Unit - 4

Q1. MVC

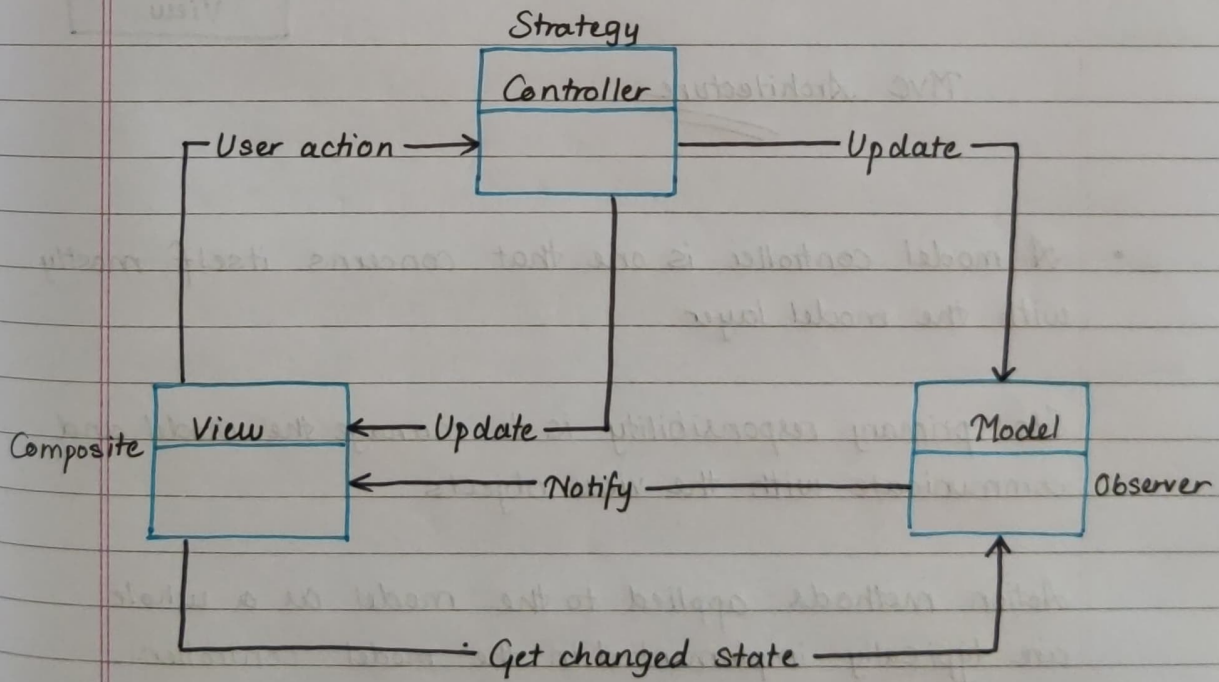
- A Controller acts as the intermediary between the application's view objects and it's model objects
- Controllers are often in charge of making sure that views have access to the model objects they need to display
- Controllers act as a conduit through which views learn about the changes to the model.
- Model-View-Controller (MVC) is a design pattern that is composed of several more basic design patterns.
- MVC is made up of Composite, Strategy and Observer patterns

Composite → The view objects in an application are actually a composite of nested views that work together in a coordinated manner. (ie. the view hierarchy)
These display components range from a window, to compound views such as a table, to individual views such as buttons

Strategy → A controller object implements the strategy for one or more view objects
The view object confines itself to maintaining the visual aspects, and it delegates all decisions to the controller

Observer → A model object keeps interested objects in an application (usually view objects) informed of changes in it's state.

- A controller object recieves the event and applies a strategy
This strategy can be to request a model object to change it's state or to request a view object to change it's behaviour or appearance.
The model object in turn notifies all objects registered as observers when it's state changes.
If the observer is a view object, it may update it's appearance accordingly

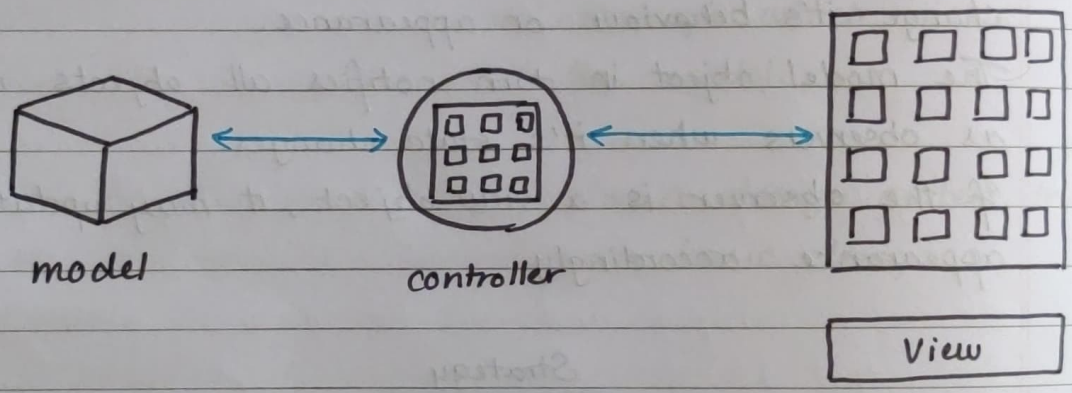


MVC Design Patterns

- The user needs to interact with an app interface in the simplest way possible

- Interface must be designed with the user in mind, and make it efficient, clear and straightforward

- As applications become more complex, we will create interfaces with more scenes and more views



MVC Architecture

- A model controller is one that concerns itself mostly with the model layer

Its primary responsibility is to manage the model and communicate with the view objects.

Action methods applied to the model as a whole are typically implemented in the model controller.

eg. NSDocumentController object automatically handles action methods related to saving files.

- A view controller is a controller that concerns itself mostly with the view layer

  It's primary responsibility is to manage the interface and communicate with the model objects.

  Action methods ~~implemen~~ concerned with the data displayed in the view are typically implemented in a view controller.

  eg. NSWindowController object is an example of view controller.

- A coordinating controller is typically an NSWindowController, or NSDocumentController or an instance of a custom subclass of NSObject.

  It's role is to oversee or coordinate the functioning of the entire application or part of an application.

  The coordinating controller provides services such as →

→ Responding to delegation messages and observing notifications

→ Responding to action messages

→ Managing the lifetime of owned objects (eg. releasing them at proper time)

→ Establishing connections between objects and performing other set-up tasks.

Q2. iOS App life cycle

- Apps are a sophisticated interplay between our custom code and the system framework
- system framework → system framework provides the basic infrastructure that all apps need to run.
- custom code → we provide the code required to customize that infrastructure and give the app the look and feel that we want.
- The system moves our app from state to state in response to actions happening throughout the system.
- For example, when the user presses the home button, a phone call comes in.
- Any such interruption could occur, and the state of the currently running app changes in response to it.
- The different states in which an app can be in are listed as follows →

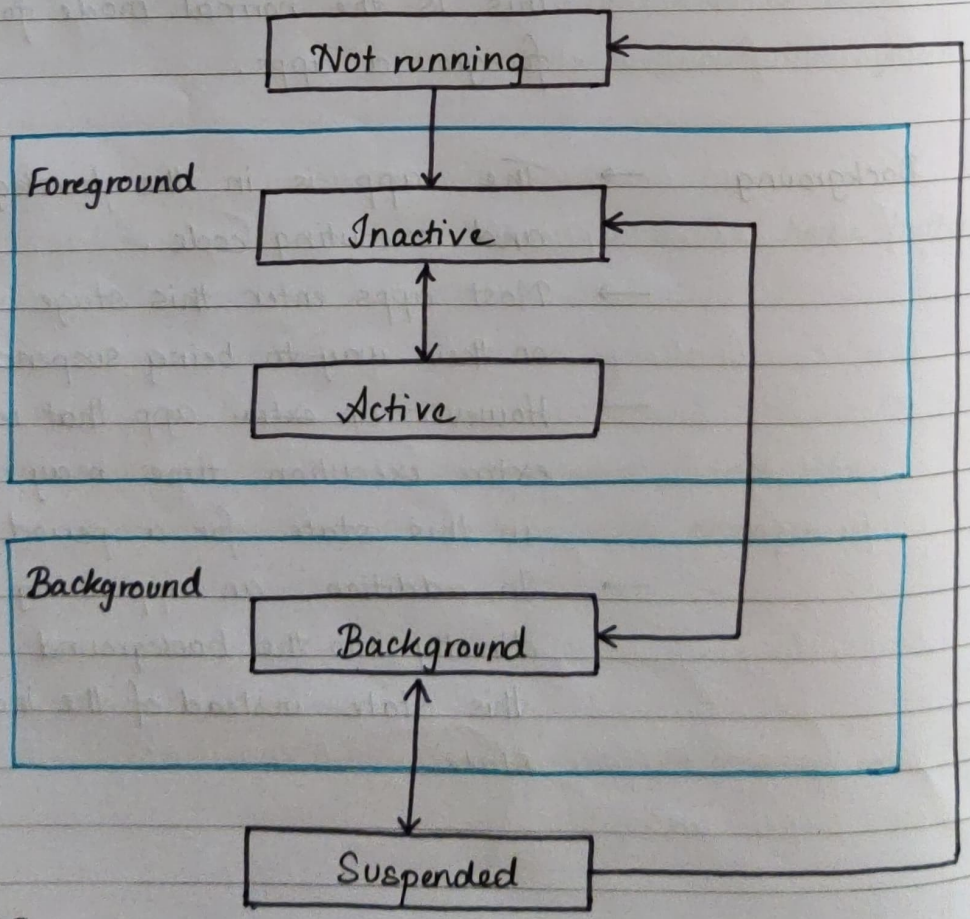
State	Description
1. Not running	→ The app has not been launched, or it was running but was terminated by the system.
2. Inactive	→ The app is running in the foreground, but is currently not recieving events (It may be executing other code though) → The app usually stays in this state only breifly as it transitions to another state
3. Active	→ The app is running in the foreground and is recieving events. → This is the normal mode for foreground apps.
4. Backgroung	→ The app is in the background and executing code. → Most apps enter this stage briefly on their way to being suspended. → However an ~~extra~~ app that requests extra execution time may remain in this state for a period of time. → In addition, an app being launched directly into the background enters this state instead of the inactive state.

5. Suspended → The app is in the background but is not executing code

→ The system moves apps to this state and does not notify them before doing so

→ While suspended, an app remains in memory, but does not execute any code.

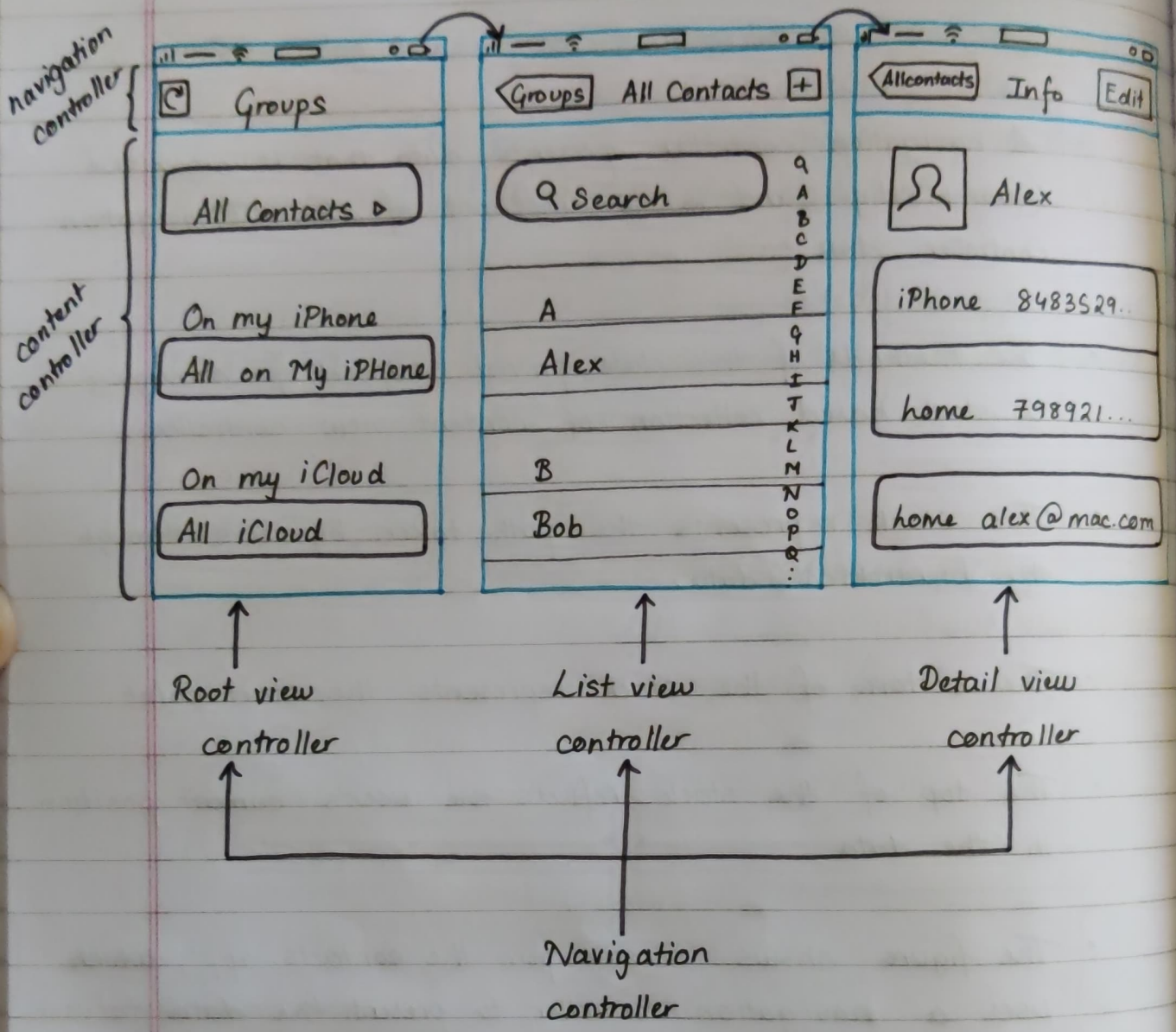
→ When a low memory condition occurs, the system may purge suspended apps without notice to make more space for the foreground apps.



APP LIFECYCLE

Q3. Navigation controllers and story boards

- A navigation controller presents data that is organized hierarchically, and is an instance of the UINavigation-Controller class
- The methods of this class provide support for managing a stack based collection of content view controllers.
- This stack represents the path taken by users through the hierarchical data.
- The bottom of the stack represents the start point
- The top of the stack reflects the user's current position in the data
- The figure shows screens from the contacts app, which uses a navigation controller to present the data to user.
- The navigation bar at the top of each page is owned by the navigation controller
- The rest of each screen displayed to the user is managed by a content view controller that presents the information at a specific level of the data hierarchy
- As the user interacts with the controls in the interface, those controls tell the navigation controller to display the next view controller in the sequence, or dismiss the current view controller.



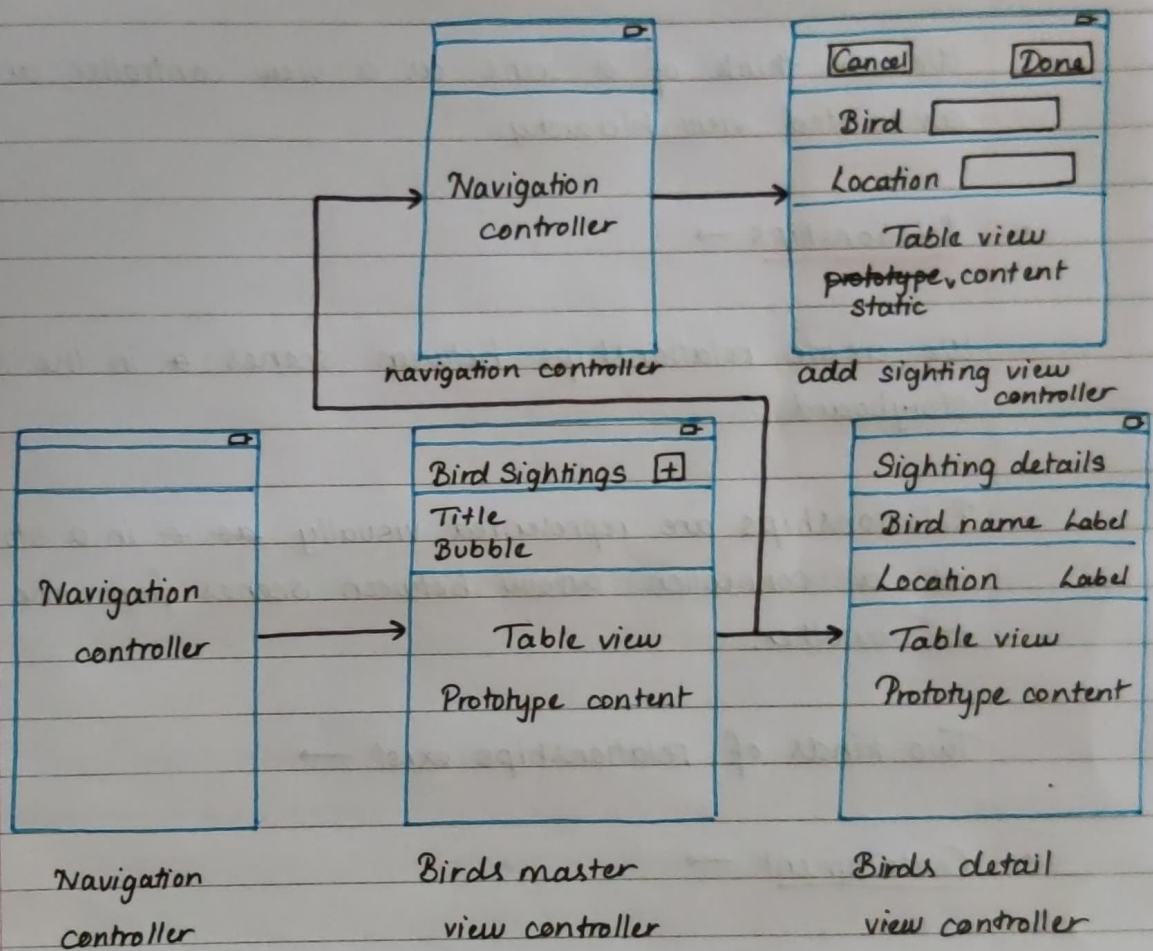
Story board →

- A story board is the visual representation of the appearance and flow of our application

- When we implement our app using story boards, we use interface builder to organize our app's view controllers and any associated views.

- Interface Builder allows us to understand the flow through the app at a glance

- The resulting story board is stored as a file in the project



Story Board.

- The above example shows a story board.

- Often iOS can instantiate the view controllers in our storyboard at the ~~time~~ moment when they are needed.

- Similarly view hierarchy associated with each controller is automatically loaded when it needs to be displayed.

- Scene →

A scene represents an onscreen content area that is managed by a view controller.

We can think of a scene as a view controller and it's associated view hierarchy.

- Relationships →

We create relationships between scenes ~~o~~ in the same storyboard

Relationships are represented visually ~~as~~ ~~o~~ in a storyboard as a connection arrow between scenes from one scene to another.

Two kinds of relationships exist →

- Containment →

It represents a parent-child relationship between two scenes

An advantage of using containment relationships in a story board is that the Interface Builder can adjust the appearance of the child view controller to reflect the presence of it's ancestors.

View controllers present in other view controllers are instantiated when the parent controller is instantiated.

For example, the first connection from the navigation controller to another scene represents defines the first view controller pushed on to the navigation stack

This controller is automatically instantiated when the navigation controller is instantiated.

- Segue →

A segue represents a visual transition from one scene to another. At runtime, a segue can be triggered by various actions

When a segue is triggered, it causes a new view controller to be instantiated and transitioned on screen

Types of segues →

- a push segue pushes the destination view controller onto the navigation controller's stack
- a model segue presents the destination view controller
- a popover segue displays the destination view controller in a popover
- a custom segue allows us to design our own transition to display the destination view controller.

Q6. Databases

SQLite is at the heart of Android's database support

It is simple, small, light weight RDBMS implementation with simple API

SQLite is so dominant in the mobile world due to →

- Low memory consumption
- Ease of use
- Free availability

It is open source, and ACID compliant (atomicity, consistency, isolation, durability)

It uses SQL query language

Datatypes in SQLite →

- NULL – null value
- INTEGER – any no. which is not a floating point no.
- REAL – a floating point no.
- TEXT – text string or single characters
- BLOB – (Binary large object) The value is a blob of data, stored exactly as it was in the input.

methods in SQLiteDatabase class →

- openOrCreateDatabase() →

This method will open an existing database or create one.

```
SQLiteDatabase myDatabase;
myDatabase = openOrCreateDatabase("my_sqlite_
                database.db", Context.MODE_PRIVATE,
                SQLiteDatabase.CREATE_IF_NECESSARY, null);
```

- **Create Table →**

```
String createTable = "CREATE TABLE demo(id
                INTEGER, PRIMARY KEY AUTOINCREMENT,
                firstName TEXT, lastName TEXT);
myDatabase.execSQL(createTable);
```

- **Insert Records →**

```
ContentValues values = new ContentValues();
values.put("firstName", "First Name");
values.put("lastName", "Last Name");
long newAuthorID = myDatabase.insert("demo", " ", values);
```

- **Update Records →**

```
ContentValues values = new ContentValues();
values.put("firstName", "New First Name");
myDataBase.update("demo", values, "id=?",
                new String[] {demoId.toString()});
```

- Record Deletion →

```
String [] whereArgs = {"20", "30"};
recAffected = myDatabase.delete("demo", "recID > ? and
              recID < ?", whereArgs);
```

SQLite Classes →

- SQLiteCursor – Cursor implementation that exposes the results from a query on a SQLiteDatabase
- SQLiteDatabase – exposes methods to manage an SQLite database. ~~These~~ It has methods for
  - creating
  - opening
  - closing
  - Inserting
  - Updating
  - deleting
  - querying
- SQLiteOpenHelper – A helper class to manage database creation
- SQLiteProgram – A base class for compiled SQLite programs.
- SQLiteQuery – A SQLite program that represents a query that reads the resulting rows into a CursorWindow.

- SQLiteQueryBuilder - A convenience class that helps build SQLQueries

- SQLiteStatement - A pre-compiled statement against a SQLiteDatabase that can be reused.

Queries using SQLite →

- Raw queries take for input a syntactically correct SQL select statement.
  The select query can be as complex as needed and involve any number of tables

- Simple queries are compact parameterized select statements that operate on a single table

  simple queries use a template implicitly representing a condensed version of a typical SQL ~~sta~~ select statement.

  Simple queries can only retrieve data from a single table.

A method's signature has a fixed sequence of seven arguements representing :

1. the table name

2. the columns to be retrieved

3. the search condition (where - clause)

4. arguements for the where - clause

5. the group - by clause

6. having - clause

7. the order - by clause