# Abstract

Our app is designed to provide a convenient and flexible learning experience to individuals who want to enhance their knowledge and skills through video-based learning. With a focus on delivering high-quality educational content, we have carefully curated a selection of videos covering various topics, including the two initial skills we are launching with.

One of the unique features of our app is the ability to save videos in the cache, which makes them available to watch even when there is internet variation. This is especially useful for students who may not have reliable access to the internet or professionals who need to learn on the go.

As we continue to develop and improve our app, we will be adding more skills and functionalities, such as the download function in the next release, to make learning even more accessible and convenient. Our ultimate goal is to provide a platform that empowers individuals to learn and grow in their personal and professional lives.

Table of Contents

[Abstract 1](#_Toc128432147)

[Development Stack 3](#_Toc128432148)

[Application Infrastructure 3](#_Toc128432149)

[App Architecture 4](#_Toc128432150)

[Sequence Diagram 6](#_Toc128432151)

[Data model 7](#_Toc128432152)

[Use case 7](#_Toc128432153)

[Process diagram 10](#_Toc128432154)

[Application diagram 11](#_Toc128432155)

# Development Stack

1. Flutter
2. dart
3. Firebase
4. Android Studio

# Application Infrastructure

1. **A client-side mobile app built using Flutter to provide a responsive and smooth user experience.**
2. **A server-side API built using a modern web framework like Node.js, which connects to the Firebase database to fetch and store video links and video thumbnails.**
3. **A cloud-based database like Firebase to store and manage course data, video thumbnails, and video links.**
4. **A content delivery network (CDN) like Amazon CloudFront or Google Cloud CDN to ensure fast and reliable video streaming to users.**
5. **Third-party services like Google Analytics and Crashlytics to monitor app performance and crashes.**

# App Architecture

For this tutor app that plays videos from YouTube and downloads video links from Firebase, we would use a Clean Architecture.

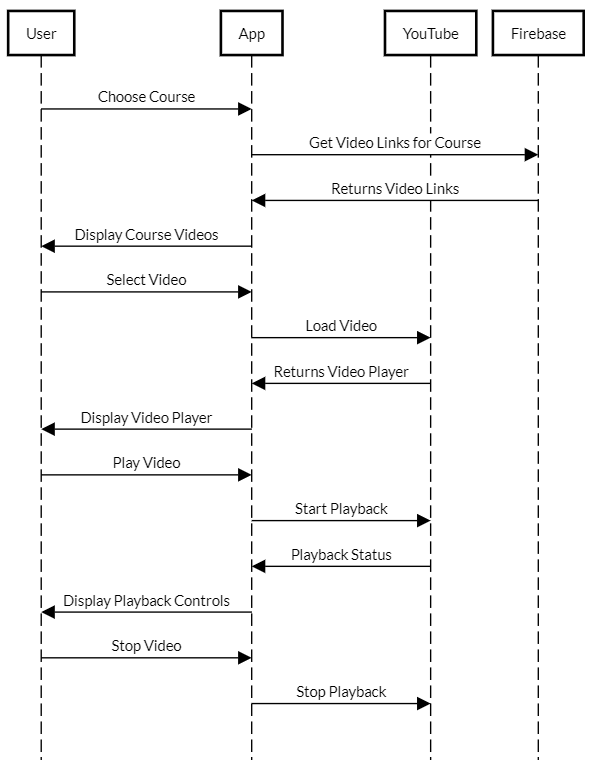
Using the Clean Architecture we would divide the app into several layers such as the presentation layer, domain layer, and data layer. This architecture promotes loose coupling between these layers, which makes it easy to test, maintain, and modify the app in the future.

In the case of this app, the presentation layer could be implemented using Flutter, which would handle the UI and user interactions. The domain layer would handle business logic and use cases, while the data layer would be responsible for accessing data from Firebase.

* Presentation Layer: This layer will contain the UI components that allow users to choose a course and play the video. It will also contain the code to handle the interactions between the UI and the domain layer.
* Domain Layer: This layer will contain the business logic of the app. It will be responsible for downloading the video links from Firebase, playing the video from YouTube, and handling any errors or exceptions that may occur.
* Data Layer: This layer will contain the code to interact with Firebase to download the video links. It will also contain the code to interact with YouTube to play the video. For this purpose, we would use the firebase api and youtube API.

By using this architecture, we can create a robust and scalable app that is easy to maintain and test.

# Sequence Diagram



# Data model

1. Course: represents a course that contains one or more videos, with attributes such as name, description, and image.
2. Video: represents a video that can be played within a course, with attributes such as title, description, URL (e.g., YouTube video ID), and duration.



# Use case

* Student



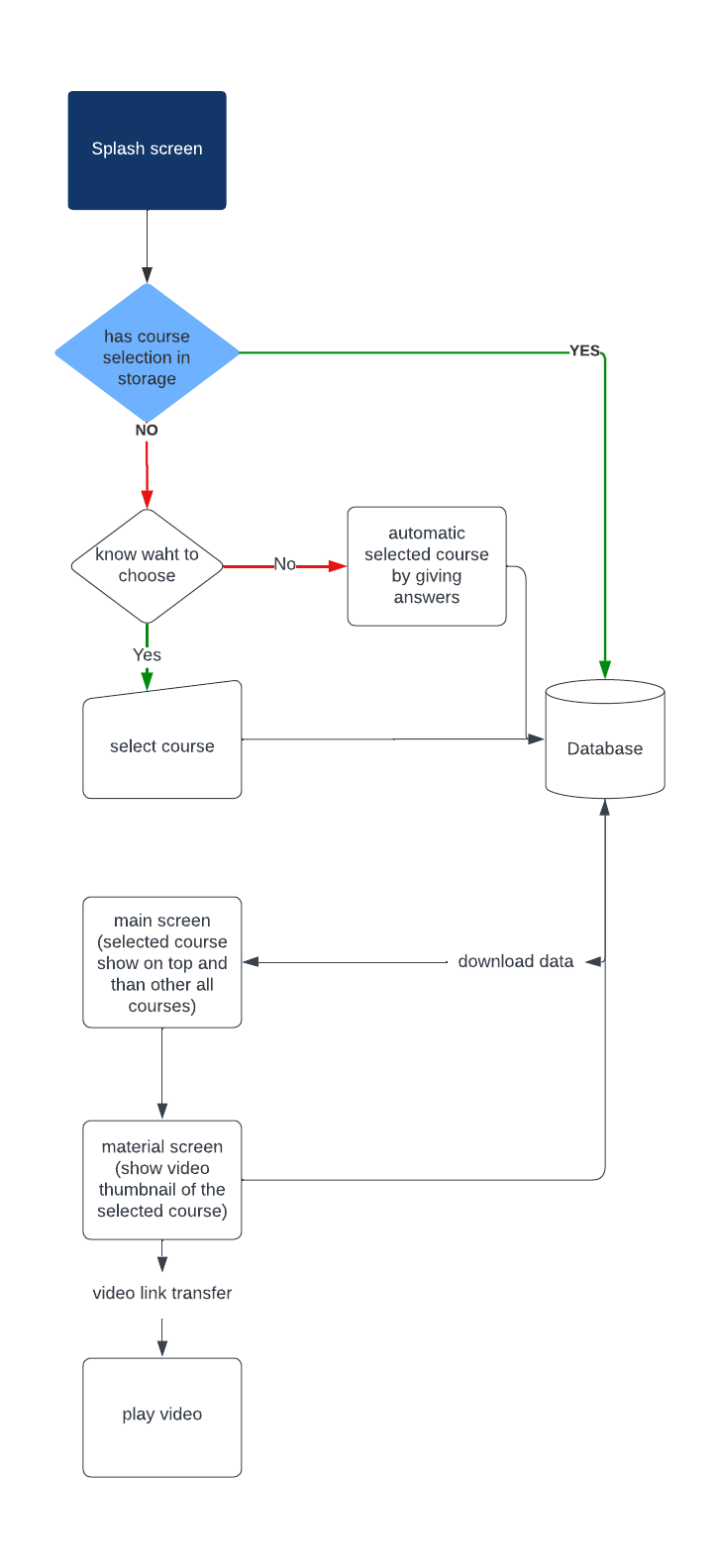
* Software Engineer



* Teacher



# Process diagram



# Application diagram

┌─────────────────------------------------------------------─┐

│ UI/Presentation │

│ Layer │

└─────────────────-------------------------------------------─┘

▲

│

┌─────────────────────------------------------------------─┐

│ Domain/Business │

│ Layer │

└─────────────────────------------------------------------─┘

▲

│

┌───────────────────-------------------------------------───┐

│ Data/Infrastructure │

│ Layer │

└───────────────────-------------------------------------───┘

The diagram shows three main layers: the UI/Presentation Layer, the Domain/Business Layer, and the Data/Infrastructure Layer.

The UI/Presentation Layer handles the user interface and user interaction of the app. It communicates with the Domain/Business Layer to retrieve or send data and business logic. This layer is responsible for rendering the user interface and handling user inputs, such as selecting a course or playing a video.

The Domain/Business Layer is where the app's business logic resides. It contains the use cases that the app needs to accomplish and the data models and repositories required to perform these tasks. This layer communicates with the Data/Infrastructure Layer to retrieve or send data from external sources.

The Data/Infrastructure Layer handles the data access and storage for the app. It includes data repositories, network clients, and any external services or APIs required to perform data operations. This layer communicates with the Domain/Business Layer to retrieve or send data from external sources.

This diagram follows the Clean Architecture principles, where each layer has a clear responsibility and is independent of the other layers. The layers are separated by boundaries that allow for testability, maintainability, and flexibility.