Lingoger è un'applicazione nata per facilitare e velocizzare l'apprendimento delle lingue tramite l'utilizzo di lezioni testuali, contenenti varie informazioni riguardo ad un determinato argomento specificato, e dei test per verificare la comprensione di ciò che è stato studiato.
Le lezioni sono pensate per essere brevi, e allo stesso modo i quiz sono molto rapidi da svolgere, per consentire all'utente di poter utilizzare l'app tutti i giorni, anche quando ha poco tempo a disposizione.
Ogni test propone diverse domande, ciascuna delle quali con tre possibili risposte, di cui solo una corretta.

Il pattern che ho utilizzato è il pattern comportamentale Mediator, in particolare nella parte dell'applicazione che va a gestire lo svolgimento dei test. Il pattern è stato molto utile a non appesantire, tramite l'aggiunta di moltissime dipendenze anche tra i vari elementi, la classe principale TestController.
Ciascun test prevede la presenza di molti buttons: tre per ciascuna domanda più un button di conferma, utilizzato per la correzione, che viene attivato soltanto nel momento in cui l'utente ha completato il test (e quindi ha fornito una risposta per ciascuna domanda). Di conseguenza, invece di aggiungere come detto molte dipendenze caotiche tra i vari buttons, ho sfruttato la classe Mediatore, TestMediator, per gestire le comunicazioni. In particolare, ogni qual volta viene effettuato un click sui buttons, il TestController richiama il mediatore nel metodo che gestisce le azioni su tutti i buttons ad eccezione quello di conferma, buttonClick.
In particolare viene richiamato il metodo del mediatore newClick, che si occupa di registrare in una mappa le risposte alle domande in base al button che viene cliccato, e fa sì che venga selezionata una sola risposta a domanda; inoltre richiama i metodi, sempre nella classe del mediatore, che consentono di abilitare il button di conferma quando necessario.
Inoltre, al click sul button di conferma (gestiti dal metodo confirmationClick in TestController) viene richiamato il metodo checkAnswers in TestMediator. Questo metodo va a validare le risposte date, contando quante sono giuste, in modo da mostrare il punteggio finale all'utente.
Un altro metodo importante è il metodo setButtons in TestMediator, che setta lo stile per i buttons e inoltre va ad associare le istanze dei bottoni che il mediatore deve contenere a quelle del Controller designato, il TestController.

Alcune tecnologie utilizzate all'interno dell'applicativo sono:

- Il pattern MVC (Model-View-Controller) per organizzare intuitivamente i contenuti, separando la logica di Business dall'aspetto grafico, e lasciare così che il Controller gestisca le comunicazioni tra i due (nota: il Controller nel caso dell'MVC fa di per sé la parte del mediatore :))
- Un meccanismo di permanenza basato su un database SQLite: gli utenti, alla registrazione, vengono salvati nella tabella user; il progresso effettuato tramite lo svolgimento delle lezioni e dei test viene salvato

rispettivamente in lezioniSvolte e testSvolti; le informazioni per le varie lezioni sono estrapolate tramite query dalle lezioniNomeLingua, mentre quelle per i test da domandeNomeLingua e risposteNomeLingua.
- Query parametriche, con PreparedStatement, usate per evitare problemi di SQL Injection, insieme a delle Callable di vario tipo eseguite da un thread executor (classe ExecutorProvider) in modo da non appesantire il Main Thread e rallentare di conseguenza l'esecuzione, e inoltre avere la possibilità di estrapolare i risultati ed utilizzarli in altre parti di codice al di fuori delle query. Le query sono contenute nelle classi TestCallables, LessonCallables e userCallables, e create dalla classe QueryCreator.
- SpringSecurity, per salvare le password in modo criptato e sicuro sul database.
- Le librerie per invio delle email di Java (Simple Java Mail, utilizzato nella classe del model EmailSender) in modo da inviare un'email di conferma alla registrazione.
- Diverse regex, nella classe ValidatorUtility, per controllare che alcune stringhe, come quelle dell'email e della password forniti dall'utente fossero nel formato corretto.
- SceneBuilder, per la creazione dei vari file fxml presenti nell'applicazione.
- Css, per la creazione di un tema predefinito che richiami i colori scelti per il logo.

Inoltre, ho utilizzato Maven per la gestione delle dipendenze, e git per salvare man mano le varie modifiche al codice e ripristinarlo a versioni precedenti nel caso di problemi.

Nota: la mascotte dell'applicazione è di proprietà della Nintendo, è stata estrapolata dal videogioco Animal Crossing.