Chapter 5
Frontend

In the context of mobile app development, the term "front end" refers to the user interface (UI) and user experience (UX) components of the app that users directly interact with on their mobile devices. It includes everything that is visible and interactive in the app, from the visual design to the user interactions and animations. The front end of a mobile app typically involves the following aspects:

- **UI Design**: This involves creating the visual design of the app's screens or views, including the layout, typography, colors, and graphical assets. It focuses on creating an appealing and consistent visual style that aligns with the app's branding and target audience.
- **User Interactions**: Front-end development includes implementing the user interactions within the app, such as tapping buttons, scrolling, swiping, and entering data through forms. It involves capturing user input and responding to it appropriately, ensuring a smooth and intuitive user experience.
- **App Screens and Views**: Mobile apps are composed of different screens or views that display specific information or perform certain tasks. The front-end development involves creating and connecting these screens, defining their structure and flow, and managing the navigation between them.
- **Layout and Components**: Front-end developers work with layout systems and UI components to structure and organize the content on the app's screens. This includes using containers, grids, lists, and other layout techniques to present information in a visually pleasing and usable manner.
- **Animations and Transitions**: Front-end development may involve incorporating animations and transitions to enhance the user experience and make the app feel more dynamic. This includes animating UI elements, creating smooth transitions between screens, and providing visual feedback for user interactions.
- **Mobile Platform APIs**: Mobile app front-end development often involves utilizing platform-specific APIs and frameworks provided by the mobile operating systems (such as iOS or Android). These APIs allow developers to access device features, such as camera, location, sensors, and push notifications, to enhance the functionality and user experience of the app.
- **Cross-Platform Frameworks**: In some cases, front-end development for mobile apps involves using cross-platform frameworks like React Native, Flutter, or Xamarin. These frameworks allow developers to write a single codebase that can be used to build apps for multiple platforms, reducing development time and effort.

Overall, the front end of a mobile app focuses on creating an engaging and intuitive user experience by designing and implementing the app's UI, user interactions, and visual elements. It requires a combination of design skills, coding knowledge, and understanding of the mobile platform's guidelines and best practices.

What we used in our project:

UI Design: Figma

Figma is a cloud-based design and prototyping tool used for creating user interfaces (UI) and collaborating on design projects. It provides a platform for designers to create, share, and iterate on design mockups, wireframes, and interactive prototypes. Here are some key features and benefits of Figma:

- **Cloud-Based Collaboration:** Figma is cloud-based, which means designers can access their projects from anywhere with an internet connection. Multiple team members can collaborate in real-time, making it easy to work together on design projects, provide feedback, and make changes simultaneously.
- **Design Tools:** Figma offers a wide range of design tools and features to create UI designs. It includes vector editing tools, advanced typography controls, color management, image editing capabilities, and more. Designers can create reusable components and styles to maintain consistency across their designs.
- **Prototyping and Interactions:** Figma allows designers to create interactive prototypes by defining transitions, animations, and user flows. Designers can link screens and components to create clickable prototypes, simulate user interactions, and demonstrate the app's functionality and user experience.
- **Collaboration and Feedback:** Figma provides features for sharing designs and collecting feedback from stakeholders and team members. Designers can invite collaborators to view and comment on designs, making it easy to gather input, iterate on designs, and track changes. Comments and annotations can be added directly on the design canvas.
- **Version History and Design History:** Figma maintains a complete version history of design files, allowing designers to review and revert to previous versions if needed. It also provides a design history feature that allows users to see how a design has evolved over time and view specific changes made by collaborators.
- **Developer Handoff:** Figma includes features to facilitate the handoff of design assets to developers. Designers can generate design specs, export assets, and access design details such as dimensions, colors, and typography information. This simplifies the process of translating designs into code.
- **Plugins and Integrations:** Figma has a robust ecosystem of plugins and integrations that extend its functionality. Designers can use plugins to

automate repetitive tasks, access design assets from external libraries, and integrate with popular tools and platforms such as Jira, Slack, and GitHub.

- **Cross-Platform Support:** Figma is available for macOS, Windows, and as a web-based application. This cross-platform support enables designers to work on different operating systems without compatibility issues.

Figma is widely used by individual designers, design teams, and organizations of all sizes. It offers a free plan with limited features and pricing options for more advanced features and team collaboration. Its collaborative nature, ease of use, and comprehensive set of design tools make it a popular choice for UI/UX designers.

Project management: Scrum and Agile

Scrum and Agile are project management methodologies that emphasize flexibility, collaboration, and iterative development. They are widely used in software development and other industries to improve productivity, deliver value, and respond to changing requirements. Here's an overview of Scrum and Agile:

Agile Methodology:

Agile is an overarching philosophy and set of principles for software development that promotes adaptive planning, evolutionary development, continuous improvement, and flexible responses to change. Agile methodologies value individuals and interactions, working software, customer collaboration, and responding to change over following rigid plans and processes.

Agile methodologies typically involve:

- **Iterative Development:** Projects are divided into small iterations or increments called "sprints" that typically last from one to four weeks. Each sprint aims to deliver a potentially shippable increment of the product.
- **Collaborative Teams:** Cross-functional teams work closely together, including developers, testers, designers, and stakeholders. Collaboration, communication, and self-organization are essential for Agile teams.
- **Product Backlog:** The product requirements are captured in a prioritized list called the product backlog. This backlog is dynamic and evolves throughout the project as new requirements and insights emerge.
- **User Stories:** Requirements are often expressed as user stories, which capture the perspective of end-users or stakeholders. User stories describe the desired functionality in a concise format that can be easily understood and estimated by the team.
- **Continuous Integration and Testing:** Agile teams emphasize continuous integration, frequent testing, and regular feedback to ensure the quality and

stability of the software. Automated testing and continuous delivery practices are often employed.
- **Adaptability:** Agile methodologies prioritize responding to change over following a rigid plan. Teams regularly inspect and adapt their processes, requirements, and solutions based on feedback and changing circumstances.

Scrum Framework:

Scrum is a specific Agile framework that provides a structured approach to software development. It is based on iterative and incremental development and focuses on empowering self-organizing teams. Scrum incorporates several key elements:

- **Scrum Team:** A cross-functional team consisting of a product owner, a Scrum master, and development team members collaborates to deliver valuable increments of the product.
- **Sprints:** Development work is organized into fixed-duration sprints, typically lasting two to four weeks. Each sprint begins with a sprint planning meeting and ends with a sprint review and retrospective.
- **Product Backlog and Sprint Backlog:** The product backlog contains a prioritized list of features, enhancements, and bug fixes. Before each sprint, the team selects a subset of items from the product backlog to be completed during the sprint, creating a sprint backlog.
- **Daily Stand-up Meetings:** Daily stand-up meetings (also called daily scrums) provide a short and focused opportunity for team members to synchronize their activities, discuss progress, and identify any impediments.
- **Sprint Review and Retrospective:** At the end of each sprint, a sprint review is conducted to demonstrate the completed work to stakeholders. This is followed by a sprint retrospective to reflect on the team's performance and identify areas for improvement.
- **Scrum Artifacts:** Scrum utilizes artifacts like burn-down charts, task boards, and product increment to visualize progress and help the team manage their work effectively.

Scrum provides a structured framework for Agile development, emphasizing transparency, inspection, and adaptation. It promotes a collaborative and iterative approach, enabling teams to deliver value more effectively and adapt to changing requirements.

Both Agile and Scrum are widely adopted methodologies in software development, offering flexibility, collaboration, and a focus on delivering customer value. However, it's important to note that Agile is a broader methodology encompassing various frameworks, while Scrum is a specific framework within the Agile approach.

UI Implementation: UIKit

The UIKit framework is a fundamental framework for building user interfaces in iOS apps. It provides a set of classes, protocols, and other resources that help developers create and manage graphical user interfaces (GUI) for iOS apps. Here's an overview of some key components of the UIKit framework:

- **Views**: UIKit provides a variety of view classes (e.g., UIView, UIImageView, UIButton, UILabel) that serve as the building blocks for creating the visual elements of your app's user interface.
- **View Controllers**: UIKit includes view controller classes (e.g., UIViewController, UITableViewController, UINavigationController, UITabBarController) that help manage the presentation and behavior of your app's user interface. View controllers handle user interactions, manage the lifecycle of views, and coordinate navigation and transitions between views.
- **Controls**: UIKit offers various control classes (e.g., UIButton, UITextField, UISwitch, UIStepper) that allow users to interact with your app. These controls handle user input, respond to events, and provide ways to gather data or trigger actions.
- **Navigation**: UIKit provides navigation-related classes (e.g., UINavigationBar, UINavigationItem, UINavigationController) to facilitate hierarchical navigation within your app's interface. These classes enable the creation of navigation stacks and handle the display of navigation bars, buttons, and other navigation-related elements.
- **Layout and Auto Layout**: UIKit includes layout-related classes (e.g., UIStackView, UILayoutGuide, NSLayoutConstraint) that help you manage the placement and sizing of views within your app's user interface. Auto Layout, a system provided by UIKit, offers a powerful constraint-based layout system that allows for flexible and adaptive user interfaces.
- **Table Views and Collection Views**: UIKit provides table view (UITableView) and collection view (UICollectionView) classes that help you display and manage lists or grids of content. These classes support data-driven display, custom cell configurations, and interaction handling.
- **Drawing and Graphics**: UIKit includes classes for working with graphics and drawing in your app's user interface. Classes like UIColor, UIImage, and CGContext allow you to work with colors, images, and custom drawings.
- **Text and Typography**: UIKit provides classes for working with text and typography. Classes like UILabel, UITextField, and UITextView offer text display and editing capabilities, while typography-related classes (UIFont, NSParagraphStyle, NSAttributedString) allow you to control text appearance and formatting.

These are just some of the key components of the UIKit framework. The framework offers many more classes, protocols, and resources that contribute to building rich and interactive user interfaces in iOS apps. For detailed information and usage examples, refer to the official Apple documentation for UIKit.

Programming languages: Swift

The programming language used is Swift. Swift is a powerful and intuitive programming language developed by Apple for building applications on various Apple platforms, including iOS, macOS, watchOS, and tvOS. It is designed to be easy to learn and write, while also providing robust performance and safety features. Here are some key features and aspects of Swift:

- **Modern and Safe**: Swift incorporates modern language features and concepts, making it easier to read, write, and maintain code. It provides built-in safety features that help developers avoid common programming errors and reduce the likelihood of crashes and bugs.
- **Syntax and Expressiveness**: Swift offers a clean and expressive syntax that is concise and readable. It provides features like type inference, optionals, closures, generics, and pattern matching, allowing developers to write expressive and efficient code.
- **Static Typing**: Swift is statically typed, which means variables and expressions have a type that is checked at compile-time. This helps catch errors early in the development process and improves code reliability and performance.
- **Inferencing and Type Safety**: Swift utilizes type inference to automatically deduce the type of variables and expressions based on their initial values. This reduces the need for explicit type annotations, while still ensuring type safety and catching potential type mismatches.
- **Optionals**: Swift introduces the concept of optionals to handle the absence of a value. Optionals allow developers to write safer code by explicitly stating when a value may be nil (absent) and providing mechanisms to safely unwrap and handle optional values.
- **Memory Management**: Swift incorporates automatic memory management using Automatic Reference Counting (ARC). It automatically tracks and manages the allocation and deallocation of memory for objects, relieving developers from manual memory management tasks.
- **Functional Programming Support**: Swift supports functional programming paradigms with features like higher-order functions, closures, immutability, and functional composition. This allows developers to write more concise and expressive code, leveraging functional programming concepts.
- **Interoperability**: Swift is designed to work seamlessly with existing Objective-C code. It provides interoperability, allowing developers to use both

Swift and Objective-C within the same project. This makes it easy to adopt Swift gradually and leverage existing codebases and frameworks.

- **Playgrounds**: Swift Playgrounds is an interactive development environment that allows developers to experiment, prototype, and learn Swift code. It provides an interactive and visual way to see the results of code execution and encourages iterative development and exploration.
- **Open Source**: Swift is an open-source language, allowing the community to contribute to its development and improvement. The open-source nature of Swift has resulted in the growth of an active and supportive community, with numerous libraries, frameworks, and resources available for developers.

Swift has gained popularity among developers due to its modern syntax, safety features, performance, and support from Apple. It continues to evolve with regular updates and new features introduced in each version. The official Swift website and Apple's Swift documentation provide comprehensive resources to learn and explore the language further.

FrameWorks

Combine

Combine is a framework introduced by Apple that provides a declarative approach for handling asynchronous events and managing streams of values in Swift. It is designed to work seamlessly with Swift's functional and reactive programming paradigms, allowing developers to write concise and composable code for handling asynchronous operations and data flow.
Key features and concepts of Combine include:

- **Publisher-Subscriber Model:** Combine follows the publisher-subscriber pattern, where publishers emit values over time, and subscribers receive and react to those values. Publishers represent a sequence of values or events, while subscribers define what to do with those values.
- **Operators:** Combine offers a wide range of operators that allow you to transform, filter, combine, and manipulate streams of values. Operators enable you to perform various data processing operations, such as mapping values, filtering out unwanted elements, merging multiple streams, and more.
- **Cancellable:** Combine provides the Cancellable protocol, which allows you to cancel or dispose of subscriptions to stop receiving values from publishers. This helps manage resources and prevent memory leaks.
- **Error Handling:** Combine has built-in error handling capabilities. Publishers can emit error events, and subscribers can handle and react to those errors using operators like catch, retry, and replaceError.

- **Schedulers:** Combine introduces schedulers to control the execution context and timing of events. Schedulers allow you to specify where and when publishers emit values and where subscribers receive and process those values. Schedulers are useful for managing concurrency, handling background tasks, and controlling event dispatching.
- **Combine Framework Integration:** Combine integrates seamlessly with other Apple frameworks, such as URLSession, NotificationCenter, and Core Data. It provides publishers and operators that allow you to work with asynchronous operations in these frameworks using Combine's declarative approach.

Combine brings many benefits to Swift development, including:

- **Simplified Asynchronous Programming:** Combine provides a unified approach to handling asynchronous tasks, making it easier to reason about and manage complex asynchronous operations in a declarative manner.
- **Code Reusability and Composition:** Combine's operators and functional programming style enable code reuse and composability. You can chain operators together to build complex data pipelines and create reusable components for handling data flow.
- **Error Handling and Resilience:** Combine's built-in error handling mechanisms make it easier to handle errors and build resilient code. You can use operators to handle errors, retry operations, or recover from failures gracefully.
- **Concurrency Control:** Combine's schedulers allow you to control the execution context and concurrency of events. You can specify when and where publishers emit values and handle subscriptions on specific dispatch queues or execution contexts.

Combine is primarily used in iOS, macOS, watchOS, and tvOS app development to handle asynchronous operations, reactively update UIs, and manage data flow between components.

```
func createRecord(for user: User) {
DatabaseManager.shared.collectionUsers(add: user)
  .sink { [weak self] completion in
            if case .failure( let error ) = completion {
            self?.error = error.localizedDescription
          }
   } receiveValue: { state in
         print("Adding user record to database: \(state)")
  }
  .store(in: &subscriptions)
}
```

MAPKit

MapKit is a framework provided by Apple that allows developers to integrate interactive maps into their iOS, macOS, watchOS, and tvOS applications. It provides a set of classes, protocols, and APIs that enable developers to display maps, annotate locations, calculate routes, and interact with map data. Here are some key features and capabilities of MapKit:

- **Map Display:** MapKit allows you to embed interactive maps within your app's user interface. You can display different types of maps, including standard, satellite, and hybrid views, with zooming and panning capabilities.
- **Annotations:** MapKit enables you to add annotations to the map, such as pins, callouts, or custom markers, to mark specific locations or points of interest. Annotations can display additional information, images, or custom views when tapped by the user.
- **Geocoding and Reverse Geocoding:** MapKit provides geocoding and reverse geocoding services. Geocoding allows you to convert an address or place name into geographic coordinates (latitude and longitude). Reverse geocoding allows you to retrieve the address or place information based on given coordinates.
- **Overlays:** MapKit supports overlays, which are graphical shapes or paths drawn on the map. Overlays can be used to highlight areas, draw routes, display polygons, or provide other visual representations on the map.
- **Map Interaction:** MapKit allows users to interact with the map through gestures, such as pinching to zoom, dragging to pan, and tapping to select annotations. You can customize the behavior of these gestures and handle user interactions using delegate methods.
- **Map Region and Camera:** MapKit provides methods to set and control the visible region and camera perspective of the map. You can specify the center coordinate, zoom level, and viewing angle to control the initial view or programmatically adjust the map's display.
- **Map Overlays and Tiles:** MapKit allows you to overlay custom map tiles or layers on top of the standard map view. This enables the integration of third-party map data, such as custom map styles, weather information, or additional geographic data.
- **Route Calculation and Directions:** MapKit offers functionality to calculate routes and provide directions between locations. You can request and display driving, walking, or transit directions, and customize the appearance and behavior of the route overlays.
- **Map Clustering:** MapKit supports clustering annotations to improve performance and user experience when displaying a large number of annotations in a small area. Clustered annotations group together based on proximity and expand when zoomed in.

MapKit provides a range of customization options, allowing you to customize the appearance of the map, annotations, overlays, and user interactions to match the style and needs of your app. It also integrates with other Apple frameworks and services, such as Core Location for retrieving the user's current location or Core Animation for advanced map animations.

MapKit simplifies the process of integrating maps and location-based features into your app, making it easier to create engaging and location-aware experiences for your users.

You can integrate maps into your app using either a MKMapView or a MKMap ViewController. The map view provides more flexibility as you can embed it within any view hierarchy, while the map view controller is a standalone view controller dedicated to displaying a map. When dealing with a large number of annotations, MapKit offers built-in clustering functionality. Clustering groups nearby annotations into a single cluster annotation, which helps improve performance and maintain map clarity.

MapKit supports overlays to draw custom shapes on the map. Overlays can be used to highlight boundaries, display custom routes, or visualize geographical data. You can create custom overlay objects and add them to the map view.

Vision

The Vision framework is a powerful framework provided by Apple that allows developers to incorporate computer vision and image analysis capabilities into their iOS, macOS, and tvOS applications. It provides a wide range of tools and functionality to perform tasks such as image recognition, object detection, face analysis, text detection, barcode scanning, and more. Here are some key features and capabilities of the Vision framework:

- **Image Analysis:** The Vision framework provides a set of APIs to analyze and process images. You can perform tasks like face detection, face tracking,

facial landmark detection, face identification, and face expression analysis. It also includes APIs for detecting and tracking objects within images.

- **Text Detection and Recognition**: Vision allows you to detect and recognize text within images. It can identify text regions, extract the textual content, and provide information about the location, size, and orientation of the detected text.
- **Barcode and QR Code Detection:** The Vision framework enables you to detect and read barcodes and QR codes from images or live video streams. It supports various barcode formats, such as UPC, Code 128, QR code, and more.
- **Image Classification:** Vision provides APIs for image classification, allowing you to categorize images into predefined or custom classes. You can train and create custom models using machine learning techniques or use pre-trained models available in Core ML.
- **Image Alignment:** Vision includes features for image alignment, which can be useful for tasks like aligning images, creating panoramas, or detecting image similarities.
- **Core ML Integration:** The Vision framework seamlessly integrates with Core ML, which is Apple's machine learning framework. You can leverage pre-trained machine learning models, create custom models, or perform on-device model inference for various computer vision tasks.
- **Real-time and Offline Processing:** Vision offers real-time image analysis capabilities, allowing you to process live video feeds or image streams in real-time. It also supports offline processing, where you can analyze images stored locally on the device.
- **Performance Optimization:** Vision incorporates optimizations for efficient image processing, leveraging the device's hardware capabilities, such as the GPU, to maximize performance and minimize processing time.

The Vision framework simplifies the integration of computer vision capabilities into your applications. It provides high-level APIs and abstractions, reducing the complexity of implementing computer vision algorithms from scratch. With Vision, you can develop applications that can recognize objects, detect faces, read text, and perform various image analysis tasks, opening up possibilities for augmented reality, image recognition, and other vision-based applications.

```
guard let resnetModel = try? VNCoreMLModel(for: resnet().model)
else{ fatalError("Loading CoreML Model Failed") }

let request = VNCoreMLRequest(model: resnetModel){ (request, error) in
  guard let results = request.results as? [VNCoreMLFeatureValueObservation] else {
      fatalError("Model failed to process image.")}
    }
```

```
request.imageCropAndScaleOption = .scaleFill

let handler = VNImageRequestHandler(ciImage: image)
do {
    try handler.perform([request])
}catch {
    print(error)
}
```

Core ML

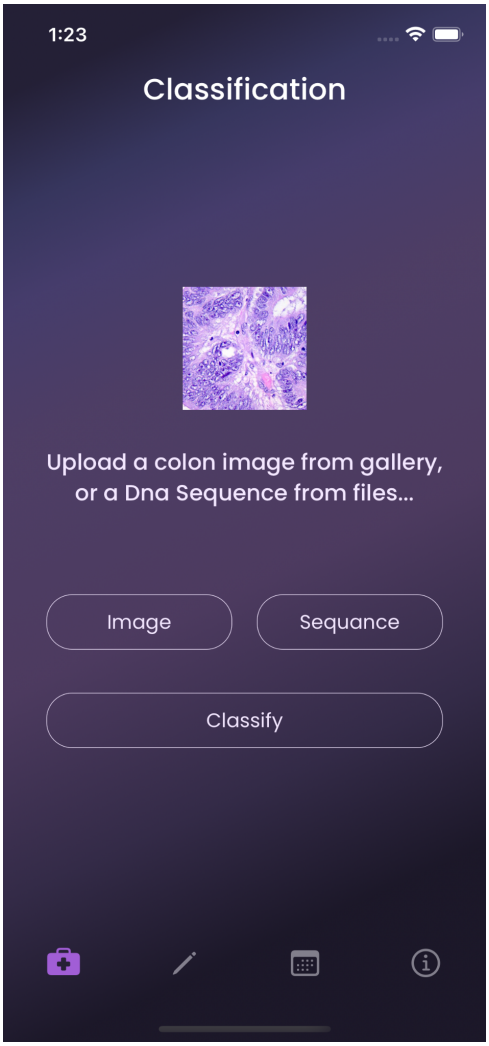
Core ML is a framework provided by Apple that allows developers to integrate machine learning models into their iOS, macOS, watchOS, and tvOS applications. It provides a unified and streamlined approach for running machine learning algorithms on-device, enabling real-time and offline inference without relying on server-side processing. Here are some key features and concepts of Core ML:

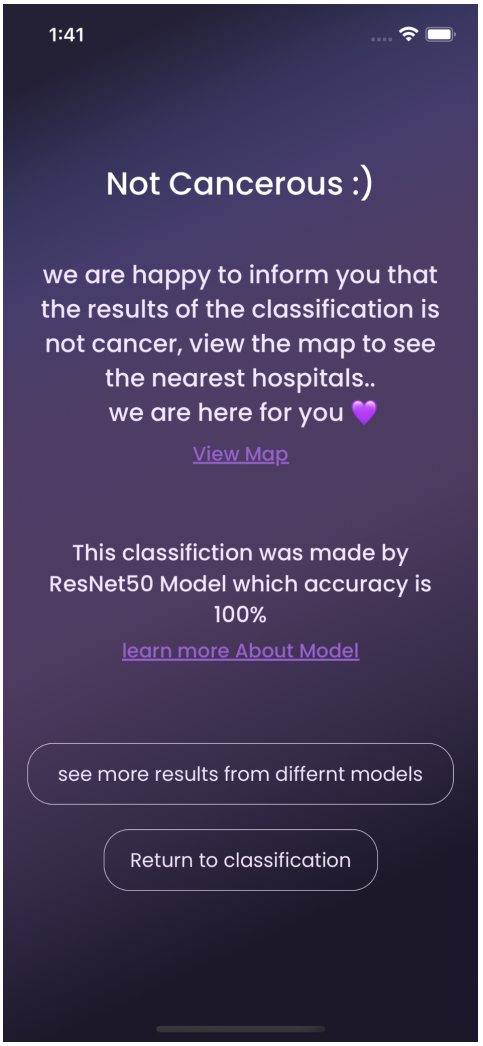
- **Model Integration:** Core ML allows you to integrate pre-trained machine learning models into your app. These models can be created using popular machine learning frameworks such as TensorFlow, Keras, PyTorch, or scikit-learn, and then converted to the Core ML format using conversion tools provided by Apple.
- **On-Device Inference:** Core ML enables you to perform machine learning inference directly on the user's device, leveraging the device's CPU, GPU, or dedicated neural engine for efficient and fast model execution. This eliminates the need for constant network connectivity and reduces latency associated with server-based inference.
- **Supported Model Types:** Core ML supports a variety of machine learning model types, including classification models, regression models, object detection models, natural language processing models, and more. It supports popular model file formats such as TensorFlow Lite (.tflite), ONNX (.onnx), and custom formats using Apple's Neural Network Exchange (NNC) format.
- **Model Optimization:** Core ML includes optimization techniques to improve model performance, such as model quantization (reducing the model size and computation requirements), model pruning (removing unnecessary model parameters), and model caching (enhancing inference speed for recurrent model use).
- **Privacy and Security:** Core ML prioritizes user privacy and data security. The framework performs on-device inference, ensuring that sensitive user

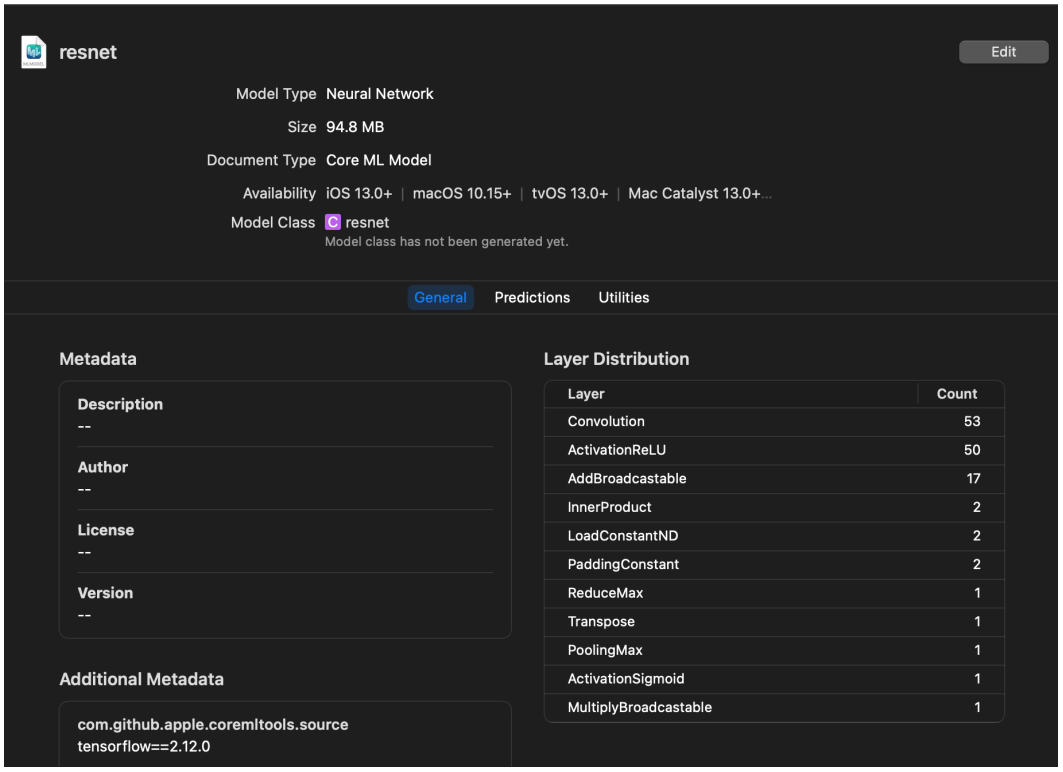
data remains on the device without being transmitted to external servers. Additionally, Core ML models can be encrypted to protect proprietary or sensitive information.

- **Integration with Other Apple Technologies:** Core ML seamlessly integrates with other Apple frameworks and technologies, such as Vision for computer vision tasks, Natural Language for natural language processing, and Accelerate for high-performance mathematical computations.
- **Core ML Tools:** Core ML is supported by a set of developer tools, including Xcode's Core ML Model Compiler, which helps optimize and convert trained models into the Core ML format. Xcode also provides a Model Visualizer for inspecting and debugging Core ML models.

Core ML empowers developers to leverage the power of machine learning within their apps, enabling features such as image recognition, object detection, sentiment analysis, language translation, and more. It brings the benefits of machine learning to the edge, allowing applications to provide intelligent and personalized experiences while maintaining user privacy and offline functionality.





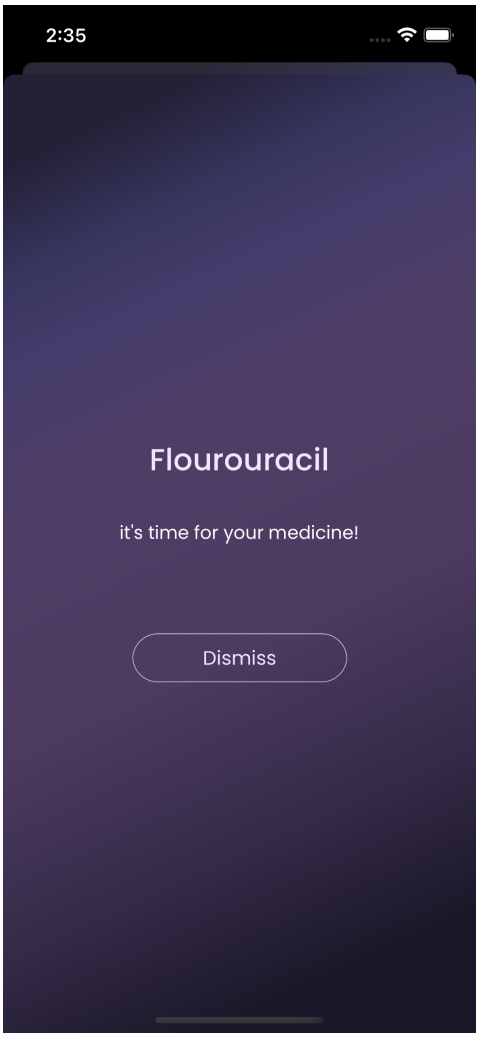


User Notifications

UserNotifications is a framework provided by Apple for managing and delivering notifications to the user in iOS and macOS applications. It allows you to schedule, present, and handle various types of notifications, including local notifications and remote notifications (also known as push notifications). Here are the key features and functionalities of UserNotifications:

- **Local Notifications:** UserNotifications enables you to schedule and present local notifications on the user's device. Local notifications are triggered by your app and can include text, sound, badges, and custom actions. They can be used to remind users of events, display important information, or engage users with interactive content.
- **Remote Notifications (Push Notifications):** UserNotifications provides support for remote notifications, which are sent from a remote server to your app. These

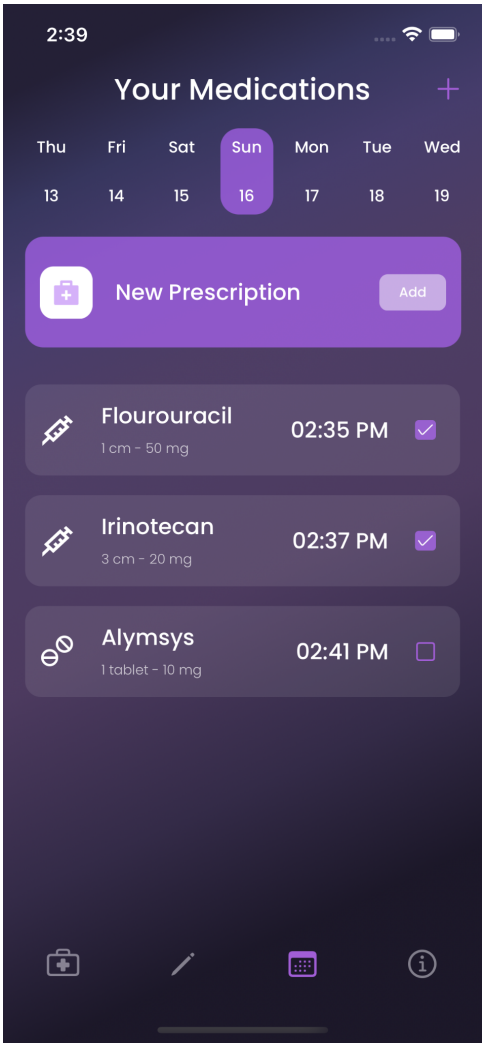
notifications are delivered even when your app is not running, allowing you to engage with users and provide timely updates or information. To use remote notifications, you need to configure your app for push notifications and handle the received notifications using the UserNotifications framework.

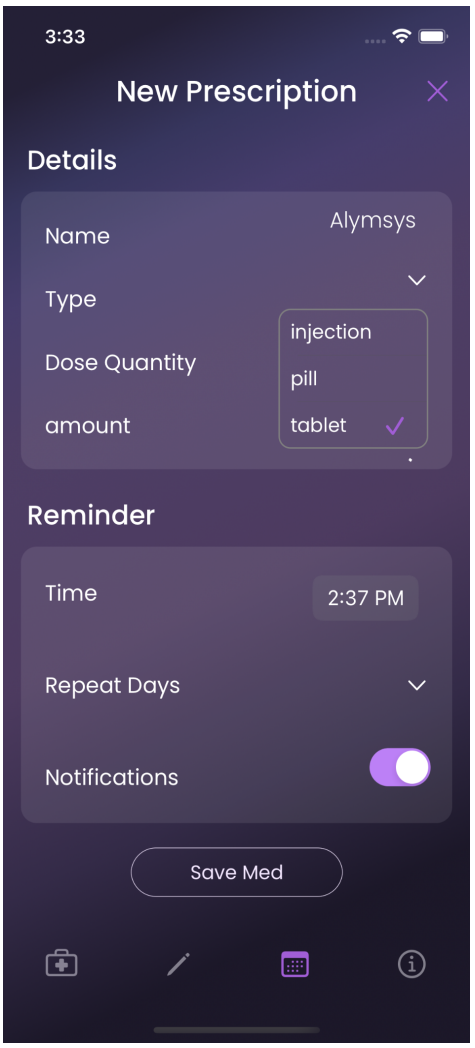


- **Notification Content:** You can customize the appearance and content of your notifications using the UserNotifications framework. You can include titles, subtitles, body text, and attachments such as images or videos. Additionally, you can add custom actions to allow users to interact with the notification, such as dismissing it or performing specific actions directly from the notification.
- **Notification Management:** UserNotifications provides APIs to manage notifications, including scheduling, updating, and canceling notifications. You can schedule notifications to be delivered at specific times or trigger them based on user interactions or specific events in your app. You can also update existing notifications with new content or cancel them if they are no longer relevant.
- **Notification Handling:** When a notification is delivered to your app, you can use the UserNotifications framework to handle the notification and perform custom actions. You can implement delegate methods to receive callbacks when a notification is presented or interacted with by the user. This allows you to respond to user actions, navigate to specific parts of your app, or perform background tasks triggered by notifications.

- **Notification Settings and Permissions:** UserNotifications provides APIs to manage notification settings and permissions. You can request user permission to deliver notifications, check the current authorization status, and open the system settings screen to allow users to modify notification preferences for your app.

Overall, UserNotifications framework simplifies the process of managing and delivering notifications in your iOS and macOS applications. It provides a unified interface for handling both local and remote notifications, giving you control over the content, appearance, and user interactions of your notifications.





Architectural Patterns: MVVM

MVVM (Model-View-ViewModel) is a software architectural pattern that separates the presentation logic and user interface (UI) code from the underlying business logic and data. It provides a structured and maintainable way to organize and decouple code in an application. Here's an overview of the key components of MVVM:

- **Model**: The model represents the data and business logic of the application. It encapsulates the data structure, rules, and operations related to the application's domain. Models are typically responsible for retrieving and manipulating data from various sources, such as a database or web service.

- **View**: The view is responsible for displaying the user interface (UI) elements to the user. It represents the visual presentation of the data and provides a way for users to interact with the application. In MVVM, the view is often implemented using platform-specific UI frameworks, such as UIKit for iOS or Android's XML layouts.
- **ViewModel**: The view model acts as an intermediary between the view and the model. It contains the presentation logic and exposes the necessary data and behaviors that the view needs to display and interact with. The view model abstracts the underlying data and business logic, making it easier to test and maintain. It also facilitates data binding between the view and the model.

The interaction between these components follows a unidirectional flow:

- The view observes and reacts to changes in the view model. It typically binds UI elements to properties and commands provided by the view model. When the view needs to display data or perform an action, it triggers the corresponding command or updates the view model's properties.
- The view model retrieves and manipulates the data from the model layer. It exposes the necessary data and behaviors as properties and commands that the view can use. It also notifies the view of any changes in the data through events or bindings.
- The model layer handles the data storage and retrieval, performs business logic operations, and notifies the view model of any changes in the data.

MVVM offers several benefits:

- **Separation of Concerns**: MVVM separates the UI logic (view) from the data and business logic (model and view model). This separation allows for better maintainability, testability, and reusability of code.
- **Testability**: With MVVM, the view model can be easily tested in isolation, as it does not depend on the UI framework. This facilitates unit testing of the application's business logic without the need for complex UI interactions.
- **Data Binding**: MVVM promotes the use of data binding mechanisms, which establish a connection between the view and the view model. Data binding allows changes in the view model to automatically update the view, eliminating the need for manual UI updates.
- **Flexibility**: MVVM supports adaptability to changing UI requirements. The view and view model can be modified or replaced independently, as long as they adhere to the established contract.

Storage: Core Data and FireBase

Storage in apps refers to the mechanisms and techniques used to store and manage data within an application. There are several types of storage options available for apps, depending on the nature of the data and the requirements of the application.

Core Data

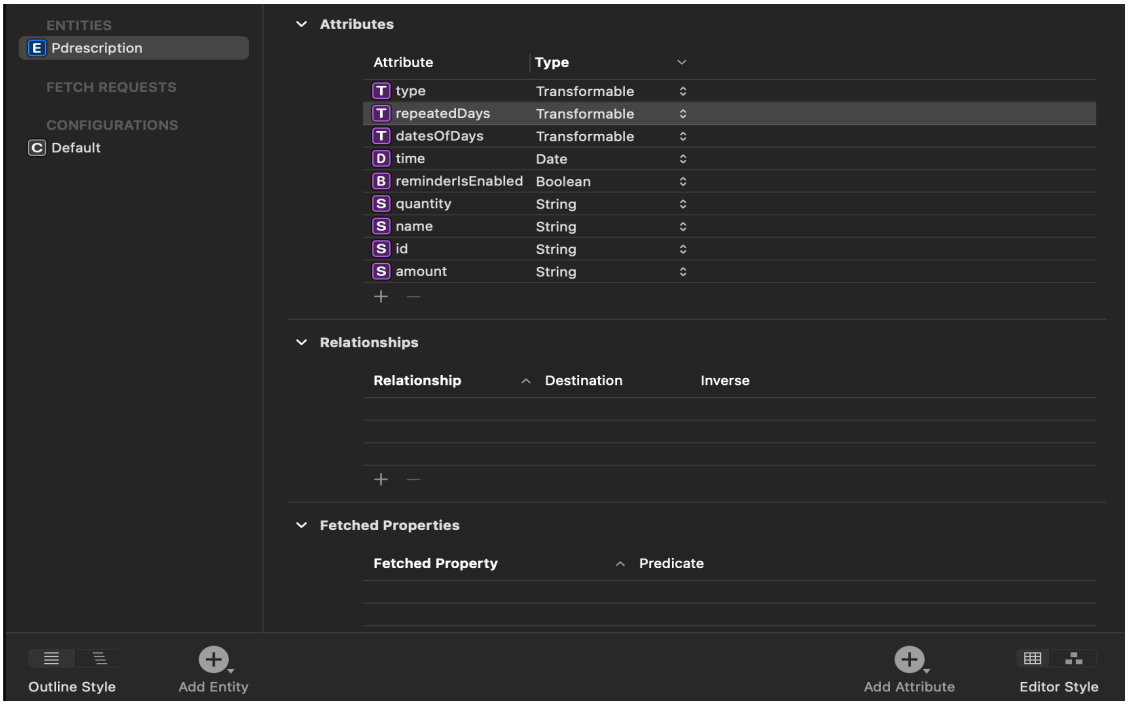
Core Data is a framework provided by Apple that allows developers to manage the storage, retrieval, and manipulation of data in their iOS, macOS, watchOS, and tvOS applications. It provides an object graph management and persistence framework, making it easier to work with complex data models and handle data persistence.
Key features and concepts of Core Data include:

- **Object Graph Management:** Core Data manages object graphs, which represent the relationships and structure of your data. It allows you to define entities, attributes, and relationships between objects in a data model. You can create, update, and delete objects within the object graph.
- **Persistence:** Core Data provides mechanisms for persisting data to disk. It supports various persistent store types, including SQLite, binary, XML, and in-memory stores. You can choose the appropriate store type based on your application's requirements.
- **Managed Object Context:** Core Data uses a managed object context as an interface to the underlying data store. The managed object context is responsible for managing the lifecycle of objects, performing fetch requests, and tracking changes to objects.
- **Entity:** An entity in Core Data represents a real-world object or concept that you want to model. It corresponds to a table in a database and defines the attributes and relationships of the objects that will be stored.
- **Attribute:** An attribute represents a property or piece of information associated with an entity. It can be of different types, such as string, number, date, or boolean.
- **Relationship:** A relationship describes the association between two entities. It can be a one-to-one, one-to-many, or many-to-many relationship. Relationships allow you to establish connections and navigate between related objects.
- **Fetch Requests:** Core Data provides fetch requests to retrieve data from the persistent store based on specified criteria. Fetch requests can be customized with predicates, sorting descriptors, and other parameters.
- **Faulting:** Core Data uses faulting to optimize memory usage. Faulting allows it to represent objects as lightweight placeholders until their attributes or relationships are accessed. When accessed, Core Data automatically loads the necessary data from the persistent store.

- **Migration:** Core Data supports lightweight and custom migration to handle changes in the data model over time. It provides mechanisms to automatically map and migrate data from previous versions of the data model.

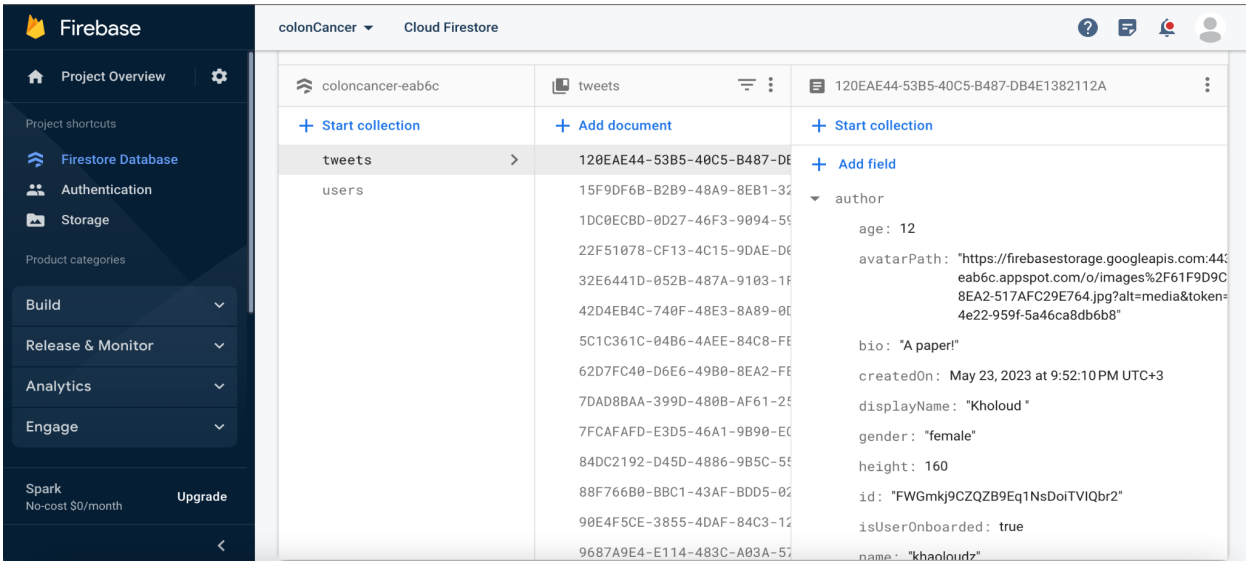
Core Data provides various APIs for interacting with the underlying data store, such as NSManagedObjectContext for managing object contexts, NSManagedObject for working with objects, and NSFetchRequest for fetching data.

Core Data is widely used in iOS and macOS app development for managing application data and providing offline capabilities. It simplifies data management, reduces boilerplate code, and offers powerful features for handling complex data models.



FireBase

Firebase is a comprehensive platform provided by Google for building web and mobile applications. It offers a suite of services and tools that help developers with various aspects of app development, including authentication, real-time database, cloud storage, hosting, cloud functions, and more. Here are some key components and features of Firebase:



- **Firebase Authentication:** Firebase Authentication provides a secure and easy-to-use authentication system for your app. It supports authentication using email/password, phone number, social media logins (such as Google, Facebook, Twitter), and more.

```
class AuthManager{
static let shared = AuthManager()
 func registerUser(with email: String, password: String) -> AnyPublisher<User, Error>{
  return Auth.auth().createUser(withEmail: email, password: password)
   .map(\.user)
   .eraseToAnyPublisher() }}
```

- **Cloud Firestore:** Cloud Firestore is a flexible and scalable NoSQL document database provided by Firebase. It allows you to store, sync, and query data in real-time across multiple clients and platforms. Firestore offers offline support and real-time updates, making it ideal for building collaborative and real-time applications.

```
func collectionTweets(retrieveTweets forUserId: String) -> AnyPublisher<[Tweet], Error> {
            db.collection(tweetsPath).whereField("authorID", isEqualTo: forUserId)
              .getDocuments()
              .tryMap(\.documents)
              .tryMap { snapshots in
               try snapshots.map({
                        try $0.data(as: Tweet.self)    }) }
              .eraseToAnyPublisher() }
```

- **Real-time Database:** Firebase Real-time Database is a cloud-hosted NoSQL database that allows you to store and sync data in real-time. It uses WebSockets to provide real-time updates to connected clients and enables collaborative features like chat, live data synchronization, and real-time notifications.
- **Cloud Storage:** Firebase Cloud Storage provides secure and scalable cloud storage for your app's files, including images, videos, audio, and more. It offers easy integration and supports direct uploads and downloads, access control, and powerful features like image resizing and transformations.
- **Cloud Functions:** Firebase Cloud Functions allows you to run serverless functions in the cloud. It enables you to execute code in response to events from other Firebase services, such as database changes, authentication events, and HTTP requests. Cloud Functions help you automate tasks, perform backend processing, and integrate with third-party services.
- **Firebase Hosting:** Firebase Hosting provides fast and secure hosting for your web app, static content, and dynamic content. It offers content delivery through a global CDN (Content Delivery Network), automatic SSL (Secure Sockets Layer) certificates, and easy deployment with Firebase CLI.
- **Analytics and Performance Monitoring:** Firebase includes powerful analytics and performance monitoring tools. You can gain insights into user behavior, track app usage, and measure the performance of your app. Firebase Analytics provides detailed reports, while Performance Monitoring helps you identify performance bottlenecks.

Firebase offers SDKs for various platforms, including iOS, Android, and web, making it easy to integrate Firebase services into your app. It provides a unified console for managing your app's configuration, monitoring usage, and accessing analytics.
Overall, Firebase is a powerful and feature-rich platform that provides developers with a wide range of tools and services to build, scale, and manage web and mobile applications. offers robust features to enhance user experience.

Chapter 6
Backend : flask

What is flask ?

Flask is a popular open-source web framework for building web applications in Python. It is a lightweight and flexible framework that provides developers with the tools they need to build scalable and secure web applications in a short amount of time.

Flask is a micro-framework, which means that it does not come with all the bells and whistles of a full-fledged web framework like Django. Instead, Flask provides a simple and minimalist approach to building web applications, which makes it a great choice for small to medium-sized projects.

One of the key features of Flask is its simplicity. Flask provides a simple API that makes it easy to build web applications without getting bogged down in complex syntax and boilerplate code. Flask also provides a number of extensions and plugins that can be used to add functionality to the framework, such as support for database integration, authentication, and authorization.

Another advantage of Flask is its flexibility. Flask allows developers to choose the components they need for their web application, rather than being forced to use a predefined set of tools. This makes it easier to build applications that are tailored to the specific needs of the project.

Flask is also known for its excellent documentation and community support. The Flask documentation is comprehensive and easy to navigate, and there are many resources available online to help developers get started with the framework. Additionally, the Flask community is active and supportive, with many developers available to answer questions and provide guidance.

In summary, Flask is a lightweight and flexible web framework for building web applications in Python. Its simplicity, flexibility, and excellent documentation make it a popular choice for developers who want to build web applications quickly and efficiently.

Connecting flask with frontend

Connecting Flask with an iOS frontend is an excellent way to build a robust and scalable application that can be accessed from anywhere at any time. Flask is a popular Python web framework that provides a flexible and lightweight approach to

building web applications. With Flask, developers can quickly create RESTful APIs that can communicate with iOS frontend applications.

To connect Flask with iOS frontend, developers can use various third-party libraries such as Flask-RESTful, Flask-API, and Flask-JSON. These libraries provide an easy way to create RESTful APIs that can handle HTTP requests and responses and communicate with the frontend application.

The first step in connecting Flask with an iOS frontend is to create a RESTful API using Flask. This can be done by defining routes and functions that handle HTTP requests and responses. For example, a route can be defined to handle a GET request for retrieving data from the backend, while a POST request can be used to add new data to the backend.

Once the RESTful API is created, developers can use JSON to communicate between the Flask backend and the iOS frontend. JSON is a lightweight data interchange format that is easy to use and parse. The Flask-JSON library can be used to simplify the process of converting data to JSON and back.

Finally, the iOS frontend can be developed using any programming language such as Swift, Objective-C, or React Native. The frontend application can send HTTP requests to the Flask backend and receive JSON responses. The JSON responses can then be parsed and displayed in the frontend application.

In conclusion, connecting Flask with an iOS frontend is a powerful way to build scalable and robust applications. With Flask, developers can quickly create RESTful APIs that can communicate with iOS frontend applications. By using JSON to communicate between the backend and the frontend, developers can simplify the process of exchanging data. Overall, Flask provides an excellent platform for building modern web applications that can be accessed from anywhere at any time.

```
    app = Flask(__name__)

with open('lgb.pkl', 'rb') as f:
    model = pickle.load(f)

print("+"*50, "Model is loaded")

@app.route('/')
def Home():
    return render_template("index.html")

@app.route('/prediction', methods = ["POST"])
```

```
def prediction():
    img = request.files['img']
    img.save("img.jpg")
    image = cv2.imread("img.jpg")

    image = cv2.resize(image, (64,64))
    y =image.reshape(1,image.shape[0],image.shape[1],3)
    y = y.reshape(y.shape[0], -1)
    pred = model.predict(y)



    pred=int(pred)
    return render_template("prediction.html", data=pred)

if __name__ == "__main__":
    app.run(debug=True)
```

Chapter 7
Conclusion

In conclusion, Colon Care, a colon cancer classification and detection app, has the potential to be a valuable tool in the fight against colon cancer. By providing users with a personalized risk assessment, promoting screening, and empowering patients to take control of their health, Colon Care could help increase awareness, encourage early detection, and improve outcomes for patients with colon cancer.

The app could also offer support and resources to those who have been diagnosed with colon cancer, including information on treatment options and support groups. With its user-friendly and accessible interface, Colon Care could help reduce the mortality rate associated with colon cancer by making it easier for people to identify their risk factors and seek medical attention if necessary.

Overall, Colon Care has the potential to be a powerful tool in the prevention, detection, and treatment of colon cancer. Further research and development are needed to ensure the accuracy and effectiveness of the app, but with the right support and resources, Colon Care could make a significant impact on the health of individuals and populations around the world.

References

1. Allison J.E. Colorectal cancer screening guidelines: The importance of evidence and transparency. *Gastroenterology.* 2010.

2. An F.P., Liu J.E. Medical Image Segmentation Algorithm Based on Optimized Convolutional Neural Network-Adaptive Dropout Depth Calculation. *Complexity.* 2020.

3. Araghi M., Soerjomataram I., Jenkins M., Brierley J., Morris E., Bray F., Arnold M. Global trends in colorectal cancer mortality: Projections to the year 2035. *Int. J. Cancer.* 2019.

4. Rathore S., Hussain M., Ali A., Khan A. A recent survey on colon cancer detection techniques. *IEEE/ACM Trans. Comput. Biol. Bioinform.* 2013.

5. Kitayama J., Nagawa H., Tsuno N., Osada T., Hatano K., Sunami E., Saito H., Muto T. Laminin mediates tethering and spreading of colon cancer cells in physiological shear flow. *Br. J. Cancer.* 1999.

6. Bychkov D., Linder N., Turkki R., Nordling S., Kovanen P.E., Verrill C., Walliander M., Lundin M., Haglund C., Lundin J. Deep learning based tissue analysis predicts outcome in colorectal cancer. *Sci. Rep.* 2018.

7. Hur C., Chung D.C., Schoen R.E., Gazelle G.S. The management of small polyps found by virtual colonoscopy: Results of a decision analysis. *Clin. Gastroenterol. Hepatol.* 2007.

8. Gunduz-Demir C., Kandemir M., Tosun A.B., Sokmensuer C. Automatic segmentation of colon glands using object-graphs. *Med. Image Anal.* 2010.

9. Masud M., Sikder N., Nahid A.A., Bairagi A.K., AlZain M.A. A machine learning approach to diagnosing lung and colon cancer using a deep learning-based classification framework. *Sensors.* 2021.

10. Bénard F, Barkun AN, Martel M, von Renteln D. Systematic review of colorectal cancer screening guidelines for average-risk adults: Summarizing the current global recommendations. World J Gastroenterol. 2018

11. World Health Organization, Cancer, (2022).

12. Schreuders EH, Ruco A, Rabeneck L, Schoen RE, Sung JJY, Young GP, Kuipers EJ. Colorectal cancer screening: a global overview of existing programmes. Gut. 2015.

13. Araghi M, Soerjomataram I, Bardot A, Ferlay J, Cabasag CJ, Morrison DS, De P, Tervonen H, Walsh PM, Bucher O. Changes in colorectal cancer incidence in seven high-income countries: a population-based study, Lancet. Gastroenterol Hepatol. 2019.

14. Guren MG. The global challenge of colorectal cancer, Lancet. Gastroenterol Hepatol. 2019.

15. Henderson RH, French D, Maughan T, Adams R, Allemani C, Minicozzi P, Coleman MP, McFerran E, Sullivan R, Lawler M. The economic burden of colorectal cancer across Europe: a population-based cost-of-illness study, Lancet. Gastroenterol Hepatol. 2021.

16. Hossain MJ, Chowdhury UN, Islam MB, Uddin S, Ahmed MB, Quinn JMW, Moni MA. Machine learning and network-based models to identify genetic risk factors to the progression and survival of colorectal cancer. Comput Biol Med. 2021.

17. Keum N, Giovannucci E. Global burden of colorectal cancer: emerging trends, risk factors and prevention strategies. Nat Rev Gastroenterol Hepatol. 2019.

18. Kourou, K., Exarchos, T. P., Exarchos, K. P., Karamouzis, M. V. & Fotiadis, D. I. Machine learning applications in cancer prognosis and prediction. *Comput. Struct. Biotechnol. J.* 13, 8–17 (2015).

19. Uddin, S., Khan, A., Hossain, M. E. & Moni, M. A. Comparing different supervised machine learning algorithms for disease prediction. *BMC Med. Inform. Decis. Mak.* 19.

20. Sung, H. *et al.* Global cancer statistics 2020: Globocan estimates of incidence and mortality worldwide for 36 cancers in 185 countries. *CA* 71, 209–249 (2021)

21. Jiang, J. *et al.* Predictive model for the 5-year survival status of osteosarcoma patients based on the seer database and xgboost algorithm. *Sci. Rep.* 11, 5542 (2021).

22. Huang, S., Yang, J., Fong, S. & Zhao, Q. Artificial intelligence in cancer diagnosis and prognosis: Opportunities and challenges. *Cancer Lett.* 471, 61–71 (2020).

23. Lee, C. *et al.* Application of a novel machine learning framework for predicting non-metastatic prostate cancer-specific mortality in men using the surveillance, epidemiology, and end results (seer) database. *Lancet Digit. Heal.* 3, e158–e165 (2021).

24. Silva, G. *et al.* Machine learning for longitudinal mortality risk prediction in patients with malignant neoplasm in são paulo, brazil. *Artif. Intell. Life Sci.* 3, 100061 (2023).