Corso di Laurea in Informatica - Università degli Studi di Napoli Federico II
A.A. 2025/2026

UninaFoodLab

Calone Francesco N86005555

D'Angelo Mario N86005477

Codice gruppo: **OOBD39**

Insegnamento di Programmazione Object-Oriented

Indice

1 Introduzione

Il progetto UninaFoodLab nasce con l'obiettivo di offrire una soluzione software moderna e intuitiva per la gestione di corsi culinari. La piattaforma è stata sviluppata seguendo le best practice adottando tecnologie consolidate come JavaFX per la realizzazione dell'interfaccia grafica e Maven per la gestione delle dipendenze e dei processi di build. Il progetto è stato concepito per essere facilmente estendibile e manutenibile, grazie a una chiara separazione dei livelli logici. La documentazione che segue illustra le scelte progettuali, le tecnologie utilizzate e le principali funzionalità implementate, con l'intento di fornire una panoramica completa e professionale del sistema sviluppato.

1.1 Obiettivi del Progetto

Il progetto UninaFoodLab si propone di:

- Fornire una piattaforma intuitiva per la gestione di corsi culinari
- Semplificare la registrazione e la gestione degli utenti e per gli chef
- Facilitare la creazione e la gestione dei corsi, inclusa la pianificazione delle lezioni e la gestione delle ricette
- Semplificare la gestione delle prenotazioni e dei pagamenti per i corsi

2 Tool utilizzati

2.1 Maven

Maven è uno strumento di gestione e automazione dei progetti software, ampiamente utilizzato nell'ecosistema Java. Permette di gestire le dipendenze, automatizzare la compilazione, l'esecuzione dei test e la creazione dei pacchetti eseguibili.

Nel progetto UninaFoodLab, Maven è stato utilizzato per semplificare la configurazione e la gestione delle librerie necessarie, come JavaFX per la realizzazione dell'interfaccia grafica e il driver JDBC per la connessione al database PostgreSQL. Tutte le dipendenze sono dichiarate nel file 'pom.xml', che consente di mantenere il progetto facilmente aggiornabile e portabile.

L'integrazione con JavaFX è stata gestita tramite le apposite dipendenze e plugin, permettendo di compilare ed eseguire l'applicazione con semplici comandi Maven. Questo approccio ha garantito una maggiore efficienza nello sviluppo e una migliore organizzazione del codice.

2.2 JavaFX

JavaFX è una libreria per la creazione di interfacce utente grafiche (GUI) in Java. È stata progettata per fornire un ambiente di sviluppo moderno e ricco di funzionalità, consentendo la creazione di applicazioni desktop e web con un aspetto e un comportamento coerenti.

Nel progetto UninaFoodLab, JavaFX è stato utilizzato per realizzare l'interfaccia grafica dell'applicazione. Grazie a JavaFX, è stato possibile implementare facilmente elementi UI complessi, come tabelle, grafici e controlli personalizzati, migliorando l'usabilità e l'estetica dell'applicazione.

L'integrazione di JavaFX con Maven ha semplificato ulteriormente il processo di sviluppo, consentendo di gestire le dipendenze e le configurazioni necessarie per l'utilizzo della libreria in modo efficiente e organizzato.

2.3 PostgreSQL

PostgreSQL è un sistema di gestione di database relazionali.

Nel progetto UninaFoodLab, PostgreSQL è stato scelto come database principale per la sua capacità di gestire grandi volumi di dati. L'integrazione con Java è stata realizzata tramite JDBC (Java Database Connectivity), che ha permesso di stabilire una connessione tra l'applicazione Java e il database PostgreSQL in modo semplice e diretto.

3 Architettura del Progetto

L'architettura del progetto UninaFoodLab è stata progettata per garantire una chiara separazione dei compiti e una facile manutenibilità. La struttura delle directory riflette i principali livelli logici dell'applicazione, secondo il pattern MVC e le best practice di progettazione.

Le principali suddivisioni sono:

- **Boundary**: contiene le classi responsabili dell'interfaccia grafica e dell'interazione con l'utente, realizzate tramite JavaFX e FXML.
- **Controller**: gestisce la logica applicativa e il flusso degli eventi tra la GUI e i dati.
- **Entity**: suddivisa ulteriormente in *DAO* (Data Access Object) e *DTO* (Data Transfer Object). I DAO si occupano della persistenza e dell'accesso ai dati, mentre i DTO rappresentano le strutture dati scambiate tra i vari livelli.
- **JDBC**: contiene le classi e le utility per la connessione e la gestione del database PostgreSQL.
- **Utils**: raccoglie le classi di supporto e gli strumenti riutilizzabili all'interno del progetto.

Questa organizzazione favorisce la modularità e la scalabilità del sistema, permettendo di isolare le responsabilità e facilitare l'estensione futura. Ogni componente interagisce con gli altri tramite interfacce ben definite, riducendo le dipendenze e migliorando la qualità del codice.

La scelta di suddividere le entity in DAO e DTO consente di gestire in modo efficiente sia la persistenza che il trasferimento dei dati, mentre la presenza di una directory dedicata alle utility semplifica la gestione delle funzionalità trasversali.

3.1 Struttura delle Directory

La struttura delle directory del progetto è organizzata come segue:

- **src/main/java**: contiene il codice sorgente dell'applicazione.
- **src/main/resources**: contiene le risorse dell'applicazione, come file FXML e immagini.
- **src/test/java**: contiene i test automatizzati.

4 Gestione della GUI

La gestione della Graphical User Interface (GUI) nel progetto UninaFoodLab è stata affidata a JavaFX, una tecnologia che permette di realizzare applicazioni desktop moderne, interattive e dal design professionale. L'utilizzo di JavaFX, insieme ai file FXML e CSS, ha consentito di separare in modo netto la logica di presentazione dalla logica applicativa, favorendo la manutenibilità e l'estendibilità del software.

Ogni sezione dell'applicazione è rappresentata da una scena dedicata, come la schermata di login, la homepage utente, la homepage chef, la gestione dei corsi e la visualizzazione delle ricette. Queste scene sono definite tramite file FXML, che descrivono la struttura e il layout degli elementi grafici, e sono arricchite da fogli di stile CSS che garantiscono coerenza visiva e personalizzazione dell'interfaccia.

La scelta di organizzare la GUI in scene distinte permette di gestire in modo ordinato le diverse funzionalità offerte agli utenti, facilitando sia l'esperienza d'uso che lo sviluppo incrementale del progetto. Inoltre, la presenza di controller dedicati per ciascuna scena assicura una gestione efficace degli eventi e delle interazioni, mantenendo il codice pulito e facilmente testabile.

4.1 Struttura della GUI

La struttura della GUI di UninaFoodLab si basa su una suddivisione modulare in scene, ciascuna dedicata a una funzionalità specifica dell'applicazione. Ogni scena è definita da un file FXML, che descrive in modo dichiarativo la disposizione degli elementi grafici (bottoni, tabelle, form, ecc.) e ne facilita la modifica senza dover intervenire direttamente sul codice Java.

I principali file FXML includono, ad esempio:

- `accountmanagement.fxml`
- `accountmanagementchef.fxml`
- `calendardialog.fxml`
- `cardcorso.fxml`
- `createcourse.fxml`
- `editcourse.fxml`
- `enrolledcourses.fxml`
- `homepagechef.fxml`
- `homepageutente.fxml`
- `loginpage.fxml`

- `logoutdialog.fxml`
- `monthlyreport.fxml`
- `paymentpage.fxml`
- `registerpage.fxml`
- `successdialog.fxml`
- `usercards.fxml`

Ogni file FXML è associato a una classe Boundary e a un Controller dedicato, che si occupano rispettivamente della gestione dell'interfaccia e della logica degli eventi. Questa organizzazione consente di mantenere il codice pulito e facilmente estendibile, favorendo la collaborazione tra sviluppatori e la suddivisione dei compiti.

La personalizzazione grafica è affidata ai fogli di stile CSS, che permettono di differenziare l'aspetto delle varie scene e di adattare l'interfaccia alle esigenze dei diversi ruoli (utente e chef). Ad esempio, la dashboard dello chef presenta strumenti e colori distintivi rispetto a quella dell'utente, rendendo immediata la comprensione delle funzionalità disponibili.

4.2 Personalizzazione grafica con CSS

La personalizzazione dell'interfaccia grafica è stata realizzata tramite fogli di stile CSS dedicati, che permettono di controllare l'aspetto di ogni elemento della GUI in modo flessibile e centralizzato. Ogni scena principale dispone di un proprio file CSS, come `loginpage.css`, `navbar.css`, `chef.css`, `courses.css`, che definiscono colori, font, spaziature e comportamenti visivi.

Questa scelta consente di mantenere una coerenza stilistica tra le diverse schermate e di differenziare l'esperienza utente in base al ruolo. Ad esempio, le dashboard di chef e utente presentano palette di colori e layout specifici, facilitando la navigazione e la comprensione delle funzionalità disponibili. Inoltre, l'utilizzo dei CSS semplifica eventuali modifiche grafiche future, rendendo il progetto facilmente adattabile a nuove esigenze o preferenze estetiche.

4.3 Accesso e Registrazione

Le funzionalità di accesso e registrazione sono fondamentali per la sicurezza e la personalizzazione dell'esperienza utente. Queste operazioni sono gestite tramite scene dedicate e controller specializzati, che garantiscono la validazione dei dati, la gestione degli errori e la corretta distinzione tra i ruoli di chef e utente visitatore.

4.3.1 Login

Il processo di autenticazione avviene tramite la schermata definita in `loginpage.fxml`, gestita da `LoginBoundary.java` e `LoginController.java`. L'utente inserisce le proprie credenziali e, al clic sul pulsante di login, il controller verifica i dati e mostra eventuali messaggi di errore o accede alla dashboard appropriata. Un esempio di metodo di gestione dell'evento potrebbe essere:

Il flusso di login è gestito in modo strutturato e sicuro. Di seguito viene mostrato un estratto reale del codice utilizzato:

```
// LoginBoundary.java
@FXML
private void LoginClick(ActionEvent event) {
    String email = emailField.getText();
    String password = passwordField.getText();
    String errorMsg = controller.handleLogin(event, email, password);
    if (errorMsg != null) {
        errorLabel.setText(errorMsg);
        errorLabel.setVisible(true);
```

```
    } else {
        errorLabel.setText("");
        errorLabel.setVisible(false);
    }
}

// LoginController.java
public String handleLogin(ActionEvent event, String email, String password) {
    try {
        Stage stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
        UtenteVisitatoreDao utenteDao = new UtenteVisitatoreDao();
        String tipo = utenteDao.TipoDiAccount(email, password);
        if (tipo == null) {
            return "Tipo di account non riconosciuto.";
        }
        if ("c".equals(tipo)) {
            ChefDao chefDao = new ChefDao();
            Chef chef = new Chef();
            chef.setEmail(email);
            chef.setPassword(password);
            chefDao.recuperaDatiUtente(chef);
            Chef.loggedUser = chef;
            SceneSwitcher.switchToMainApp(stage, "/fxml/homepagechef.fxml", "UninaFoodLab - Homepage");
            return null;
        } else if ("v".equals(tipo)) {
            UtenteVisitatore utente = new UtenteVisitatore();
            utente.setEmail(email);
            utente.setPassword(password);
            utenteDao.recuperaDatiUtente(utente);
            UtenteVisitatore.loggedUser = utente;
            SceneSwitcher.switchToMainApp(stage, "/fxml/homepageutente.fxml", "UninaFoodLab - Homepage")
            return null;
        } else {
            return "Email o password errati.";
        }
    } catch (IOException e) {
        return "Errore interno. Riprova.";
    }
}
```

In questo modo, il sistema distingue tra chef e utente visitatore, mostrando la dashboard corretta in base al ruolo. Gli eventuali errori vengono gestiti e comunicati all'utente in modo chiaro e immediato.

4.3.2 Registrazione

La registrazione di un nuovo account avviene tramite la schermata `registerpage.fxml`, gestita da `RegisterBoundary.java` e `RegisterController.java`. L'utente può scegliere se registrarsi come Chef o come Utente Visitatore, compilando i campi richiesti e selezionando il tipo di account.

Il controller si occupa di validare i dati inseriti, mostrando messaggi di errore dettagliati in caso di campi mancanti, password non coincidenti, email non valida o dati incoerenti. Se la registrazione va a buon fine, l'utente viene inserito nel database e visualizza una dialog di conferma.

Ecco un estratto del codice reale che gestisce la validazione e la registrazione:

```
// RegisterBoundary.java
@FXML
```

```
private void onRegistratiClick(ActionEvent event) {
    // Estrai i dati dalla GUI
    String nome = textFieldNome.getText().trim();
    String cognome = textFieldCognome.getText().trim();
    String email = textFieldEmail.getText().trim();
    String password = textFieldPassword.getText();
    String confermaPassword = textFieldConfermaPassword.getText();
    String genere = comboBoxGenere.getValue();
    String descrizione = textFieldDescrizione.getText().trim();
    String anniEsperienza = textFieldAnniEsperienza.getText().trim();
    boolean utenteSelezionato = radioUtente.isSelected();
    boolean chefSelezionato = radioChef.isSelected();
    var dataNascita = datePickerDataNascita.getValue();

    // Passa i dati al controller
    String errore = controller.validaRegistrazione(
        nome, cognome, email, password, confermaPassword, genere,
        descrizione, anniEsperienza, utenteSelezionato, chefSelezionato, dataNascita
    );

    if (errore != null) {
        labelErrore.setText(errore);
        labelErrore.setVisible(true);
        return;
    }

    String esito = controller.registraUtente(
        nome, cognome, email, password, genere, descrizione, anniEsperienza,
        utenteSelezionato, chefSelezionato, dataNascita
    );
    if (esito == null) {
        labelErrore.setText("");
        labelErrore.setVisible(false);
        showSuccessMessage("Registrazione completata con successo!");
        onIndietroClick(event);
    } else {
        labelErrore.setText(esito);
        labelErrore.setVisible(true);
    }
}

// RegisterController.java
public String validaRegistrazione(
    String nome, String cognome, String email, String password, String confermaPassword,
    String genere, String descrizione, String anniEsperienza,
    boolean utenteSelezionato, boolean chefSelezionato, LocalDate dataNascita
) {
    // ...validazione campi obbligatori, password, email, tipo account, ecc...
    return valid ? null : messaggioErrore.toString();
}

public String registraUtente(
    String nome, String cognome, String email, String password, String genere,
    String descrizione, String anniEsperienza, boolean utenteSelezionato,
```

```
    boolean chefSelezionato, LocalDate dataNascita
) {
    // ...inserimento nel database e gestione errori...
    return null; // oppure messaggio di errore
}
```

Questa logica garantisce che solo dati corretti e coerenti vengano accettati, migliorando la sicurezza e la qualità dell'esperienza utente. In caso di successo, l'utente viene reindirizzato alla schermata di login per accedere con le nuove credenziali.

4.4 Ruoli e funzionalità: Utente e Chef

Nei paragrafi successivi verranno analizzate nel dettaglio le funzionalità e le interazioni specifiche per ciascun ruolo, evidenziando le differenze tra utente visitatore e chef sia dal punto di vista della GUI che della logica applicativa.

4.5 Interazione con l'Utente

L'interazione con l'utente costituisce uno degli aspetti centrali dell'applicazione UninaFoodLab. Attraverso la GUI, l'utente può accedere a tutte le funzionalità offerte dal sistema, come la consultazione dei corsi, la gestione del proprio profilo, la prenotazione e la registrazione, e l'utilizzo delle dashboard dedicate. Ogni azione viene gestita in modo reattivo e intuitivo, grazie all'integrazione tra i file FXML, i controller e le classi Boundary.

La progettazione dell'interazione è stata pensata per garantire semplicità d'uso, immediatezza delle risposte e chiarezza nei feedback, sia in caso di successo che di errore. Nei paragrafi successivi verranno analizzate nel dettaglio le principali funzionalità e i flussi di interazione, con esempi pratici e riferimenti al codice implementato.

4.5.1 Homepage Utente

La homepage utente rappresenta il punto di ingresso principale per l'utente visitatore dopo il login. È progettata per offrire una panoramica chiara e immediata delle funzionalità disponibili, come la ricerca e la visualizzazione dei corsi, la gestione del profilo, la consultazione dei corsi a cui si è iscritti e il logout.

La struttura della homepage è definita nel file `homepageutente.fxml`, che organizza la GUI in una sidebar laterale per la navigazione rapida e una sezione centrale dedicata ai contenuti dinamici. La sidebar permette di accedere velocemente alle principali funzionalità, mentre la sezione centrale mostra i corsi disponibili, i filtri di ricerca e la paginazione.

L'interazione è gestita dalle classi `HomepageUtenteBoundary.java` e `HomepageUtenteController.java`, che si occupano rispettivamente della gestione dell'interfaccia e della logica applicativa. Ad esempio, la ricerca dei corsi avviene tramite i filtri di categoria e frequenza, e i risultati vengono visualizzati in modo paginato grazie al controller.

La homepage mostra anche le informazioni dell'utente (nome, cognome, immagine profilo) e offre un'esperienza personalizzata, con feedback immediati sulle azioni compiute (spinner di caricamento, messaggi di errore, aggiornamento dinamico dei contenuti).

4.5.2 Iscrizione ai Corsi

L'iscrizione ai corsi è una delle funzionalità centrali per l'utente. La scena `enrolledcourses.fxml` offre una panoramica dei corsi a cui l'utente è già iscritto, con la possibilità di filtrare per categoria e frequenza tramite i rispettivi `ComboBox`. La paginazione consente di navigare tra i corsi in modo semplice e intuitivo.

L'interazione è gestita dalle classi `EnrolledCoursesBoundary.java` e `EnrolledCoursesController.java`. La boundary si occupa di visualizzare i dati e gestire gli eventi della GUI, mentre il controller implementa la logica di caricamento, filtraggio e visualizzazione dei corsi iscritti. Ad esempio, il metodo `loadEnrolledCourses()` recupera dal database i corsi a cui l'utente è iscritto, applica i filtri selezionati e aggiorna dinamicamente la visualizzazione:

```
public void loadEnrolledCourses() {
    if (boundary != null) boundary.showLoadingIndicator();
    Thread t = new Thread(() -> {
        try {
            UtenteVisitatore utente = UtenteVisitatore.loggedUser;
            // Recupera i corsi dal database
            utente.getUtenteVisitatoreDao().RecuperaCorsi(utente);
            List<Corso> corsi = utente.getCorsi();
            // Applica i filtri e aggiorna la GUI
            // ...
        } catch (Exception e) {
            e.printStackTrace();
        }
    });
    t.setDaemon(true);
    t.start();
}
```

La boundary gestisce anche la visualizzazione del numero totale di corsi iscritti, lo spinner di caricamento e la navigazione tra le pagine. L'utente può tornare alla homepage, accedere alla gestione account o effettuare il logout tramite i pulsanti laterali.

Questa organizzazione garantisce un'esperienza utente fluida, con feedback immediati e una gestione efficiente anche in presenza di molti corsi iscritti.

Inoltre, la modularità delle componenti come `CardCorsoBoundary` permette di riutilizzare la stessa logica di visualizzazione dei corsi in diverse parti dell'applicazione, mantenendo coerenza e riducendo la duplicazione del codice. La sicurezza e la coerenza dei dati sono garantite dal caricamento asincrono tramite thread dedicati, che evitano blocchi dell'interfaccia e gestiscono correttamente le eccezioni.

La paginazione e i filtri rendono la navigazione tra i corsi iscritti scalabile anche per utenti con molte iscrizioni, mentre la personalizzazione dell'esperienza (visualizzazione nome, immagine profilo, feedback visivi) contribuisce a rendere l'interazione più efficace e gradevole. Tutte queste caratteristiche sono gestite e coordinate all'interno della scena `enrolledcourses.fxml` e delle relative classi boundary e controller.

4.5.3 Visualizzazione Calendario delle Sessioni

L'utente può consultare il calendario delle lezioni e sessioni direttamente dalla scena `calendardialog.fxml`, che offre una vista mensile interattiva e dettagliata. La GUI è composta da una griglia che rappresenta i giorni del mese, una sidebar con i dettagli delle lezioni selezionate e pulsanti per la navigazione tra i mesi. La logica di visualizzazione e interazione è gestita dalle classi `CalendarDialogBoundary.java` e `CalendarDialogController.java`, che si occupano di popolare il calendario con le sessioni reali dell'utente, gestire la selezione dei giorni e mostrare i dettagli delle lezioni.

Quando l'utente seleziona un giorno, vengono visualizzate tutte le lezioni previste per quella data, con informazioni su orario, durata, tipo di sessione (online o in presenza) e descrizione. Il controller gestisce la mappa delle lezioni e aggiorna dinamicamente la GUI, garantendo un'esperienza reattiva e personalizzata. Esempio di codice per la selezione di una data e visualizzazione delle lezioni:

```
private void selectDate(VBox cell, LocalDate date) {
    if (selectedDayCell != null) { /* ... */ }
    selectedDayCell = cell;
    cell.getStyleClass().add("selected");
    selectedDate = date;
    showLessonDetails(date);
}
```

La modularità della scena consente di adattare la visualizzazione sia per utenti che per chef, mostrando le sessioni pertinenti e permettendo la conferma della presenza per le lezioni in presenza.

4.5.4 Acquisto e Pagamento dei Corsi

L'acquisto di un corso avviene tramite la scena `paymentpage.fxml`, che offre una procedura guidata e sicura per il pagamento online. L'utente può scegliere tra carte di credito salvate o inserire una nuova carta, con validazione automatica dei dati (nome, numero, scadenza, CVC) e feedback immediato sugli errori. La logica di pagamento è gestita dalle classi `PaymentPageBoundary.java` e `PaymentPageController.java`, che si occupano di validare i dati, gestire la persistenza delle carte e iscrivere l'utente al corso selezionato.

La validazione dei dati avviene sia tramite pattern regolari che tramite la classe `CardValidator.java`, che verifica il tipo di carta (Visa/Mastercard) e la validità della scadenza. Esempio di codice per la validazione della scadenza:

```
public static boolean isValidExpiryDate(String expiry) {
    try {
        String[] parts = expiry.split("/");
        int month = Integer.parseInt(parts[0]);
        int year = Integer.parseInt(parts[1]);
        // ...conversione anno e controllo validità...
        LocalDate expiryDate = LocalDate.of(year, month, 1).plusMonths(1).minusDays(1);
        LocalDate today = LocalDate.now();
        return expiryDate.isAfter(today) || expiryDate.isEqual(today);
    } catch (Exception e) {
        return false;
    }
}
```

Al termine della procedura, l'utente riceve un feedback di successo tramite dialog dedicato e il corso viene aggiunto ai corsi iscritti. La modularità della boundary consente di gestire sia carte nuove che salvate, con possibilità di persistenza nel database e visualizzazione delle carte disponibili.

4.5.5 Gestione Account Utente

La gestione dell'account utente è una funzionalità centrale che consente all'utente di visualizzare e modificare i propri dati personali, aggiornare la password, cambiare la foto profilo e accedere alle proprie carte di pagamento. Tutte queste operazioni sono gestite tramite la scena `accountmanagement.fxml`, che offre una GUI intuitiva e suddivisa in sezioni tematiche: informazioni personali, sicurezza, foto profilo e azioni aggiuntive.

La logica applicativa è implementata nelle classi `AccountManagementBoundary.java` e `AccountManagementController.ja`
La boundary si occupa di gestire gli eventi della GUI e di delegare le operazioni al controller, che interagisce con il database per il recupero e la modifica dei dati utente. Ad esempio, all'inizializzazione vengono caricati i dati reali dell'utente loggato e visualizzati nei rispettivi campi:

```
private void loadUserData() {
    if (loggedUser != null) {
        utenteDao.recuperaDatiUtente(loggedUser);
        originalName = loggedUser.getNome();
        originalSurname = loggedUser.getCognome();
        originalEmail = loggedUser.getEmail();
        originalBirthDate = loggedUser.getDataDiNascita();
        boundary.getNameField().setText(originalName);
        boundary.getSurnameField().setText(originalSurname);
        boundary.getEmailField().setText(originalEmail);
        boundary.getBirthDatePicker().setValue(originalBirthDate);
        boundary.getUserNameLabel().setText(originalName + " " + originalSurname);
        boundary.setProfileImages(propic);
    }
}
```

L'utente può modificare i dati personali (nome, cognome, email, data di nascita) e salvare le modifiche, che vengono validate (ad esempio, controllo email valida e età minima) e poi aggiornate nel database. È possibile anche cambiare la password, con verifica della password attuale e conferma della nuova password. In caso di errore, vengono mostrati messaggi di feedback chiari e immediati.

La gestione della foto profilo avviene tramite un file chooser che consente di selezionare un'immagine dal proprio dispositivo. Il controller si occupa di validare il formato, salvare l'immagine nella directory dedicata e aggiornare il percorso nel database, mostrando la preview aggiornata nella GUI:

```
public void changePhoto() {
    FileChooser fileChooser = new FileChooser();
    // ...configurazione fileChooser...
    File selectedFile = fileChooser.showOpenDialog(stage);
    if (selectedFile != null) {
        // ...validazione e copia file...
        loggedUser.setUrl_Propic(relativePath);
        utenteDao.ModificaUtente(loggedUser);
        boundary.setProfileImages(relativePath);
    }
}
```

La scena offre anche la possibilità di accedere alla pagina delle carte utente, tornare alla homepage o ai corsi iscritti, e di effettuare il logout tramite pulsanti dedicati. Tutte le operazioni sono accompagnate da dialog di conferma o successo, garantendo un'esperienza utente sicura e trasparente.

La modularità tra boundary e controller, insieme all'uso di metodi getter/setter e all'integrazione con il database, consente una gestione efficiente e scalabile dell'account utente, con possibilità di estensione per future funzionalità.

4.5.6 Gestione Carte Utente

La gestione delle carte di pagamento è accessibile tramite la scena `usercards.fxml`, che consente all'utente di visualizzare, aggiungere ed eliminare le proprie carte salvate. L'interfaccia è suddivisa in due sezioni principali: il form per l'aggiunta di una nuova carta e la lista delle carte già memorizzate.

La logica applicativa è gestita dalle classi `UserCardsBoundary.java` e `UserCardsController.java`. All'apertura della pagina, vengono caricate dal database tutte le carte associate all'utente e visualizzate nella lista. Ogni carta mostra il nome del titolare, le ultime quattro cifre, la data di scadenza e il circuito (Visa/Mastercard), con una grafica chiara e badge identificativo.

Per aggiungere una nuova carta, l'utente deve compilare il form con nome, numero, scadenza e CVV. La validazione dei dati avviene sia tramite pattern regolari che tramite la classe `CardValidator.java`, che verifica il tipo di carta e la validità della scadenza. Esempio di validazione:

```
if (!CardValidator.isValidCardType(number)) {
    boundary.showFieldError("cardNumber", "Tipo di carta non supportato. Accettiamo solo Visa e Masterca
    return;
}
if (!CardValidator.isValidExpiryDate(expiry)) {
    boundary.showFieldError("expiry", "La carta è scaduta o la data non è valida");
    return;
}
```

Se la carta è valida, viene salvata nel database e associata all'utente tramite le DAO dedicate. Un dialog di successo conferma l'operazione e la lista viene aggiornata in tempo reale.

L'utente può eliminare una carta selezionata dalla lista, con conferma e aggiornamento immediato della visualizzazione. Tutti i campi del form sono dotati di feedback visivi per errori e formattazione automatica (ad esempio, spaziatura del numero carta e formato scadenza MM/AA).

La modularità tra boundary e controller garantisce una gestione sicura e intuitiva dei metodi di pagamento, con possibilità di estensione per future integrazioni (es. impostazione carta predefinita, supporto ad altri

circuiti). L'integrazione con il database assicura la persistenza e la coerenza dei dati tra le varie scene dell'applicazione.