Descrizione del progetto finale



Nome del progetto di laboratorio:

Pasta Lab

Gruppo



Informazioni sul gruppo di laboratorio.

|  |  |
| --- | --- |
| Nome del gruppo: | Pasta Lab |
|  |  |
| Componenti: | \* Balu Sandra  \* Cosma Ada  \* Velea Catalina |
|  |  |
| Github/Gitlab Repo Link: | ## https://github.com/adacosma/PastaTimer   * Github id: *fabcira* * University Gitlab: Fabio Ciravegna ## |

Date



Le date principali del documento.

|  |  |
| --- | --- |
| Data di sottomissione della proposta di progetto | 15.05.2025 |
|  |  |
| Data di accettazione della proposta di progetto | 22.05.2025 |

Descrizione breve



Descrizione della App in una frase.

|  |
| --- |
| PastaLab is an application centered around a timer that gives live directions on how to cook  different types of pasta and create a suited sauce for the selected recipe. |

Definizioni



Di seguito la definizione dei termini, abbreviazioni e acronimi utilizzati.

|  |  |
| --- | --- |
| Termine | Definizione |
| MVVM | (Model-View-ViewModel): An architectural pattern that separates user interface logic (View), business logic (ViewModel), and data access (Model). It enhances maintainability and testability. |
| ViewModel | A lifecycle-aware component that holds UI-related data and business logic. It exposes data to the UI and survives configuration changes. |
| LiveData | An observable data holder class from Android Jetpack that automatically updates the UI when the data changes. |
| Room | A Jetpack library that provides an abstraction layer over SQLite to simplify database access and management. |
| Jetpack Compose | A modern UI toolkit for building native Android interfaces using a declarative approach. |
| Coroutine | A Kotlin feature for asynchronous programming that allows non-blocking, concurrent tasks using lightweight threads. |

# 1. Contesto del progetto

Lunghezza suggerita: 2 pagine

## 1.1. Situazione attuale

Currently, most cooking apps available on the Play Store are very limited in terms of personalization and flexibility. Some work as simple kitchen timers with fixed durations, while others only display general recipes for various dishes. However, these solutions are not made specifically for pasta, and they do not adapt to different types of pasta or ingredients.

For example, cooking times for spaghetti made from durum wheat are different from those for whole grain fusilli, but most apps use a one-size-fits-all approach. Users have to guess or check packaging every time.

In addition, most of these apps do not allow the user to specify personal food preferences. There is usually no way to filter recipes based on allergens (such as milk, eggs, nuts) or dietary choices such as vegetarianism. This can be a serious problem for users with allergies or intolerances, who may accidentally follow a recipe that is not safe for them.

Another limitation is connectivity. Many existing apps rely heavily on internet access, which is needed to load recipes or data from the cloud. If the user is offline (for example, in the kitchen without Wi-Fi) the app becomes useless or only partially functional.

Finally, current cooking timers do not give any indication of what is happening during the cooking process. They simply count down to zero, without telling the user whether the pasta is still undercooked, properly cooked (al dente), or already overcooked. This makes it more difficult to get the desired result, especially for beginners.

## 1.2. Benefici e creazione di valore

Quali saranno i vantaggi / opportunità / risoluzione di problemi che rendono il vostro progetto appetibile / interessante?

Our app, PastaTimer, solves several of the problems described above and provides a more intelligent and personalized cooking experience.

The main feature is a dynamic pasta timer that adapts to the selected pasta type and shows real-time updates during cooking. Instead of a simple countdown, the app displays the estimated cooking state: undercooked, al dente, or overcooked. This helps users know exactly when to stop the cooking process for perfect pasta.

In addition, users can create an account and define their dietary preferences. After signing up and logging in, they can specify allergens (for example, milk, nuts, soy) and choose whether they want to receive only vegetarian recipes. The app will then automatically filter sauce suggestions based on these settings, improving both safety and convenience.

One of the key advantages is offline support. All data (including sauce recipes and pasta types) is stored locally on the device using a Room database. This means that the app works anytime, even without an internet connection.

Our app is also designed with modern Android architecture and good software practices, which makes it efficient, reliable, and easy to maintain. It combines cooking support with user health and personalization, making it useful both for casual users and for people with special dietary needs.

## 1.3. Obiettivi del progetto

Elenco di aspetti specifici e verificabili che realizzerete nella vostra App e che la renderanno “appetibile”.

|  |  |
| --- | --- |
| Aspetto 1 | Provide a pasta timer that adjusts based on the pasta type and visually tracks cooking state. |
| Aspetto 2 | Allow the user to define allergens and preferences (vegetarian) during sign-up. |
| Aspetto 3 | Filter sauce recipes accordingly and display ingredients, avoiding dangerous allergens. |
| Aspetto 4 | Store all data locally using Room; the app works without internet. |
| Aspetto 5 | Use Jetpack technologies like Compose, Navigation, LiveData, and Coroutines. |

# 2. Profilo del progetto

Lunghezza suggerita: 1 pagina

## 2.1. Ambito del progetto

Una descrizione di cosa la vostra App si propone di realizzare, cosa sarà sviluppato e cosa resterà da sviluppare alla fine del lavoro e di come il vostro progetto si legherà ad eventuali servizi e/o server esterni al telefono o all’emulatore.

Pasta Lab is an Android mobile application that helps users cook pasta perfectly while taking into account personal dietary preferences. The app provides a dynamic timer that adapts to each type of pasta and updates the user on the pasta's state throughout the cooking process.

The project implements the following core functionalities:

* User authentication (sign up / log in)
* Preference customization (allergens and vegetarian option)
* Personalized sauce recommendations
* Visual list of pasta types
* Dynamic cooking timer with live status updates
* Favorites system for sauces

The entire app is built to run offline. All data (pasta types, sauces, user settings) is stored locally using Room database. No external servers or internet connectivity are required.

By the end of the project, all core features listed above are implemented. Possible future extensions may include:

* User-submitted recipes
* Cloud sync and cross-device login
* Timer notifications in the background

## 2.2. Profilo della soluzione

Una descrizione di come sarà strutturata la App che è stata realizzata.

The Pasta Lab app follows the **MVVM (Model-View-ViewModel)** architecture to separate concerns between the UI, business logic, and data storage. This structure improves maintainability and makes the codebase easier to manage.

The UI is built using **Jetpack Compose**, which allows us to define dynamic and responsive interfaces in a declarative way. It works well with state-based updates, ensuring the UI automatically reflects changes in the underlying data.

We use **ViewModel** to hold and manage UI-related data. It communicates with the data layer and exposes observable data using **LiveData** or Compose State, allowing real-time updates to the UI.

For local persistence, we use **Room**, an abstraction over SQLite. It stores all the necessary data such as pasta types, sauces, and user preferences directly on the device, ensuring data is retained across app sessions and available offline.

# 3. Vincoli e assunti (Requirements)

Lunghezza suggerita: 2 pagine

## 3.1. Vincoli tecnologici

Quali sono i requirements tecnologici della app

* Functional requirements
* Non-functional requirements

Se i termini functional e non-functional requirements non sono chiari, si veda Wikipedia per una descrizione precisa.

Functional Requirements:

* The app must allow user registration and login
* Users must be able to configure their preferences (allergens and vegetarian option)
* Pasta types and sauces must be visually browsable
* The timer must update its state (undercooked, al dente, overcooked) in real time
* Users must be able to add and remove sauces from favorites

Non-Functional Requirements:

* The app must work offline
* The app must store data persistently using Room
* The UI must be responsive and intuitive
* The app must follow Android best practices and MVVM architecture

## 3.2. Eventuali assunti

* Eventuale assunzione 1

The device has enough storage to maintain a local database

* Eventuale assunzione 2

The user will not attempt SQL injection or malicious input

* Eventuale assunzione 3

The images used for pasta and sauces are correctly placed in the drawable resources

* Eventuale assunzione 4

The user has a modern Android version compatible with Jetpack Compose

# 4. Design



Lunghezza suggerita: 2-3 pagine

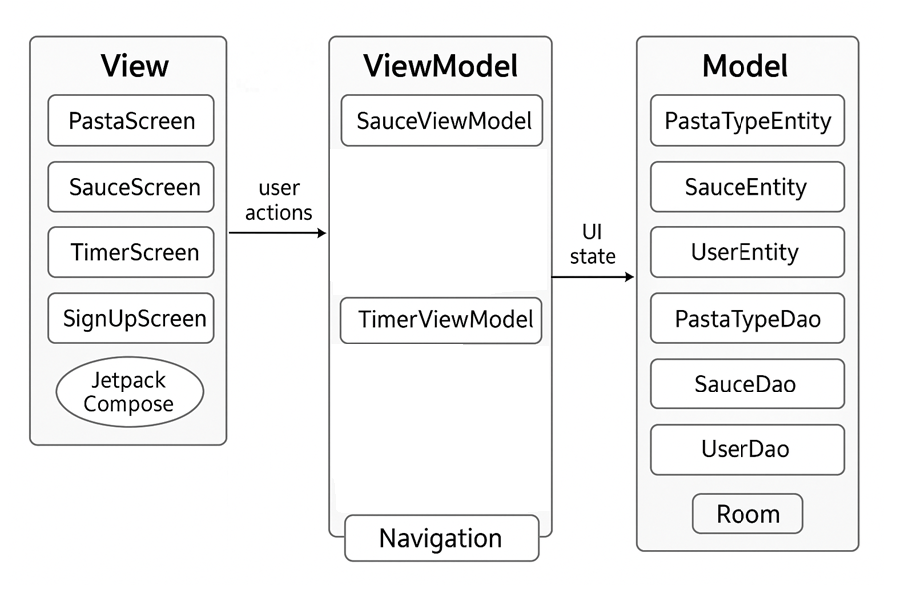
Descrivere le diverse parti del progetto (es. usando un diagramma) e spiegare come in linea di principio l’architettura soddisfa i requirements. Spiegate bene come per esempio il progetto segue il modello MVVM, come le diverse capacità di Jetpack sono usate (Live Data, Room, co-routines, etc.).

**Questa è la parte più importante del lavoro - qui dovete dimostrare di avere capito i concetti chiave del corso!!!**

The Pasta Lab app is structured using the MVVM (Model-View-ViewModel) architecture. It separates the UI logic, application logic, and data management.

Components:

* View (UI Layer): Built entirely with Jetpack Compose. Screens observe state from the ViewModel using either mutableStateOf or LiveData.
* ViewModel: Holds and manages state and business logic.
  + SauceViewModel uses Compose's mutableStateOf.
  + TimerViewModel uses LiveData and coroutines (viewModelScope) for countdown logic.
* Model (Room Layer): Uses Room for local persistence — AppDatabase, DAOs, and entities.



Jetpack Tools Used:

* Jetpack Compose – Declarative UI framework used for building all app screens.
* ViewModel – Manages screen-specific logic and holds UI state across configuration changes.
* LiveData – Used in TimerViewModel to expose countdown time and pasta cooking status.
* mutableStateOf – Used in SauceViewModel for Compose-native reactive state.
* Room – Handles local persistence for pasta types, sauces, and user preferences.
* Coroutines – Used in the timer logic (viewModelScope.launch) to manage asynchronous time updates.

# 5. Implementazione



Lunghezza suggerita: 2-3 pagine

Dettagliare l’implementazione (come avete realizzato il design – es. organizzazione delle classi – il diagramma di navigazione di Android (vedere la lezione su navigation) è un ottimo punto di partenza.

Project structure and class organization:

The main components include:

* SauceDao.kt – manages access to sauces in the Room database
* PastaTypeDao.kt – handles pasta type queries
* UserDao.kt – handles access to user-related data (username, preferences)
* SauceEntity.kt – defines the @Entity for sauces stored locally
* PastaTypeEntity.kt – defines pasta types with name and boil time
* UserEntity.kt – stores allergen and dietary preferences
* AppDatabase.kt – the main Room database builder that provides DAOs
* NavGraph.kt – defines the full Jetpack Compose navigation structure, including route parameters (e.g., "favorites/{username}")
* LogInScreen.kt – UI for user login
* SignUpScreen.kt – UI for account registration
* PersonaliseSuggestions.kt – handles allergen/vegetarian setup
* SharedComponents.kt – reusable UI widgets used during authentication
* MainMenu.kt – central navigation hub for the app
* SauceScreen.kt – displays all sauces with filters
* FavoriteSauceScreen.kt – displays only favorite sauces
* SauceDetailsScreen.kt – shows ingredients and image for a selected sauce
* TimerScreen.kt – shows dynamic countdown and pasta cooking state
* PastaScreen.kt – displays a grid of pasta types with boil time selection
* SauceViewModel.kt – manages sauce filtering, favorites, and user preferences (via mutableStateOf)
* TimerViewModel.kt – manages cooking countdown using LiveData and coroutines

Navigation between screens is implemented using Jetpack Navigation Compose. Routes are defined in NavGraph.kt and include parameterized paths:

* login/{username}
* home/{username}
* sauces/{username}
* favorites/{username}
* details/{sauceName}
* timer/{duration}

Each composable screen extracts parameters using backStackEntry.arguments, for example:

val username = backStackEntry.arguments?.getString("username") ?: ""

NavController is passed between composables to enable transitions and back navigation. For example, the "⬅ Back to Menu" button uses:

navController.navigate("home/{username}")

The app uses two main ViewModels to manage business logic and UI state: SauceViewModel and TimerViewModel.

* SauceViewModel handles all logic related to sauces. It uses Jetpack Compose’s reactive state system (mutableStateOf and mutableStateListOf) to expose:
  + favoriteSauces: list of sauces marked as favorite
  + filteredSauces: sauces filtered according to user preferences (vegetarian, allergens)
  + user: current user state (with allergens and settings)

All of these are exposed using State<T>, which Compose observes automatically to update the UI in real time. The filtering logic is applied internally using custom rules, including allergen keyword matching and vegetarian meat exclusions.

Example:  
val filteredSauces: State<List<SauceEntity>> get() = \_filteredSauces

* TimerViewModel is responsible for managing the countdown timer used to cook pasta. Unlike SauceViewModel, it uses LiveData and coroutines because time-sensitive updates need to be reactive outside of Composables as well. It exposes:
  + timeLeft: LiveData<Int> to track seconds remaining
  + status: LiveData<String> to display whether the pasta is “Undercooked”, “Al Dente”, or “Overcooked”

Countdown updates are handled inside a coroutine using viewModelScope.launch and delay. The view model includes cancelation logic to prevent multiple overlapping timers and supports resetting or restarting the timer based on selected pasta type and cooking duration.

The Pasta Lab app uses Room as its local database solution to store and manage persistent data directly on the user’s device. This ensures that the app functions entirely offline and retains all critical information across sessions.

The Room database is defined in the file AppDatabase.kt using the @Database annotation. It registers the following three entities:

* SauceEntity – stores sauce information such as name, ingredients, image resource name, and whether it is marked as favorite
* PastaTypeEntity – represents different types of pasta, each with a name and boiling time in minutes
* UserEntity – stores user-specific data, such as allergens (as a comma-separated string) and whether the user follows a vegetarian diet

The following Data Access Objects (DAOs) are used to interact with the database:

* SauceDao – provides methods like getAll(), getFavorites(), and updateFavorite(), used to display and manage sauces throughout the app
* PastaTypeDao – supplies pasta types for PastaScreen, allowing the UI to list options with appropriate boiling times
* UserDao – handles saving and retrieving user data related to dietary preferences

These DAOs are exposed through abstract methods in AppDatabase:

abstract fun pastaTypeDao(): PastaTypeDao

abstract fun sauceDao(): SauceDao

abstract fun userDao(): UserDao

Room is initialized using a singleton pattern via the getDatabase(context) function, ensuring that only one instance of the database is created throughout the app’s lifecycle.

UI with Jetpack Compose: All screens are implemented using Jetpack Compose. Core components include:

* LazyVerticalGrid for showing sauces in grid format
* Column and Box layouts for positioning UI elements
* Buttons for navigation and interaction

Each screen uses Compose state (e.g., remember, mutableStateOf) to reflect UI changes instantly. For instance, toggling a sauce as favorite updates the database and automatically refreshes the grid.

The login screen collects user preferences such as allergens and vegetarian mode. These are used in SauceScreen to dynamically filter sauces using viewModel logic:

viewModel.filterSaucesByPreferences(userPreferences)

This ensures that users only see compatible sauces based on their dietary needs.

Room ensures that all data (sauces, preferences, favorites) is stored locally. The app does not require internet access at any point. This makes it reliable in kitchens or travel environments where Wi-Fi may not be available.

* The back button ("⬅ Back to Menu") is implemented on all screens for consistent navigation
* Drawable images are dynamically loaded using image resource names, validated at runtime

The implementation phase strictly followed the design decisions and ensured clean separation of UI, logic, and data access. Although the app does not use a Repository layer, the direct ViewModel → DAO interaction was chosen due to the app’s small and focused scope. The use of Jetpack Compose, ViewModel, Room, and Coroutines enabled a reactive, modern, and offline-first experience.

# 6. Test



Lunghezza suggerita: 2 pagine

Come avete testato l’app. In genere non è sufficiente che solo l’autore la testi. Utenti esterni (possibilmente una decina) sono richiesti per un progetto eccellente.

The app underwent extensive testing throughout its development lifecycle. Testing strategies included:

Manual Testing**:**

Performed by the development team using emulators and physical devices.

Verified all navigation paths work without crashes

Ensured persistent state across app restarts for users, sauces, and preferences

Multiple unit test classes were implemented to validate the app’s key logic layers:

RegistrationValidatorTest.kt

Tests the logic of user registration:

* Empty username → returns proper error
* Duplicate username → returns specific error
* Password mismatch → returns error
* Valid input → returns success

SauceViewModelTest.kt

Tests the logic of filtering sauces based on:

* Allergen sensitivity (e.g., milk, garlic)
* Vegetarian preference
* Combined filtering
* Cases where all sauces are excluded

Filtering was correctly performed according to user dietary settings and sauce ingredients.

TimerViewModelTest.kt

Tests the cooking timer’s behavior:

* Countdown decrements correctly over simulated time
* Status (e.g., "Undercooked", "Al Dente", "Overcooked") updates as time passes
* Timer reset logic correctly restarts countdown and status

Coroutines and LiveData were used effectively to simulate time-based behavior.

UserDaoTest.kt

Tests Room database operations:

* Insertion and retrieval of users
* Validation of user fields (e.g., allergens, vegetarian flag)
* Updating user preferences works and is persisted correctly

Mockito was used to mock DAO interactions and validate data correctness.

# 7. Conclusioni



Lunghezza suggerita: 1-2 pagine

Discutere come i requirements sono stati rispettati nella soluzione finale (facendo riferimento sia ai requirements iniziali e quanto discusso nella sezione di design) e come i tests abbiano dimostrato che il progetto faccia effettivamente quanto promesso

* The final application satisfies all the initial requirements:
* User registration and login work reliably
* Preferences are saved and used to filter sauces
* The timer functions correctly and provides real-time cooking feedback
* Favorites system is functional and persists state
* The app runs entirely offline
* The architecture respects MVVM principles, and the use of Room, LiveData, and Jetpack Compose ensures reactivity, modularity, and maintainability.
* Test feedback confirmed that the application is intuitive and helpful, especially for beginner cooks or users with dietary needs.
* Further improvements could involve background timers, richer image support, and multi-language UI.
* Pasta Lab offers a complete, personalized, and robust cooking experience.