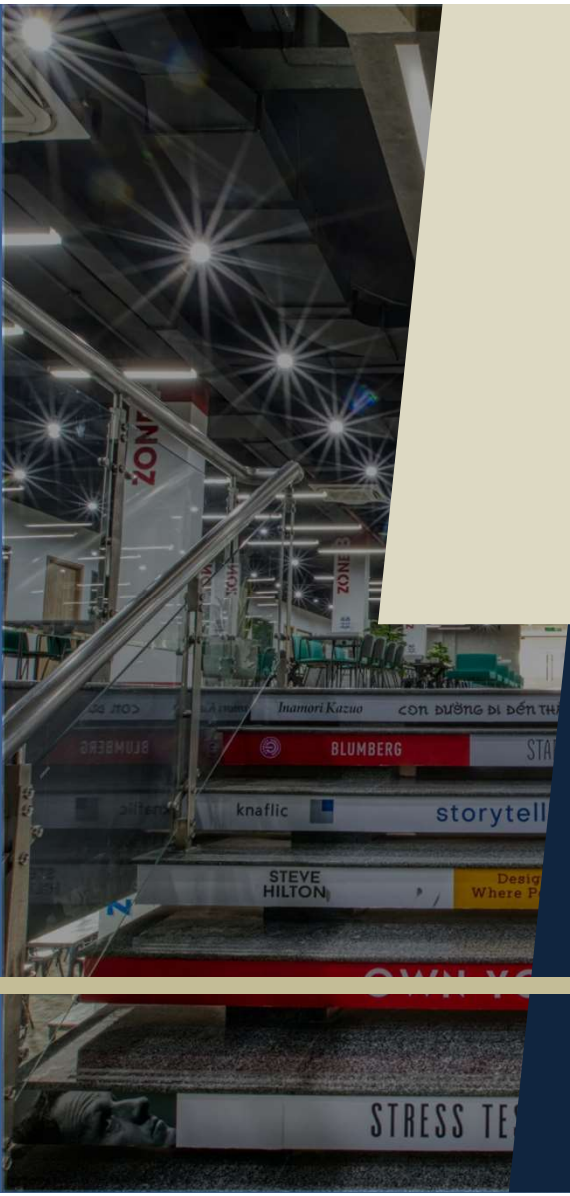



EIU
TRƯỜNG ĐẠI HỌC
QUỐC TẾ
MIỀN ĐÔNG
EASTERN
INTERNATIONAL
UNIVERSITY

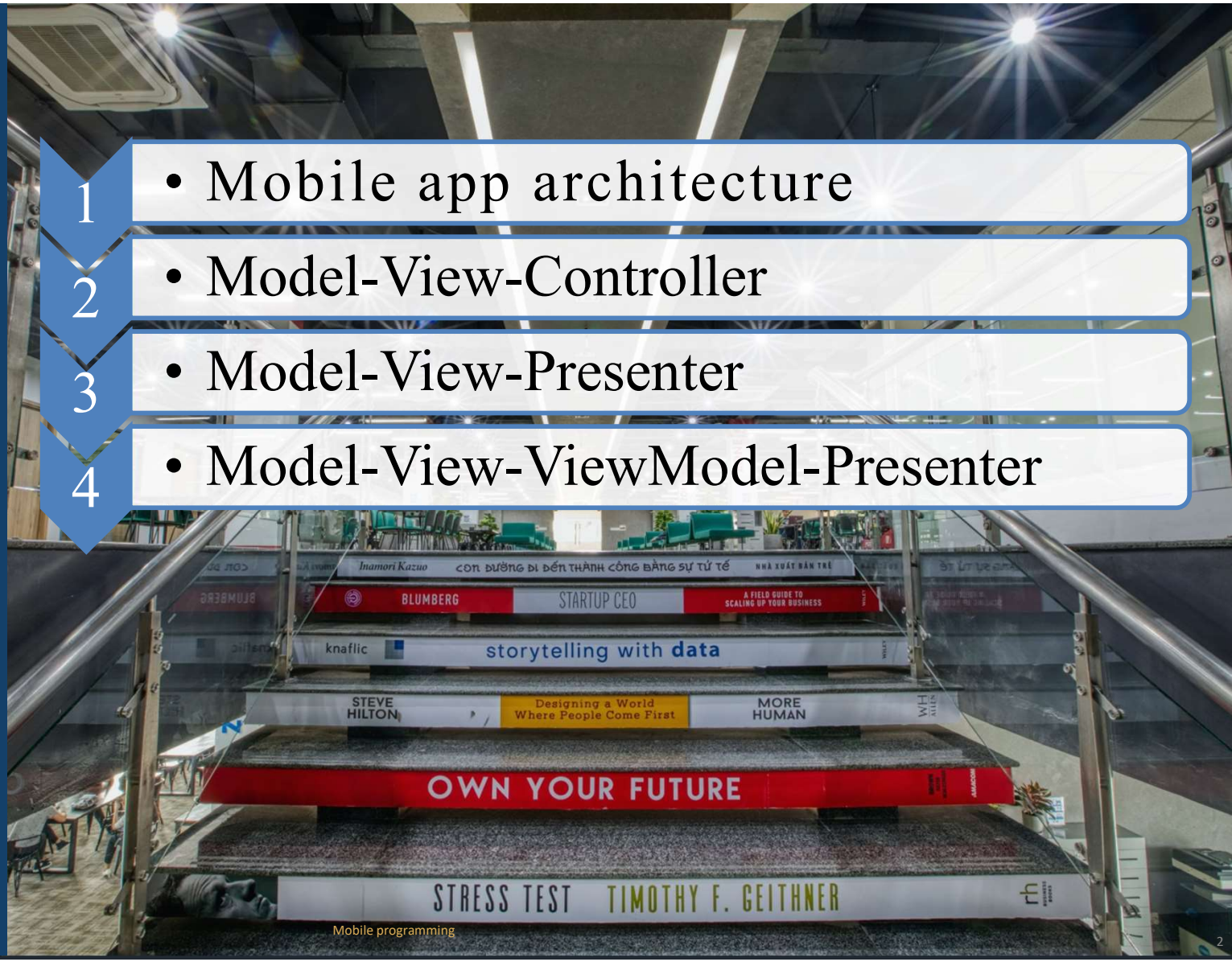
# Mobile Programming

## Chapter 2: MOBILE APP ARCHITECTURE

EIU

# CONTENT

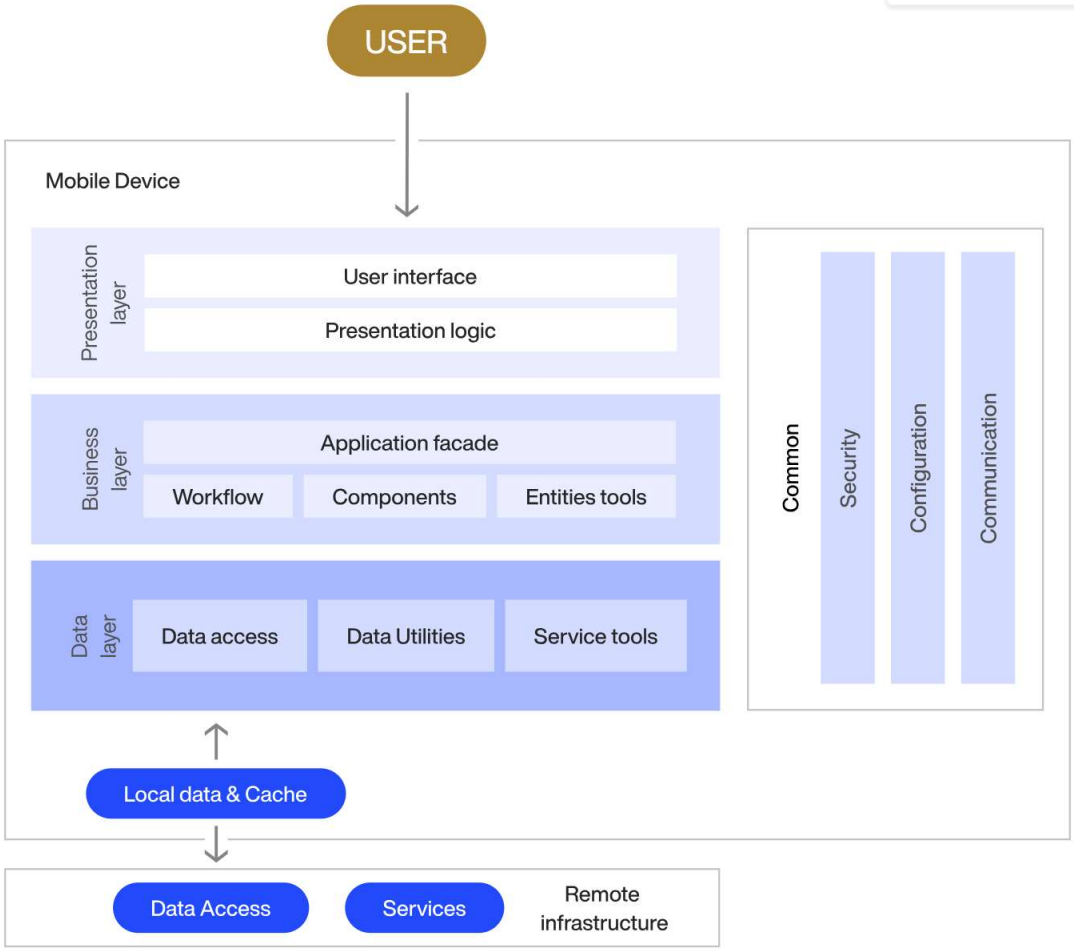
1. Mobile app architecture
2. Model-View-Controller
3. Model-View-Presenter
4. Model-View-ViewModel-Presenter



# I. Mobile app architecture

In app development, architecture refers to an app's rules, processes, and internal structure—essentially, how it's built.

It determines how the different components communicate with each other to process app inputs and deliver output for the user.



# I. Mobile app architecture

**Why is the architecture of a mobile app important?**

- Simply put, properly designed architecture makes the mobile app much more stable, efficient, and easier to work with.

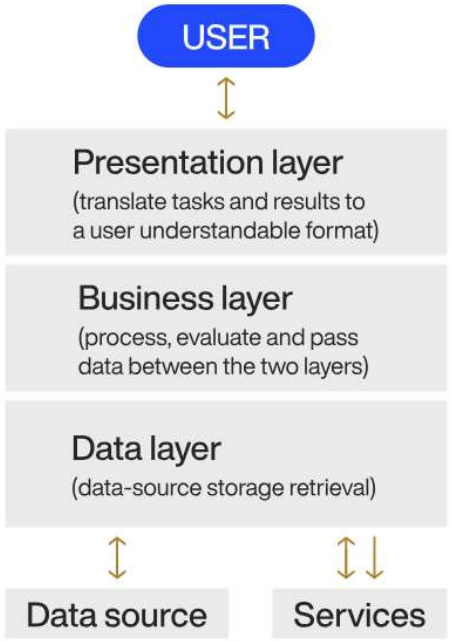
- A good architecture is important, otherwise it becomes slower and more expensive to add new capabilities in the future.

- Good architecture also allows for reusability. This allows you to use a single code in multiple projects, saving you tremendous development time and effort.

# I. Mobile app architecture

## Fundamental layers in mobile app architecture

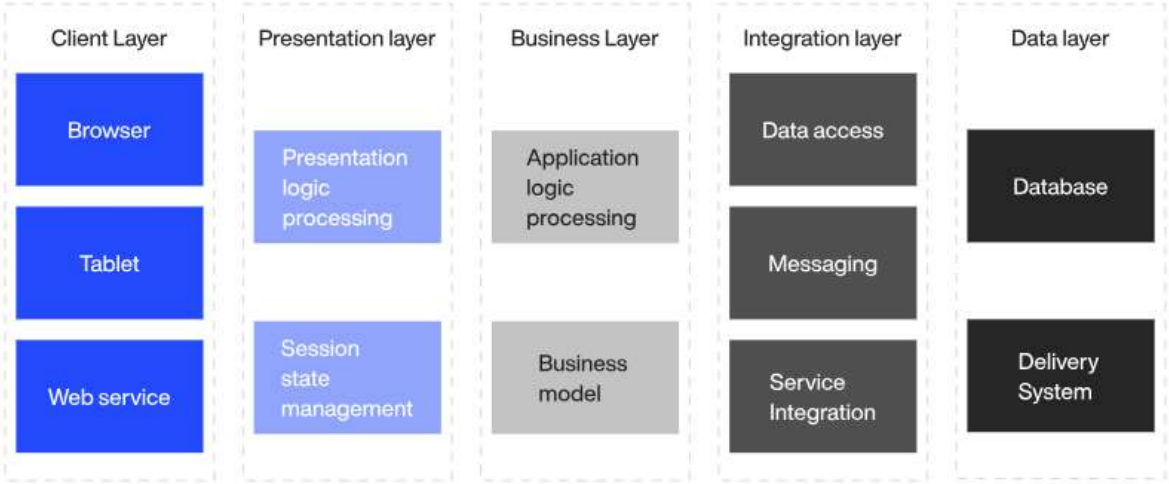
Most types of app architecture are composed of three layers—the data layer, the business layer, and the presentation layer.



# I. Mobile app architecture

## Types of mobile app architecture: Layered architecture

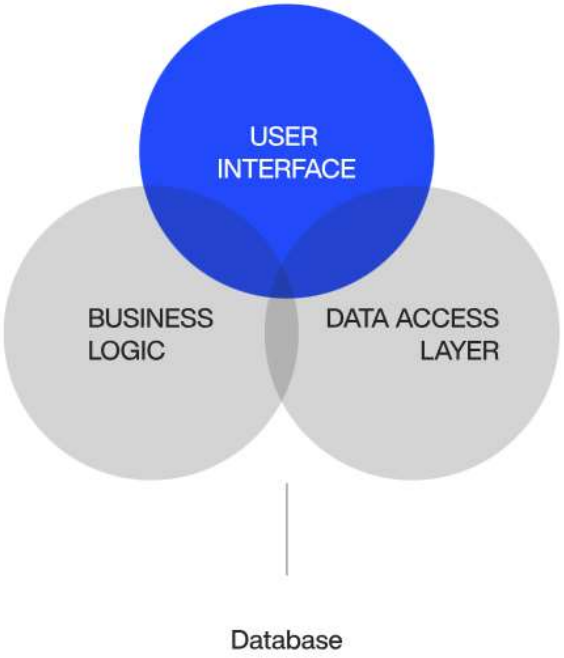
Layered or N-tier architecture is where components with similar logic or purpose are grouped, then organized into several layers or tiers.



# I. Mobile app architecture

## Types of mobile app architecture: Monolithic architecture

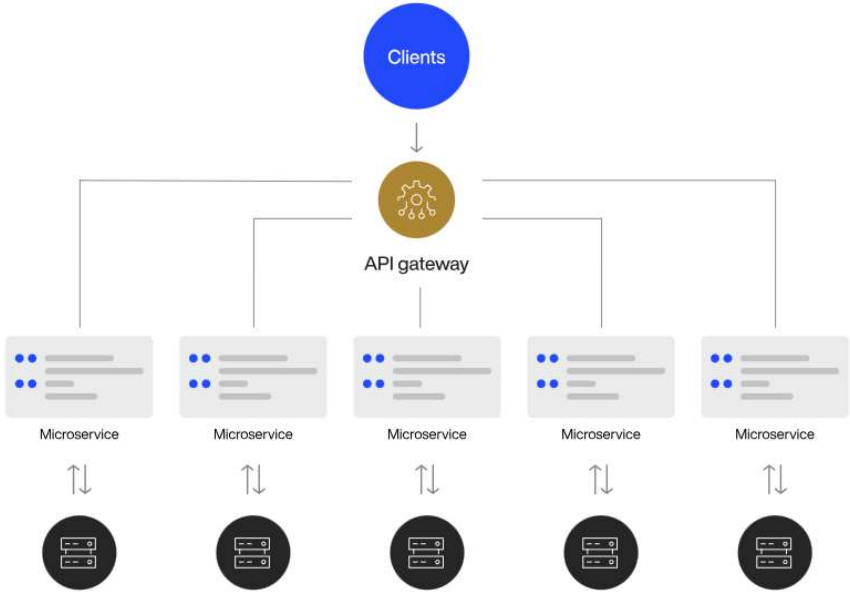
In the monolithic type of architecture, all the app components are tightly integrated, and function as a single unit called a monolith. It's an approach that you'll often see in older systems.



# I. Mobile app architecture

**Types of mobile app architecture: Microservices architecture**

- The microservices type of architecture divides the app into smaller components called a microservice.
- Each microservice is completely standalone and can function without needing any other component.
- The main client app then uses an application programming interface (API) to communicate with them, as shown below:



# I. Mobile app architecture

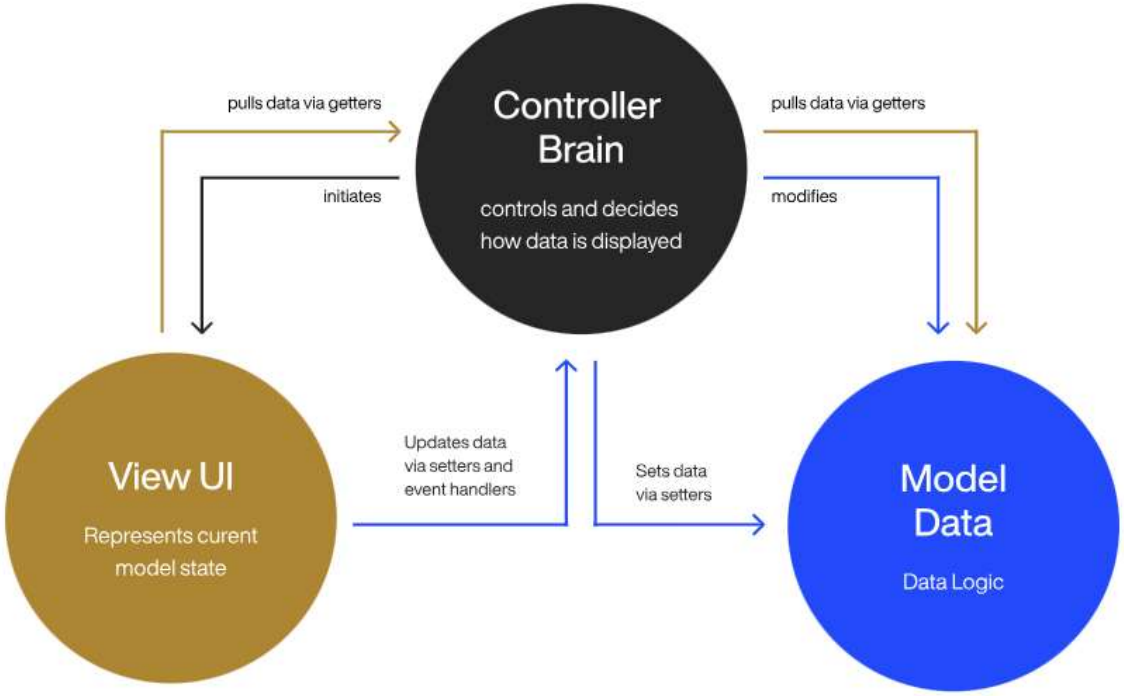
**Types of mobile app architecture patterns**

- Model-View-Controller: MVC
- Model-View-Presenter: MVP
- Model-View-ViewModel: MVVM

# II. Model-View-Controller: MVC

**Types of mobile app architecture patterns: Model-View-Controller: MVC**

The Model-View-Controller (MVC) is the simplest architecture pattern



# II. Model-View-Controller: MVC

**Types of mobile app architecture patterns: Model-View-Controller: MVC**

- The Model component is responsible for data handling, similar to the data layer. It contains the code for getting data from sources like a database, a microservice via an API, or a JSON object.

- The Controller component processes the data from the model component and sends it to the view component, essentially acting as the link between them.

- The View component is responsible for what the user sees—or the UI.

# II. Model-View-Controller: MVC

## Types of mobile app architecture patterns: Model-View-Controller: MVC

```
// models/ProductModel.js
const products = [
    { id: 1, name: 'Product 1', quantity: 10, price: 20.99 },
    { id: 2, name: 'Product 2', quantity: 5, price: 15.99 },
    // ... other products
  ];

  export const getProducts = () => products;
```

```
// components/ProductList.js
import React from 'react';
import { View, Text, FlatList } from 'react-native';
import { getProducts } from '../models/ProductModel';

const ProductList = () => {
  const products = getProducts();

  return (
    <View>
      <Text>Product List</Text>
      <FlatList
        data={products}
        keyExtractor={(item) => item.id.toString()}
        renderItem={({ item }) => (
          <View>
            <Text>{item.name}</Text>
            <Text>Quantity: {item.quantity}</Text>
            <Text>Price: ${item.price}</Text>
          </View>
        )}
      />
    </View>
  );
};
export default ProductList;
```

# II. Model-View-Controller: MVC

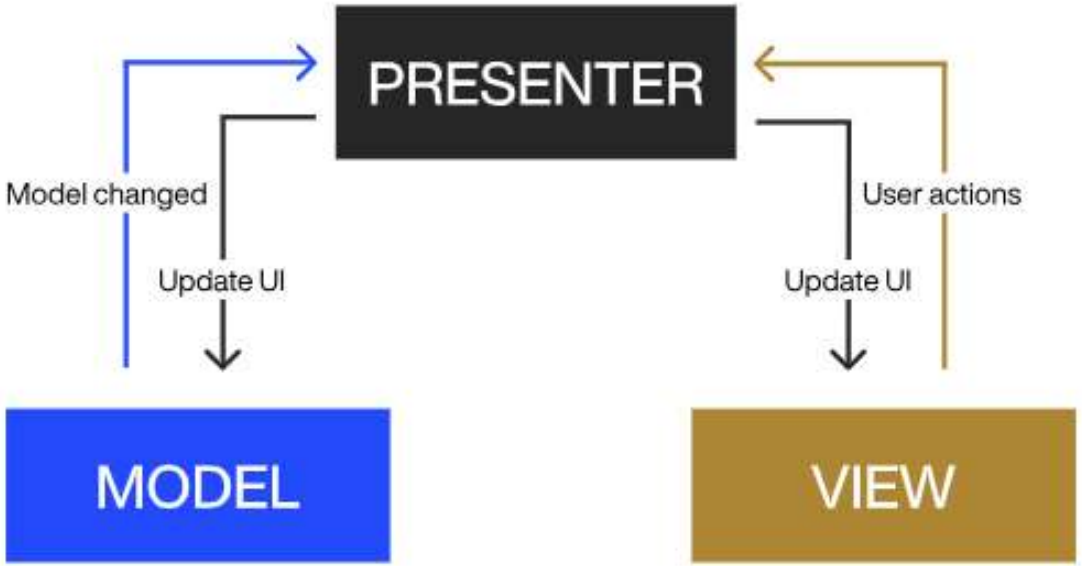
## Types of mobile app architecture patterns: Model-View-Controller: MVC

```jsx
// components/ProductController.js
import React from 'react';
import ProductList from './ProductList';

const ProductController = () => {
  // You can add additional logic here if needed

  return <ProductList />;
};

export default ProductController;
```

```jsx
// App.js
import React from 'react';
import { SafeAreaView } from 'react-native';
import ProductController from './components/ProductController';

const App = () => {
  return (
    <SafeAreaView>
      <ProductController />
    </SafeAreaView>
  );
};

export default App;
```

# III. Model-View-Presenter (MVP)

**Types of mobile app architecture patterns: The Model-View-Presenter (MVP)**
The Model-View-Presenter (MVP) architecture is another common pattern used extensively in Android development.



# III. Model-View-Presenter (MVP)

**Types of mobile app architecture patterns: The Model-View-Presenter (MVP)**

- The MVP architecture has many similarities with MVC. Here, the Model component also handles the data, and the View component is for the UI.

- The Presenter component is analogous to the Controller component in the MVC model, containing the logic that processes data for the user.

- Views are reusable in the MVP model, which makes it more modular and reusable than an MVC application.

# III. Model-View-Presenter (MVP)

## Types of mobile app architecture patterns: The Model-View-Presenter (MVP)

```
// models/ProductModel.js
const products = [
    { id: 1, name: 'Product 1', quantity: 10, price: 20.99 },
    { id: 2, name: 'Product 2', quantity: 5, price: 15.99 },
    // ... other products
  ];

  export const getProducts = () => products;
```

```
// views/ProductView.js
import React from 'react';
import { View, Text, FlatList } from 'react-native';

const ProductView = ({ products }) => {
  return (
    <View>
      <Text>Product List</Text>
      <FlatList
        data={products}
        keyExtractor={(item) => item.id.toString()}
        renderItem={({ item }) => (
          <View>
            <Text>{item.name}</Text>
            <Text>Quantity: {item.quantity}</Text>
            <Text>Price: ${item.price}</Text>
          </View>
        )}
      />
    </View>
  );
};
export default ProductView;
```

# III. Model-View-Presenter (MVP)

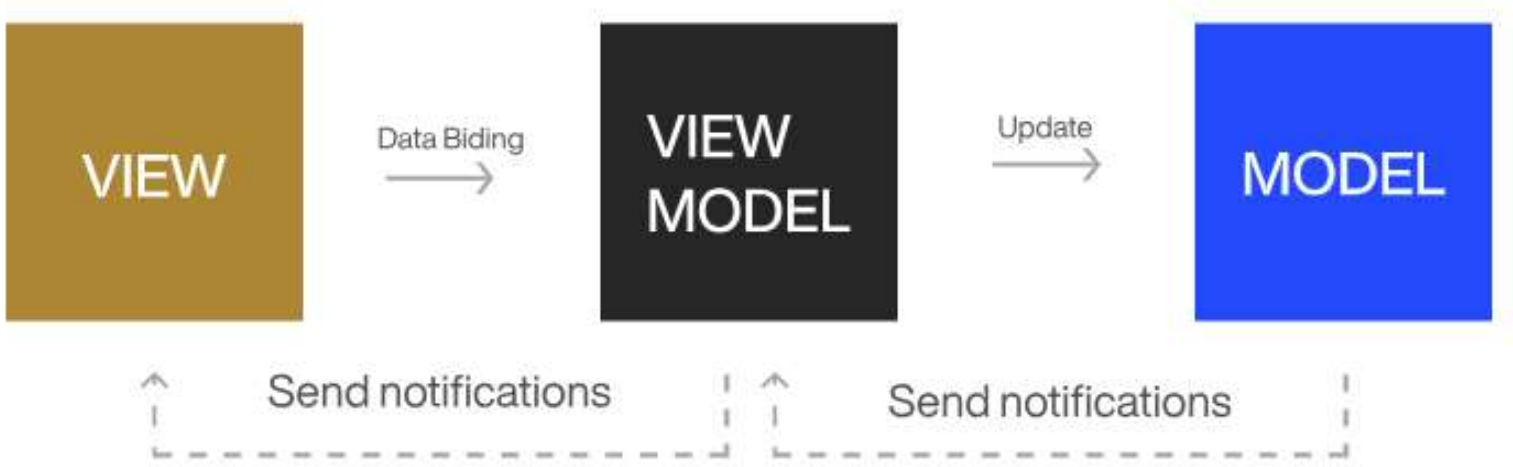
## Types of mobile app architecture patterns: The Model-View-Presenter (MVP)

```
// presenters/ProductPresenter.js
import React, { useState, useEffect } from 'react';
import ProductView from '../views/ProductView';
import { getProducts } from '../models/ProductModel';

const ProductPresenter = () => {
  const [products, setProducts] = useState([]);

  useEffect(() => {
    // Fetch data from the Model
    const fetchData = async () => {
      const productData = getProducts();
      setProducts(productData);
    };

    fetchData();
  }, []);

  return <ProductView products={products} />;
};

export default ProductPresenter;
```

```
// App.js
import React from 'react';
import { SafeAreaView } from 'react-native';
import ProductPresenter from './presenters/ProductPresenter';

const App = () => {
  return (
    <SafeAreaView>
      <ProductPresenter />
    </SafeAreaView>
  );
};

export default App;
```

# IV. Model-View-ViewModel (MVVM)

**Types of mobile app architecture patterns: Model-View-ViewModel (MVVM)**
The Model-View-ViewModel (MVVM) architecture separates the code logic as follows:



# IV. Model-View-ViewModel (MVVM)

**Types of mobile app architecture patterns: Model-View-ViewModel (MVVM)**

- View, as always, houses the visual elements like UI and text. However, unlike the MVC and MVP models, the View component can't change the UI elements directly.
- The MVVM uses data binding for this purpose. It acts as the bridge between the View and ViewModel components.
- The ViewModel contains the application logic, while the Model handles the data.
- What this means is that the ViewModel can effectively work without having to know the View component.

# IV. Model-View-ViewModel (MVVM)

## Types of mobile app architecture patterns: Model-View-ViewModel (MVVM)

```
// models/ProductModel.js
const products = [
    { id: 1, name: 'Product 1', quantity: 10, price: 20.99 },
    { id: 2, name: 'Product 2', quantity: 5, price: 15.99 },
    // ... other products
  ];

  export const getProducts = () => products;
```

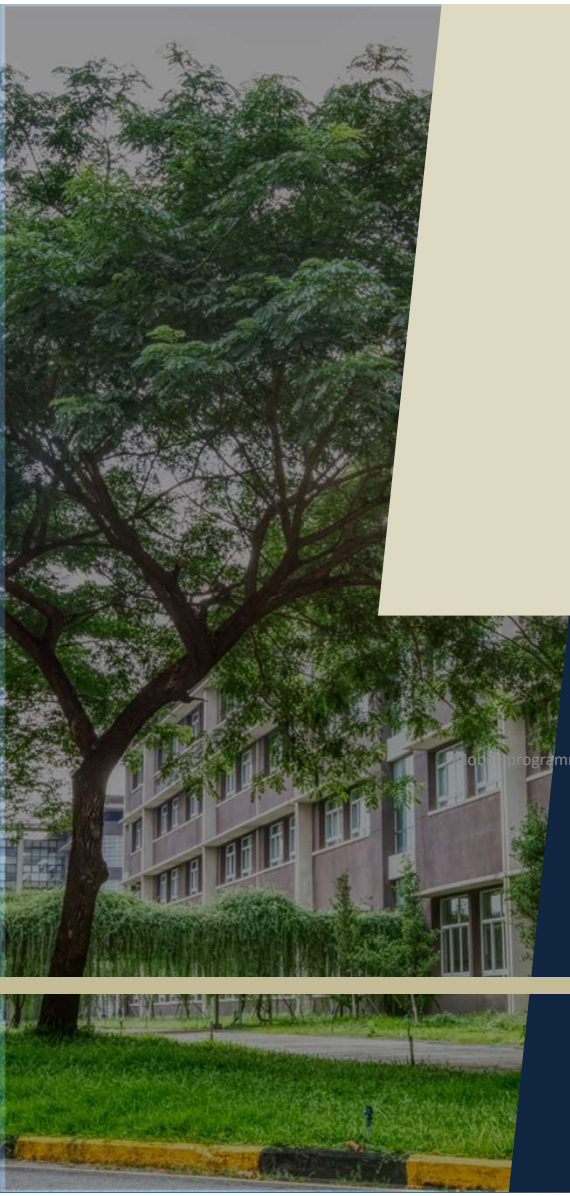
```
// views/ProductView.js
import React from 'react';
import { View, Text, FlatList } from 'react-native';

const ProductView = ({ products }) => {
  return (
    <View>
      <Text>Product List</Text>
      <FlatList
        data={products}
        keyExtractor={(item) => item.id.toString()}
        renderItem={({ item }) => (
          <View>
            <Text>{item.name}</Text>
            <Text>Quantity: {item.quantity}</Text>
            <Text>Price: ${item.price}</Text>
          </View>
        )}
      />
    </View>
  );
};

export default ProductView;
```

# IV. Model-View-ViewModel (MVVM)

## Types of mobile app architecture patterns: Model-View-ViewModel (MVVM)

```
// viewmodels/ProductViewModel.js
import { useState, useEffect } from 'react';
import { getProducts } from '../models/ProductModel';

const ProductViewModel = () => {
  const [products, setProducts] = useState([]);

  useEffect(() => {
    // Fetch data from the Model
    const fetchData = async () => {
      const productData = getProducts();
      setProducts(productData);
    };

    fetchData();
  }, []);

  return { products };
};

export default ProductViewModel;
```

```
// presenters/ProductPresenter.js
import React from 'react';
import ProductViewModel from '../viewmodels/ProductViewModel';
import ProductView from '../views/ProductView';

const ProductPresenter = () => {
  const viewModel = ProductViewModel();

  return <ProductView products={viewModel.products} />;
};

export default ProductPresenter;
// App.js
import React from 'react';
import { SafeAreaView } from 'react-native';
import ProductPresenter from './presenters/ProductPresenter';

const App = () => {
  return (
    <SafeAreaView>
      <ProductPresenter />
    </SafeAreaView>
  );
};

export default App;
```



EIU
TRƯỜNG ĐẠI HỌC
QUỐC TẾ
MIỀN ĐÔNG
EASTERN
INTERNATIONAL
UNIVERSITY

# Q&A