oggetto oggetto, Sede sede, String giorni, Timestamp dataPubblicazione): Annuncio
<<entity>>
Sede
- particellaToponomastica: [1] String
- descrizioneIndirizzo: [1] String
- civico: [1] String
- cap: [1] String
+ Sede(String particellaToponomastica, String descrizioneIndirizzo, String civico, String cap): Sede
<<entity>>
Offerta
- statoOfferta: [1] String
- dataPubblicazione: [1] Timestamp
+ Offerta(Timestamp dataPubblicazione, Studente studente, Annuncio annuncio): Offerta
<<entity>>
Oggetto
- immagineOggetto: [1] String
- categoria: [1] String
- descrizione: [1] String
+ Oggetto(String immagineOggetto, String categoria, String descrizione, Studente studente): Oggetto
<<entity>>
OfferteVendita
- prezzoOfferta: [1] String
+ OfferteVendita(Timestamp dataPubblicazione, Studente studente, Annuncio annuncio, double prezzoOfferta): OfferteVendita
<<entity>>
OfferteScambio
+ OfferteScambio(Timestamp dataPubblicazione, Studente studente, Annuncio annuncio): OfferteScambio
<<entity>>
OfferteRegalo
- motivazione: [1] String
+ OfferteRegalo(Timestamp dataPubblicazione, Studente studente, Annuncio annuncio, String motivazione): OfferteRegalo

Link al repository Github

Gruppo OOBD42:

- Giuseppe Cautiero N86005564
- Luigi Castaldo N86005588

> https://github.com/Lvi00/UninaSwap

Principali funzionalità del controller

Le classi boundary possono essere modificate in qualsiasi momento, infatti, con la separazione secondo l'euristica BCE, la logica di business definita nel controller e il dominio rimangono tali e l'implementazione dell'interfaccia grafica può essere cambiata utilizzando altri framework.

Stessa cosa per le classi DAO, ad esempio, potremo decidere in futuro di effettuare una nuova implementazione delle interfacce mediante file e il controller non si accorgerà di questa modifica.

Questo non solo rende l'intero programma più scalabile, ma separa la responsabilità di reperimento dei dati e in generale tutte le operazioni CRUD dalla logica di business che non deve essere soggetta ad importanti modifiche.

Controllo delle credenziali di registrazione

Questo metodo ha lo scopo di validare i dati di registrazione di un utente, esso prende in input un `ArrayList<String>` che contiene le credenziali dell'utente inserite nella sezione di registrazione e restituisce un codice numerico in base al problema che è stato riscontrato nelle credenziali, se viene restituito 0 allora la registrazione è andata a buon fine.

Ogni codice di errore ha un significato e nella classe RegistrazioneBoundary e ognuno di questi corrisponde ad un alert che verrà mostrato facendo una chiamata al metodo `showPopupError()` passando come parametro due stringhe, cioè quella che indica il titolo dell'errore e quella che indica la descrizione dell'errore

> 💡 Una logica simile l'abbiamo implementata anche per gli alert, che vengono mostrati solo quando un'operazione va a buon fine, il metodo corrispondente infatti è `showPopupAlert()`

In generale si prendono i valori dall'arraylist e si effettuano questi controlli:

- Verifica che tutti i campi non siano vuoti
- Verifica che la lunghezza del nome sia compresa tra 2 e 40 e che sia una stringa che rispetti la regex `^[A-Za-zÀ-ÿ'\\s]+$` in particolare questo viene controllato da un apposito metodo che è `isValidNameSurname()` e ritorna un intero, 0 se la stringa matcha, 1 altrimenti

- Effettua controlli analoghi per il cognome
- Verifica che la matricola sia una stringa di 9 caratteri e che non contenga spazi o caratteri di tabulazione
- Verifica che la lunghezza della password sia compresa tra 8 e 20
- Verifica che l'email abbia un formato valido e che rispetti la regex `^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}$` e di questo se ne occuperà il metodo `isValidEmail()`
- Verifica che lo studente che si è registrato con quella matricola, quell'email e quell'username non esista già sul db, e di questo se ne occuperà un apposito metodo presente nella classe StudenteDAO
    - Questo metodo di StudenteDAO effettuerà una query e ritornerà 1 se quello studente esiste già, altrimenti 0

```
public int checkDatiRegistrazione(ArrayList<String> credenziali) {
        String nome = credenziali.get(0);
        String cognome = credenziali.get(1);
        String matricola = credenziali.get(2);
        String email = credenziali.get(3);
        String username = credenziali.get(4);
        String password = credenziali.get(5);

        if(nome == "" || cognome == "" ||  matricola == ""
        || email == "" || username ==  "" || password ==  "") {
           return 1;
        }

    if ((nome.length() > 40 || nome.length() < 2)
    || isValidNameSurname(nome) == 1) {
       return 2;
    }

    if ((cognome.length() > 40 || cognome.length() < 2)
    || isValidNameSurname(cognome) == 1) {
       return 3;
    }

    if (matricola.length() != 9 || matricola.contains(" ")
    || matricola.contains("\t")) {
       return 4;
    }

    if(isValidEmail(email) == 1) {
        return 5;
    }

    if (username.length() > 10 || username.contains(" ")
```

```
        || username.contains("\t")) {
            return 6;
        }

        if (password.length() < 8 || password.length() > 20) {
            return 7;
        }

        if(studenteDAO.CheckStudenteEsistente(matricola, email, username) ==
1) {
            return 8;
        }

        return 0;
    }

private int isValidNameSurname(String s) {
    String sRegex = "^[A-Za-zÀ-ÿ'\\s]+$";

  Pattern pattern = Pattern.compile(sRegex);
  Matcher matcher = pattern.matcher(s);

  if (matcher.matches()) return 0;

  return 1;
}

private int isValidEmail(String email) {
        if (email == null || email.contains(" ") || email.contains("\t")) {
            return 1;
        }

        String regex = "^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}$";

    Pattern pattern = Pattern.compile(regex);
    Matcher matcher = pattern.matcher(email);

    if (matcher.matches()) return 0;

    return 1;
}
```

Controllo login studente

Questo metodo prende l’username e la password inserita dallo studente nella schermata di login e controlla l’esistenza e la correttezza della password, se il controllo va a buon fine allora viene ritornato un oggetto Studente associato allo

studente loggato, altrimento uno studente NULL, in tal caso ovviamente si ritorna un codice di errore per comunicare allo studente che le credenziali non sono corrette.

> 💡 In caso di accesso si viene rimandati alla schermata Prodotti e si valorizza l'attributo studente presente nel Controller, cioè inizia la sessione dello studente loggato

```
public int CheckLoginStudente(String username, String password){
    Studente studente = studenteDAO.LoginStudente(username, password);
    if(studente != null) {
        this.studente = studente;
        return 0;
    }

    return 1;
}
```

> 💡 La password inserita nella schermata di login non sarà mai salvata in un oggetto, essa servirà solo per confrontarla con quella salvata nel database dell'utente con quel determinato username

> 💡 Un possibile miglioramento della sicurezza potrebbe essere quello di far passare la password per una funzione di hash e salvare l'hash della password all'interno del db invece di salvarla in chiaro

Logout studente

Questo metodo viene chiamato quando si va nella schermata Profilo e si clicca su "Esci dal profilo utente", quello che fa è semplicemente portare a NULL tutti gli attributi del Controller e svuotare l'array listaOggettiDesiderati che mi permetteva di salvare gli oggetti desiderati durante la creazione di un annuncio di scambio.

> 💡 In questo modo il Garbage Collector andrà ad eliminare gli oggetti che non ci servono più

```
public void logout() {
    this.studente = null;
    this.fileOggettoAnnuncio = null;
    this.fileOggettoOfferto = null;
  this.fileOggettoAlternativo = null;
  this.immagineProfiloSelezionata = null;
```

```
    this.annuncioSelezionato = null;
  this.listaOggettiDesiderati.clear();
}
```

Controllo dei dati di un annuncio

Questo metodo viene innescato quando si clicca sul tasto “Crea annuncio” e controlla i dati inseriti nei campi della schermata Crea Annuncio. Il metodo accetta come parametro una `ArrayList<String>` che contiene i dati da controllare, un file che si riferisce all’immagine dell’annuncio, lo studente che effettua l’annuncio e un’altra `ArrayList<String>` che contiene eventuali oggetti desiderati (in caso di annuncio di tipo scambio)

I controlli che vengono effettuati sono:

- Verifica che il titolo non sia una stringa vuota e che la sua lunghezza sia minore di 50
- Verifica che la categoria non sia vuota (aggiunta per completezza ma la categoria è già un enum quindi non può essere vuota)
- Verifica che l’orario finale di disponibilità sia maggiore di quello iniziale
  - Per evitare situazioni come 12:00-09:00
- Verifica che sia stato selezionato almeno un giorno di disponibilità
  - Si provvede anche ad eliminare eventuali caratteri speciali come “-”
- Verifica che la descrizione non sia vuota e che la sua lunghezza sia minore di 100
- Verifica della validità dei campi atomici della sede
  - La particella toponomastica è un enum
  - Si controlla che la descrizione non sia vuota e che la sua lunghezza sia minore di 100
  - Si controlla che il civico non sia vuoto e che la sua lunghezza sia minore di 4
  - Si controlla mediante la regex `^[0-9]{5}$` che il cap sia una stringa fissa di 5 caratteri e che abbia come caratteri solo cifre
- Verifica della validità della tipologia (aggiunta per completezza ma la tipologia è un enum)
- Se la tipologia è vendita si controlla la validità del prezzo che deve essere maggiore di 0
- Verifica che il file non sia NULL, che abbia un nome valido e che non coincida con l’immagine di default
- Se la tipologia è scambio verifica la validità degli oggetti desiderati
- Verifica che l’annuncio non esista già nel database
  - Di questo se ne occupa il metodo `SaveAnnuncio()` della classe AnnuncioDAO

Se passano tutti i controlli allora si chiamano le varie DAO dei vari componenti dell'annuncio (SedeDAO e OggettoDAO) e l'annuncio stesso con AnnuncioDAO, infine, si restituisce al chiamante un codice che corrisponde ad un determinato popup.

```
public int checkDatiAnnuncio(ArrayList<String> datiAnnuncio, File fileSelezion
ato,
Studente studente, ArrayList<String> listaOggettiDesiderati) {
        String titolo = datiAnnuncio.get(0).trim();
        if(titolo == "" || titolo.length() > 50) return 1;

        String categoria = datiAnnuncio.get(1);
        if(categoria.equals("")) return 2;

        String inizioOrarioDisponibilità = datiAnnuncio.get(2).trim();
        String fineOrarioDisponibilità = datiAnnuncio.get(3).trim();

        if (!LocalTime.parse(inizioOrarioDisponibilità).isBefore(
        LocalTime.parse(fineOrarioDisponibilità))) {
            return 3;
        }

        String giorniDisponibilità = datiAnnuncio.get(4).replaceAll("-$", "").trim();
        if(giorniDisponibilità == "") return 4;

        String descrizione = datiAnnuncio.get(5).trim();
        if(descrizione.equals("") || descrizione.length() > 100) return 5;

        String particellatoponomastica = datiAnnuncio.get(6).trim();
        String descrizioneIndirizzo = datiAnnuncio.get(7).trim();
        String civico = datiAnnuncio.get(8).trim();
        String cap = datiAnnuncio.get(9).trim();

        String capRegex = "^[0-9]{5}$";

        if(particellatoponomastica == "" || descrizioneIndirizzo == ""
        || descrizioneIndirizzo.length() > 100 || civico == "" ||
        civico.length() > 4 || cap == "" || cap.length() != 5 ||
        !cap.matches(capRegex)) return 6;

        String tipologia = datiAnnuncio.get(10);

        if(tipologia == "") return 7;

        double prezzo = 0.0;

        if(tipologia.equals("Vendita")){
            String stringaPrezzo = datiAnnuncio.get(11);
```

```
            String prezzoRegex = "^\\d{1,3}(\\.\\d{1,2})?$";

        if (!stringaPrezzo.matches(prezzoRegex)) return 8;

            prezzo = Double.parseDouble(stringaPrezzo);

            if(prezzo <= 0 || prezzo >= 1000) return 8;
        }

        if(fileSelezionato == null || fileSelezionato.getName().isEmpty()
        || fileSelezionato.getName().startsWith("no_image")) return 9;

        String appoggioDescrizione = descrizione;

        if(tipologia.equals("Scambio")) {
            if(listaOggettiDesiderati.size() < 1 ||
            listaOggettiDesiderati.size() > 5) return 10;

            descrizione = "Oggetti desiderati: ";

            for(int i = 0; i < listaOggettiDesiderati.size(); i++) {
                descrizione = descrizione + listaOggettiDesiderati.get(i);
                if(i != listaOggettiDesiderati.size() - 1) descrizione = descrizione + ",
";
            }

            descrizione += "\n" + appoggioDescrizione;
            descrizione = descrizione.trim();
        }
        Sede sede = new Sede(particellatoponomastica, descrizioneIndirizzo, civ
ico, cap);
        int idSede = sedeDAO.SaveSade(sede);
        sede.setIdSede(idSede);

        String percorso = fileSelezionato.getAbsolutePath();

        Oggetto oggetto = new Oggetto(percorso, categoria, appoggioDescrizion
e, studente);
        int idOggetto = oggettoDAO.SaveOggetto(oggetto);
        oggetto.setIdOggetto(idOggetto);

        Timestamp dataCorrente = new Timestamp(System.currentTimeMillis());

        Annuncio annuncio = new Annuncio(titolo, true,
        inizioOrarioDisponibilità, fineOrarioDisponibilità, prezzo,
        tipologia, descrizione, oggetto, sede, giorniDisponibilità, dataCorrente);
```

```
        this.studente.getAnnunciPubblicati().clear();

        if(annuncioDAO.SaveAnnuncio(annuncio, oggetto, sede) == 1) return 11;

        return 0;
    }
```

Invia offerta di vendita

Il primo metodo permette di inviare un'offerta di vendita ad un annuncio di vendita, per prima cosa verifica la validità della controfferta e se è valido costruisce la nuova offerta, poi la passa come parametro al metodo `SaveOffertaVendita()` della classe OffertaDAO. Il secondo metodo è simile ma non tiene conto della validità della controfferta perchè viene invocato quando si accetta il prezzo dell'annuncio.

```
public int checkControffertaVendita(Annuncio annuncio, String stringaPrezzo)
{
    String prezzoRegex = "^[0-9]+(\\.[0-9]{1,2})?$";

    if (!stringaPrezzo.matches(prezzoRegex)) {
        return 1;
    }

    double prezzo = Double.parseDouble(stringaPrezzo);

    if (prezzo <= 0 || prezzo >= annuncio.getPrezzo()) {
        return 1;
    }

    Timestamp dataCorrente = new Timestamp(System.currentTimeMillis());

    OffertaVendita offerta = new OffertaVendita(dataCorrente,
    this.studente, annuncio, prezzo);

    int risultato = offertaDAO.SaveOffertaVendita(offerta);

    if(risultato == 0) SvuotaOfferteInviate();

    return risultato;
}

public int inviaOffertaVendita(Annuncio annuncio) {
    Timestamp dataCorrente = new Timestamp(System.currentTimeMillis());
    OffertaVendita offertaVendita = new OffertaVendita(dataCorrente,
    this.studente, annuncio, annuncio.getPrezzo());

    int result = offertaDAO.SaveOffertaVendita(offertaVendita);
```

```
  if(result == 0) controller.SvuotaOfferteInviate();

    return result;
}
```

Invia offerta di scambio

Questo metodo viene invocato in caso di invio di un'offerta che contiene oggetti offerti (oggetti alternativi a quelli desiderati dallo studente che ha pubblicato l'annuncio)

Per prima cosa controlla che l'offerta non esiste già, poi scorre la lista di oggetti offerti per controllare la validità dei loro dati (descrizione, categoria e l'immagine) e che non ci siano duplicati. Se tutto va a buon fine, tutti gli oggetti offerti vengono salvati, e di questo se ne occuperà il metodo `SaveOggettoOfferto()` della classe OggettiOffertiDAO.

```
public int inviaOffertaScambio(Annuncio annuncio, ArrayList<Oggetto> listaO
ggettiOfferti){
    Timestamp dataCorrente = new Timestamp(System.currentTimeMillis());

        OffertaScambio offertaScambio = new OffertaScambio(dataCorrente,
        this.studente, annuncio);

        int idOffertaInserita = offertaDAO.SaveOffertaScambio(offertaScambio);

    if(idOffertaInserita == -1) {
        return -1;
    }

    if(!listaOggettiOfferti.isEmpty()) {

        offertaScambio.setOggettiOfferti(listaOggettiOfferti);

        Oggetto oggettoPrimario, oggettoSecondario;

        for(int i = 0; i < listaOggettiOfferti.size(); i++) {
            oggettoPrimario = listaOggettiOfferti.get(i);
            for(int j = i + 1; j < listaOggettiOfferti.size(); j++) {
                oggettoSecondario = listaOggettiOfferti.get(j);
                if(oggettoPrimario.getDescrizione()
                .equals(oggettoSecondario.getDescrizione())
                && oggettoPrimario.getCategoria()
                .equals(oggettoSecondario.getCategoria())
                && oggettoPrimario.getStudente().getMatricola()
                .equals(oggettoSecondario.getStudente().getMatricola())
                && oggettoPrimario.getImmagineOggetto()
```

```
            .equals(oggettoSecondario.getImmagineOggetto())){
               System.out.println("Oggetti duplicati");
               return -2;
            }
         }
         }

         int idOggettoInserito = 0;

       for(Oggetto o: listaOggettiOfferti){
         idOggettoInserito = oggettoDAO.SaveOggetto(o);
         oggettiOffertiDAO.SaveOggettoOfferto(idOffertaInserita, idOggettoInserit
o);
       }

       setOggettiOfferti(offertaScambio, listaOggettiOfferti);
         SvuotaOfferteInviate();

            return 1;
    }

     SvuotaOfferteInviate();

    return 0;
}
```

Invia offerta di regalo

Questo metodo viene invocato in caso di invio di un'offerta di regalo. Per prima cosa mette una motivazione "Assente" se la stringa del campo motivazione è NULL o è vuota, poi crea l'offerta di tipo OffertaRegalo e la salva invocando il metodo `SaveOffertaRegalo()` della classe OffertaDAO.

```
    public int inviaOffertaRegalo(Annuncio annuncio, String motivazione){
       if(motivazione == null || motivazione.isEmpty()) motivazione = "Assente";
       if(motivazione.length()>255) return 1;
       Timestamp dataCorrente = new Timestamp(System.currentTimeMillis());
       OffertaRegalo offertaRegalo = new OffertaRegalo(dataCorrente,
       this.studente, annuncio, motivazione);

       int result = offertaDAO.SaveOffertaRegalo(offertaRegalo);

       if(result == 0) SvuotaOfferteInviate();

       return result;
    }
```

Accettare un’offerta

Questo metodo accetta come parametro un’offerta e cambia il suo stato in “Accettata” invocando il metodo `accettaOfferta()` della classe OffertaDAO. Questo’ultimo cambia lo stato dell’offerta da accettare da “Attesa” a “Accettata” e rifiuta tutte le altre offerte di quell’annuncio aggiornando il loro stato da “Attesa” a “Rifiutata”.

```
public int accettaOfferta(Offerta o) {
    int risultato = offertaDAO.accettaOfferta(o);

    if(risultato == 0) controller.SvuotaAnnunciPubblicati();

    return risultato;
}
```

Rifiutare un’offerta

Questo metodo accetta come parametro un’offerta e cambia il suo stato in “Rifiutata” invocando il metodo `rifiutaOfferta()` della classe OffertaDAO. Questo’ultimo si limita a cambiare lo stato dell’offerta da rifiutare da “Attesa” a “Rifiutata”.

```
public int rifiutaOfferta(Offerta o) {
    int risultato = offertaDAO.rifiutaOfferta(o);

    if(risultato == 0) this.studente.getAnnunciPubblicati().clear();

    return risultato;
}
```

Modificare offerte

Questi metodi vengono invocati in caso di modifica di un offerta della tipologia associata (ad es. `checkModificaControffertaVendita()` viene chiamato quando si modifica un’offerta di tipo vendita)

Il primo metodo controlla la validità del prezzo e se è un prezzo diverso da quello precedente, e mantiene le condizioni imposte (cioè deve essere positivo e minore o uguale del prezzo dell’annuncio) allora si effettua la modifica chiamando il metodo `UpdateOffertaVendita()` della classe OffertaDAO.

Il secondo metodo scorre gli oggetti offerti e controlla che ci sia stato un cambiamento di uno dei 3 campi (descrizione, categoria o immagine), se è vero allora applica le modifiche invocando il metodo `UpdateOffertaScambio()` della classe OffertaDAO.

Il terzo metodo controlla la validità del campo motivazione, se essa è valida allora invoca il metodo `UpdateOffertaRegalo()` della classe OffertaDAO.

In tutti e tre casi, se viene chiamato il metodo di aggiornamento dell'offerta si cambia lo stato di quest'ultima in "Attesa".

```
public int checkModificaControffertaVendita(OffertaVendita offerta,
String stringaPrezzo) {
    String prezzoRegex = "^[0-9]+(\\.[0-9]{1,2})?$";
    if (!stringaPrezzo.matches(prezzoRegex)) {
        return 1;
    }
    System.out.println(stringaPrezzo);
    double prezzo = Double.parseDouble(stringaPrezzo);

    if (prezzo <= 0 || prezzo == offerta.getPrezzoOfferta() ||
    prezzo > offerta.getAnnuncio().getPrezzo()) {
        return 1;
    }

    int risultato = offertaDAO.UpdateOffertaVendita(prezzo,
    offerta.getStudente(),  offerta.getAnnuncio());

    if(risultato == 0) {
        controller.setPrezzoOfferta(offerta, Double.parseDouble(stringaPrezzo));
        controller.SvuotaOfferteInviate();
    }

    return risultato;
}

public int editOffertaScambio(Oggetto oggetto, Offerta offerta,
String pathImmagine , String categoria, String descrizione) {
    if(offerta instanceof OffertaScambio) {
        OffertaScambio offertaScambio = (OffertaScambio) offerta;

    if(getImmagineOggetto(oggetto).equals(pathImmagine) &&
    getCategoriaOggetto(oggetto).equals(categoria) &&
    getDescrizioneOggetto(oggetto).equals(descrizione)) {
        return 1;
    }

    ArrayList<Oggetto> listaOggettiOfferti = offertaScambio.getOggettiOfferti();

    for(Oggetto o : listaOggettiOfferti) {
        if(getImmagineOggetto(o).equals(pathImmagine) &&
        getCategoriaOggetto(o).equals(categoria) &&
        getDescrizioneOggetto(o).equals(descrizione)) {
            return 2;
        }
    }
```

```
    if(oggettoDAO.UpdateOggetto(oggetto,pathImmagine, categoria, descrizion
e) == 1){
        return 3;
    }

    offertaDAO.UpdateStatoOfferta(offertaScambio);

    offertaScambio.setOggettiOfferti(getOggettiOffertiByOfferta(offertaScambi
o));

    SvuotaOfferteInviate();

    return 0;
    }

    return -1;
}

public int editOffertaRegaloMotivazione(OffertaRegalo offerta, String motivazi
one) {
    if (motivazione == null) motivazione = "";
    motivazione = motivazione.trim().replaceAll("\\s{2,}", " ");
    String motivazioneRegex = "^[\\p{L}0-9,.!?;:'\"()\\s-]*$";

    if (!motivazione.matches(motivazioneRegex) || motivazione.length()>255) {
        return 1;
    }

    if (motivazione.isEmpty()) {
        motivazione = "Assente";
    }

    String vecchiaMotivazione = offerta.getMotivazione();
    if (vecchiaMotivazione == null || vecchiaMotivazione.trim().isEmpty()) {
        vecchiaMotivazione = "Assente";
    }

    if (vecchiaMotivazione.equals(motivazione)) {
        return 1;
    }

    offerta.setMotivazione(motivazione);

    int risultato = offertaDAO.UpdateOffertaRegalo(motivazione,
    offerta.getStudente(), offerta.getAnnuncio());

    if(risultato == 0) {
```

```
        offerta.setMotivazione(motivazione);
            controller.SvuotaOfferteInviate();
    }

    return risultato;
}
```

Mockup login e registrazione

L'idea per il mockup della login e della registrazione è stata quella di fare uno schema semplice, quest'ultimo doveva contenere solo i campi necessari a svolgere le operazioni di autenticazione/registrazione

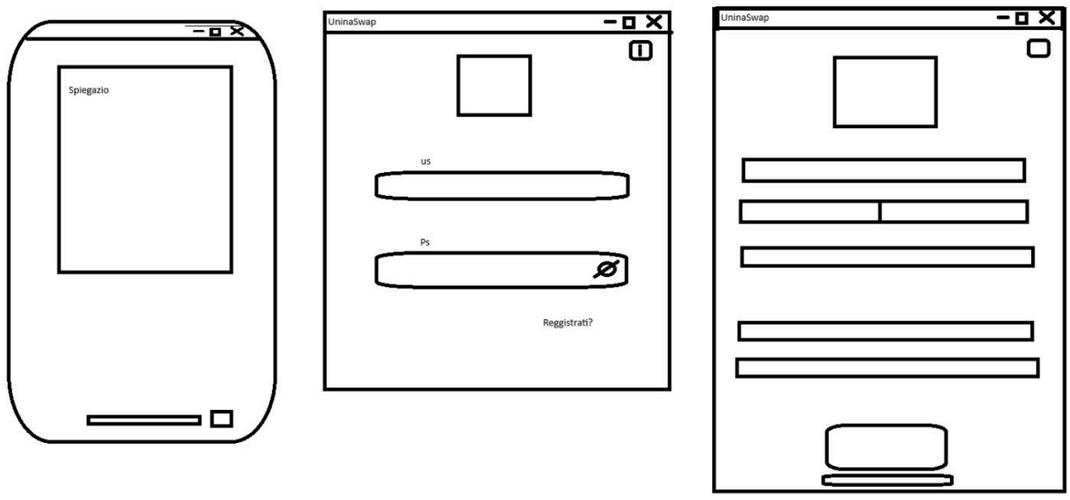
Nella login sono stati inseriti due campi:

- Il primo per l'username
- Il secondo per la password

Inoltre, abbiamo inserito un'apposita icona che permette di vedere la password in chiaro nei campi password.

Nella registrazione sono stati inseriti solo dei campi che rappresentano le informazioni personali di uno studente, nella mockup di registrazione, infatti, abbiamo inserito solo un numero generico di campi che ci ha permesso di farci un'idea sulla sua struttura generale.

Inoltre, nella login e nella registrazione abbiamo deciso di aggiungere in alto a destra un'apposita icona che descrive il formato corretto dei dati da inserire nei campi di input. La sua rappresentazione è nella prima immagine a partire da sinistra. In questa zona abbiamo deciso di aggiungere solo una parte testuale riservata alla descrizione.



Mockup dashboard

Il mockup della dashboard rappresenta un'idea generale della struttura del design della dashboard che l'utente vedrà una volta autenticato. Come si può vedere dall'immagine è possibile notare una suddivisione in parti dove sul lato sinistro,

cioè sulla sidebar, sono presenti le varie sezioni del programma, mentre nella parte orizzontale, cioè sulla navbar, è possibile notare una zona cerchiata in fondo a destra che rappresenta il nome utente e l’icona dello studente loggato. La zona centrale serve solo da placeholder perchè ci andrà solo il contenuto delle varie pagine.