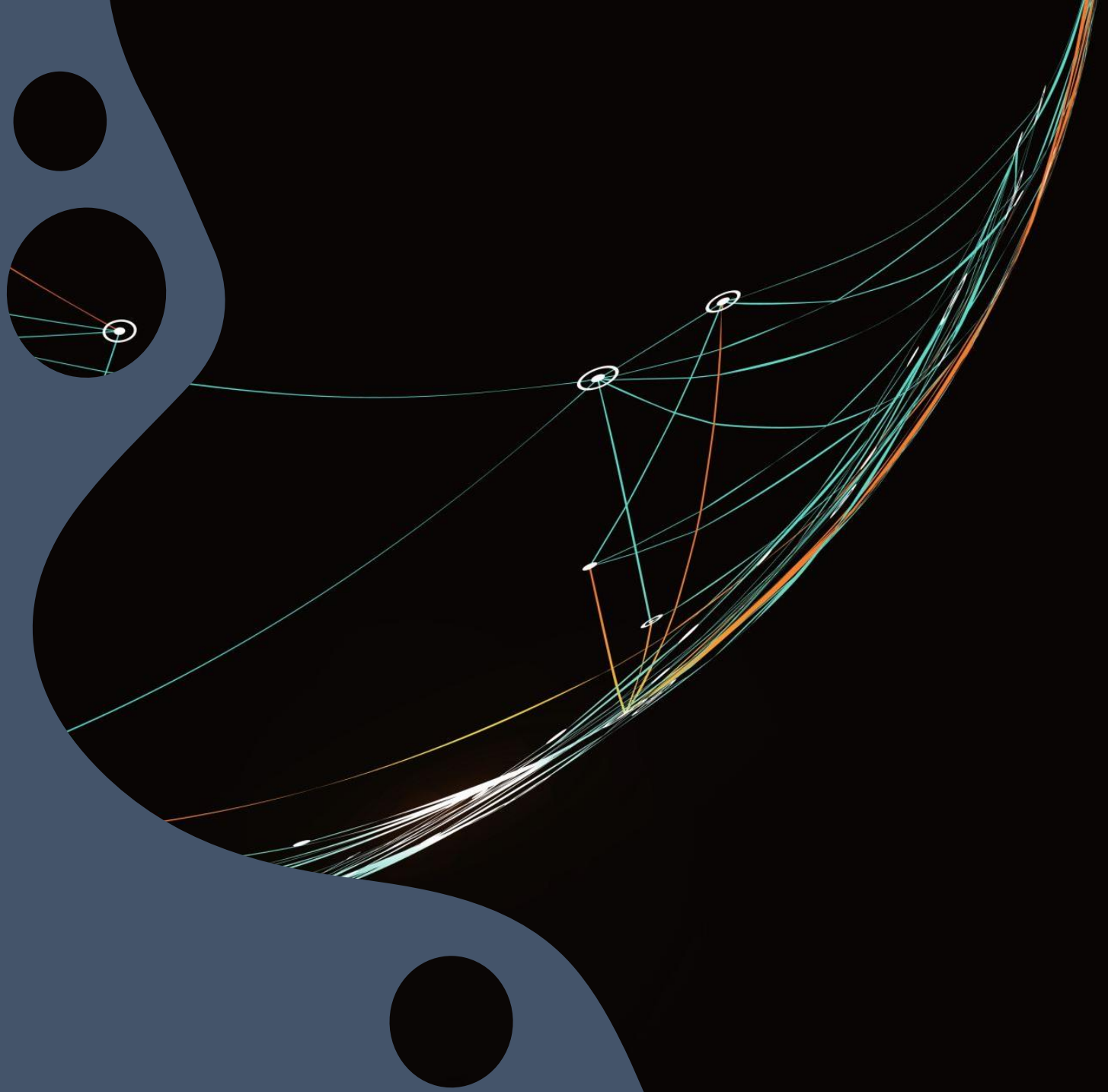
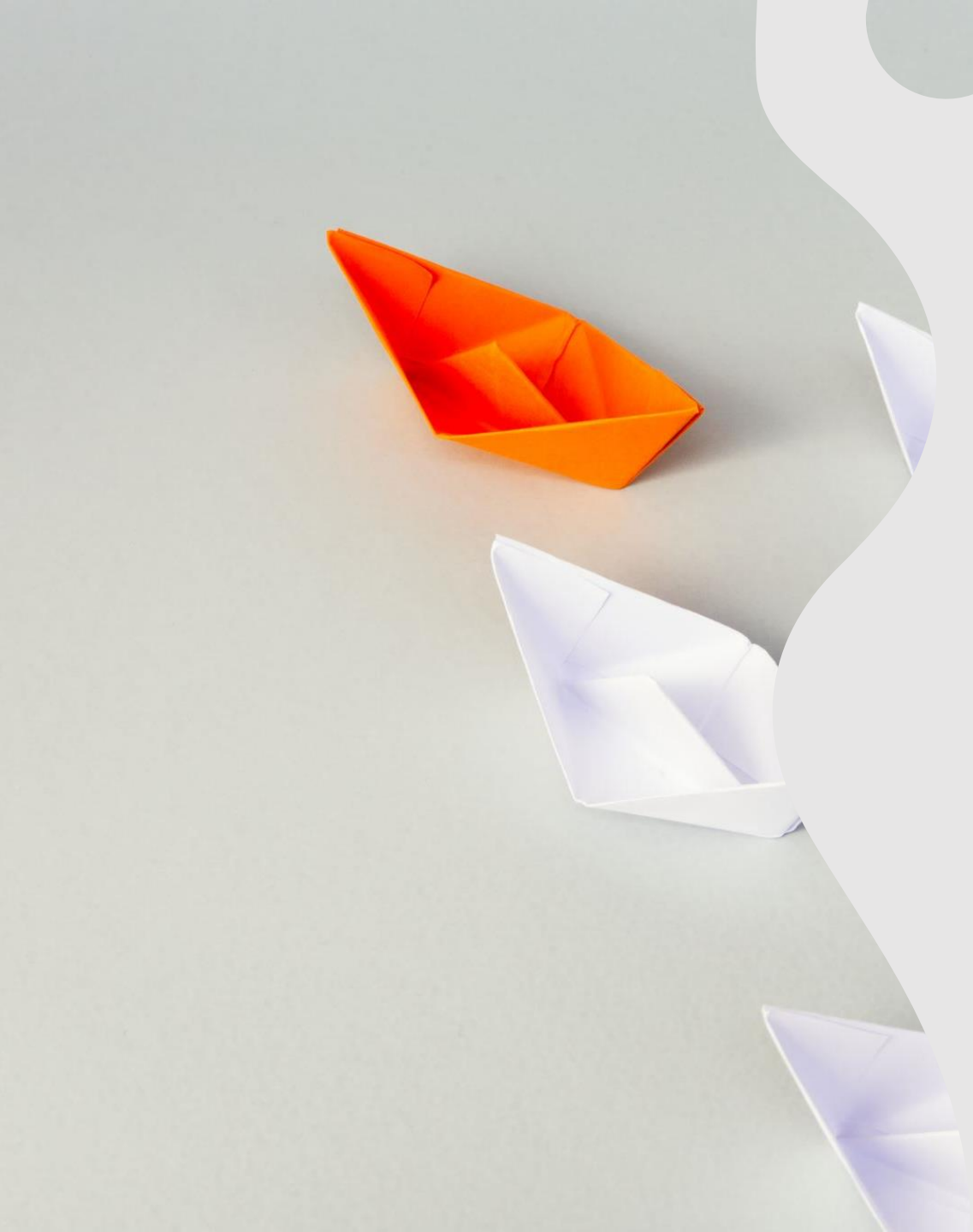
Session 1: Introduction to MVVM, Model and Basics of CRUD Operations

The Journey Begins!





Agenda

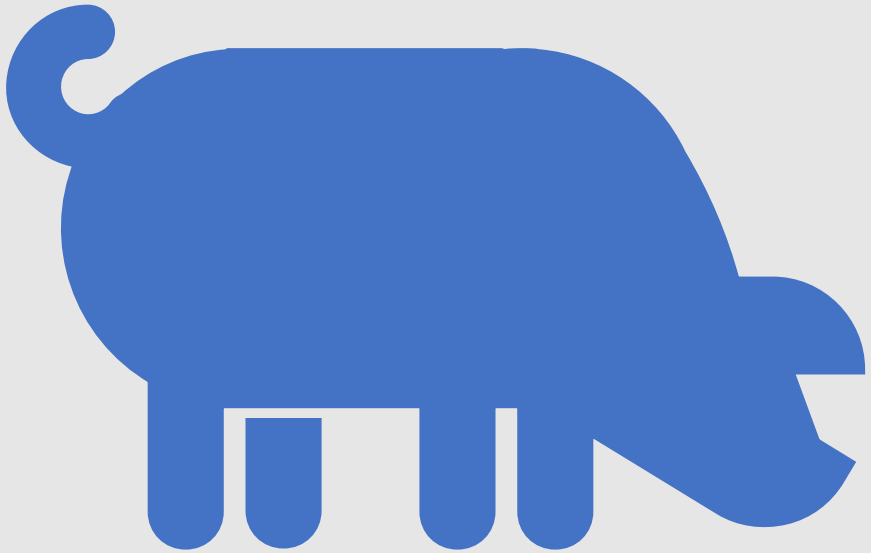
1.'Hog Wild' Scenario Introduction

2.Introduction to MVVM

3.The Role of the Model

The 'Hog Wild' Scenario

- A brief overview of the 'Hog Wild' scenario.



HOG WILD INVOICING SYSTEM

Our client, Sarah, runs a small parts and service business called Hog Wild. Sarah had been using a cash register for years but wanted to upgrade to a point of sale (POS) system. She had heard that POS systems could help her manage her inventory and sales more efficiently.

Sarah started researching different POS systems and found many options available. She spent time researching different features of the POS systems that she would like to have within her system.

Sarah will be starting with minimum functions as a starting point and customize it to fit the needs of her business. She included information about her business, such as the types of products she sold and the number of employees she had. She also included a list of features that she wanted in a POS system:

- **Inventory**
- **Customer**
- **Employees**
- **Invoicing**
- **Purchase Order**
- **Receiving**
- **Reporting**
- **Maintaining reference data (Lookups)**

Sarah wants to start with a small subset of the features and add onto it as she and her staff become comfortable with the system and as enhancements are required. Sarah has decided to start with customers, employees, inventory, parts, invoicing, and maintenance. With a new POS system, Sarah hopes to manage her inventory more efficiently and decide what products to stock better. She feels that the system will help her save time by automating many of the tasks that she used to do manually.

She has decided to contract out the building of the POS system for her business to our team. Using a partner of our company, we have outsourced the creation of the database along with some test data that you, as developers, can code against.

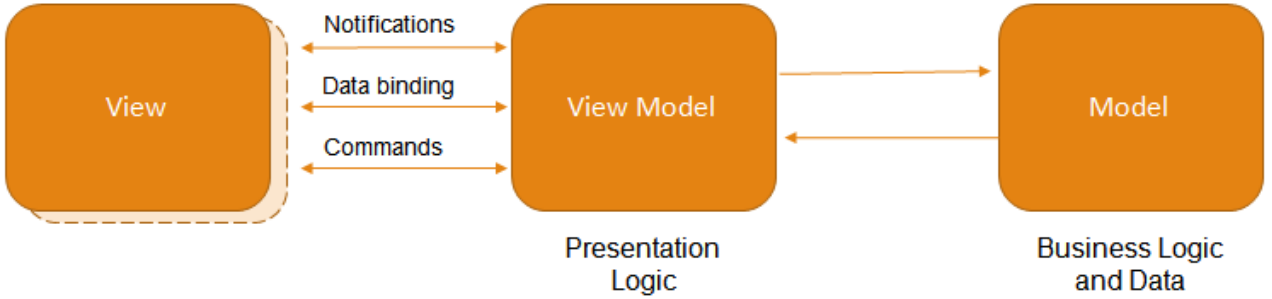

Application Demo

Introduction to MVVM

Basic definition of each component (Model-View-ViewModel)

Introduction to MVVM (Model-View-ViewModel)

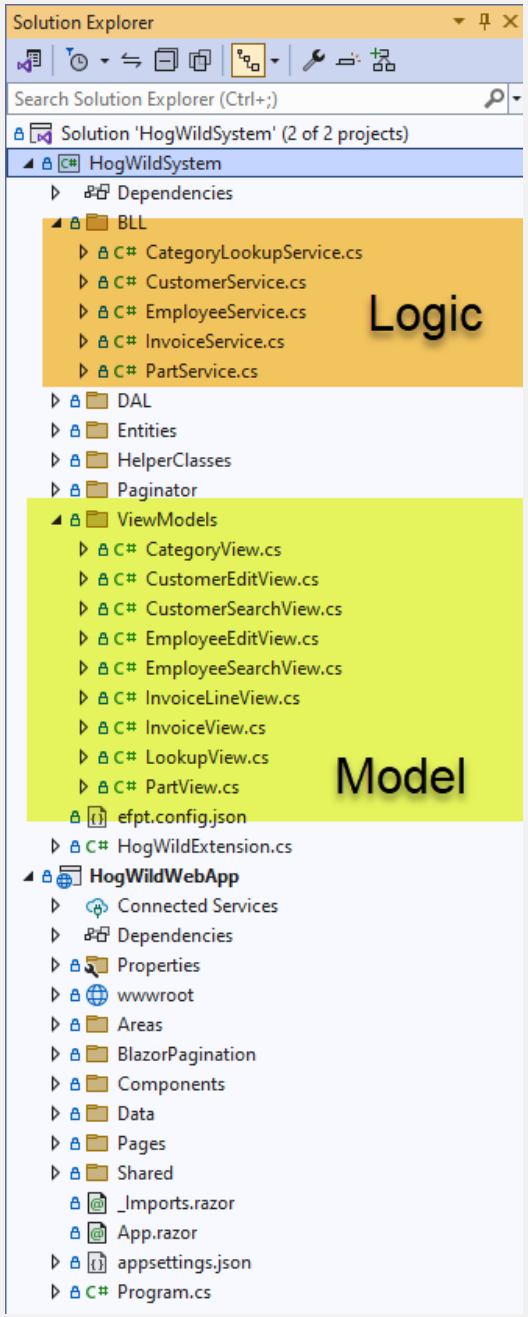
"MVVM is a design pattern used in software engineering that promotes a clear separation between application logic, user interface, and presentation logic."



The Model

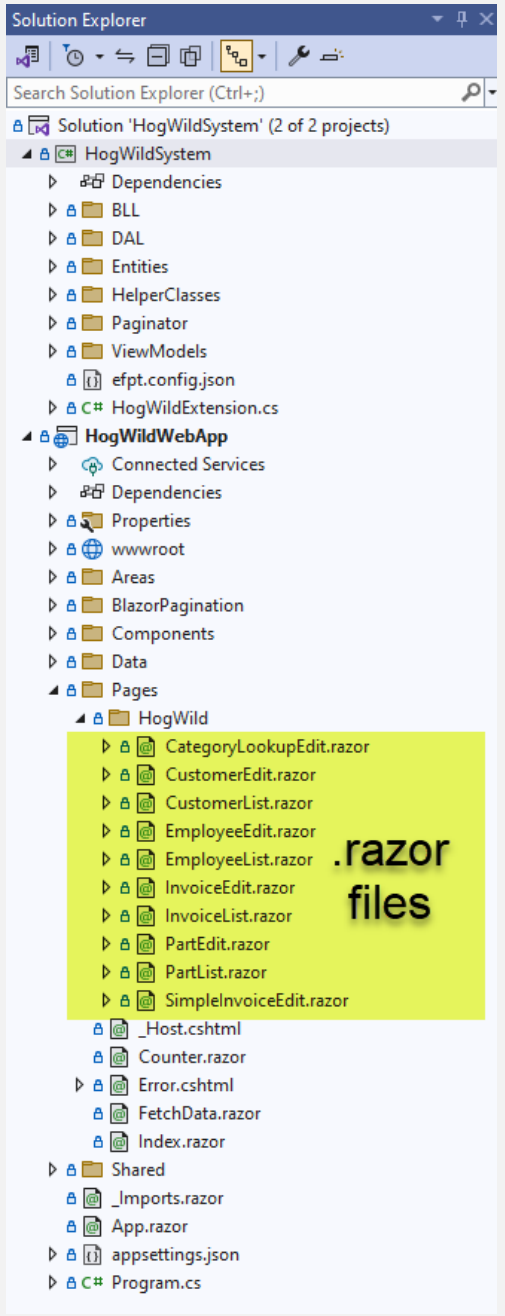
- Definition: The “Model represents the data structures and business rules of the application:
- **Manages data** (e.g., reading/writing from/to a database).
- **Responsible for data validation**.
- **Represents the "core" of the application: its logic and functionality.**

- **ViewModel** -> Model & Model Properties
- **BLL** -> Business Logic and Managing Data



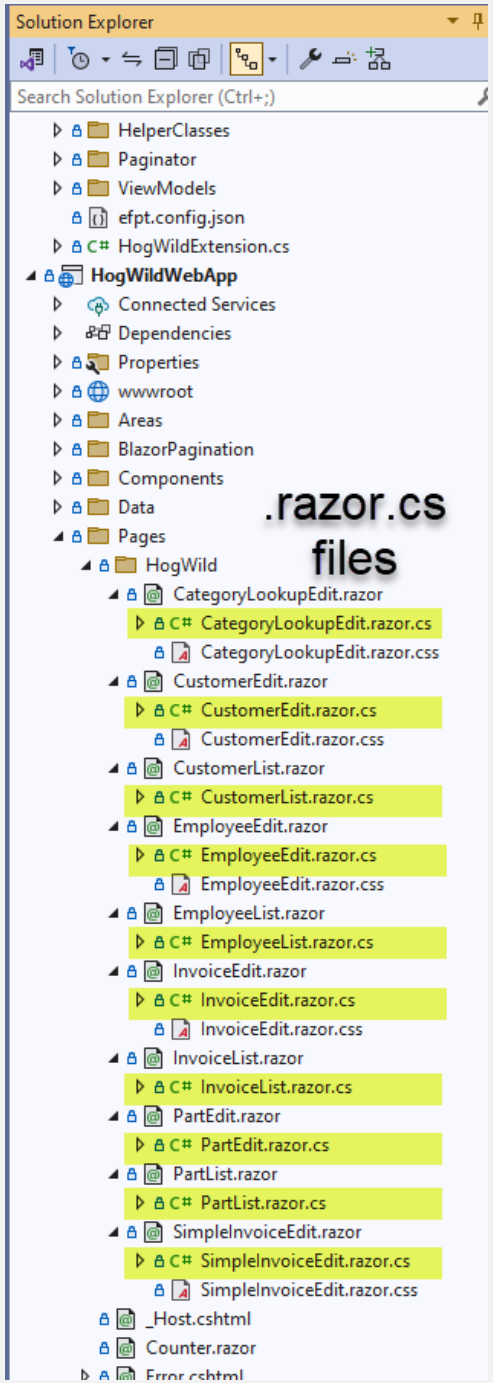
The View

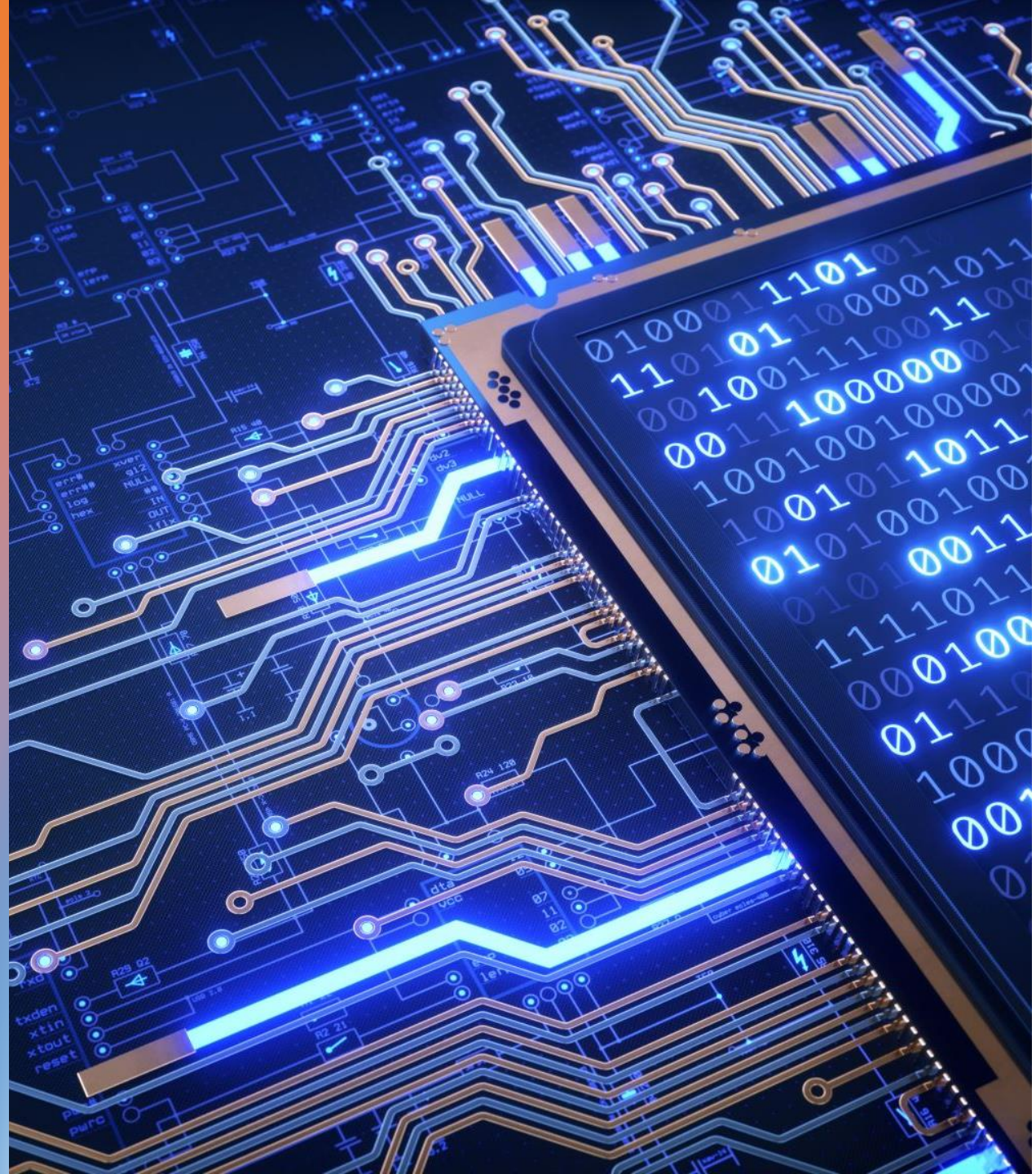
- "The View is the user interface of the application - it's everything the user sees and interacts with.“
- **Displays data** provided by the ViewModel.
- **Visual representation** (e.g., UI elements, tables, forms).
- **Represents layout, design, and aesthetics**.



The ViewModel

- **Transforms data** from the Model to a format suitable for the View.
- **Handles user inputs** and translates them into Model actions.
- **Enhances data** from the Model for display purposes (e.g., formatting, local validation).





Why MVVM?

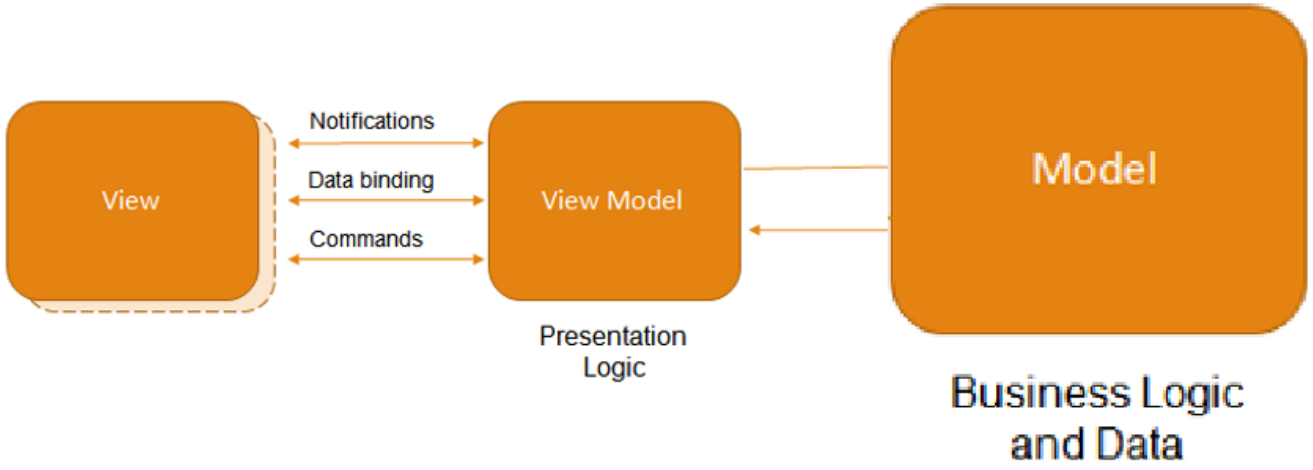
- **Separation of Concerns**: Ensures application logic, user interface, and presentation logic are distinct and separate.
- **Data Binding**: Facilitates automatic updates between the View and ViewModel, reducing boilerplate code.
- **Testability**: Due to separation, especially the ViewModel, components can be tested independently of the user interface.
- **Maintainability**: With clear responsibilities, applications become more maintainable and understandable.

The Powerhouse of Business Logic (Model)

- "In MVVM, while Views and ViewModels play their roles in presentation and interaction, the Model remains the foundation that holds and processes our application's core data and business logic."

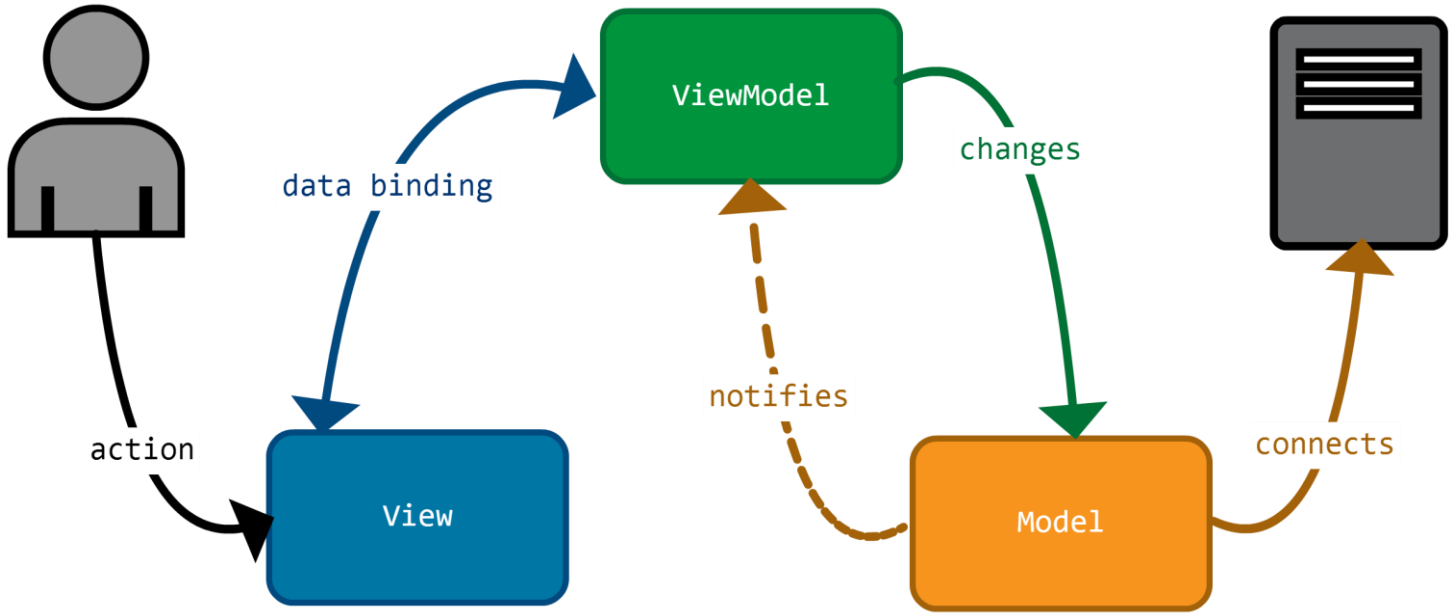
Role of the Model

- "The Model is typically independent of the user interface and presentation layer. It focuses purely on data and domain logic."



How Model Interacts with ViewModel

- "The ViewModel requests and sends data to the Model. While the ViewModel can format this data for presentation, any core business rules or logic remain in the Model. “



Characteristics of the Model

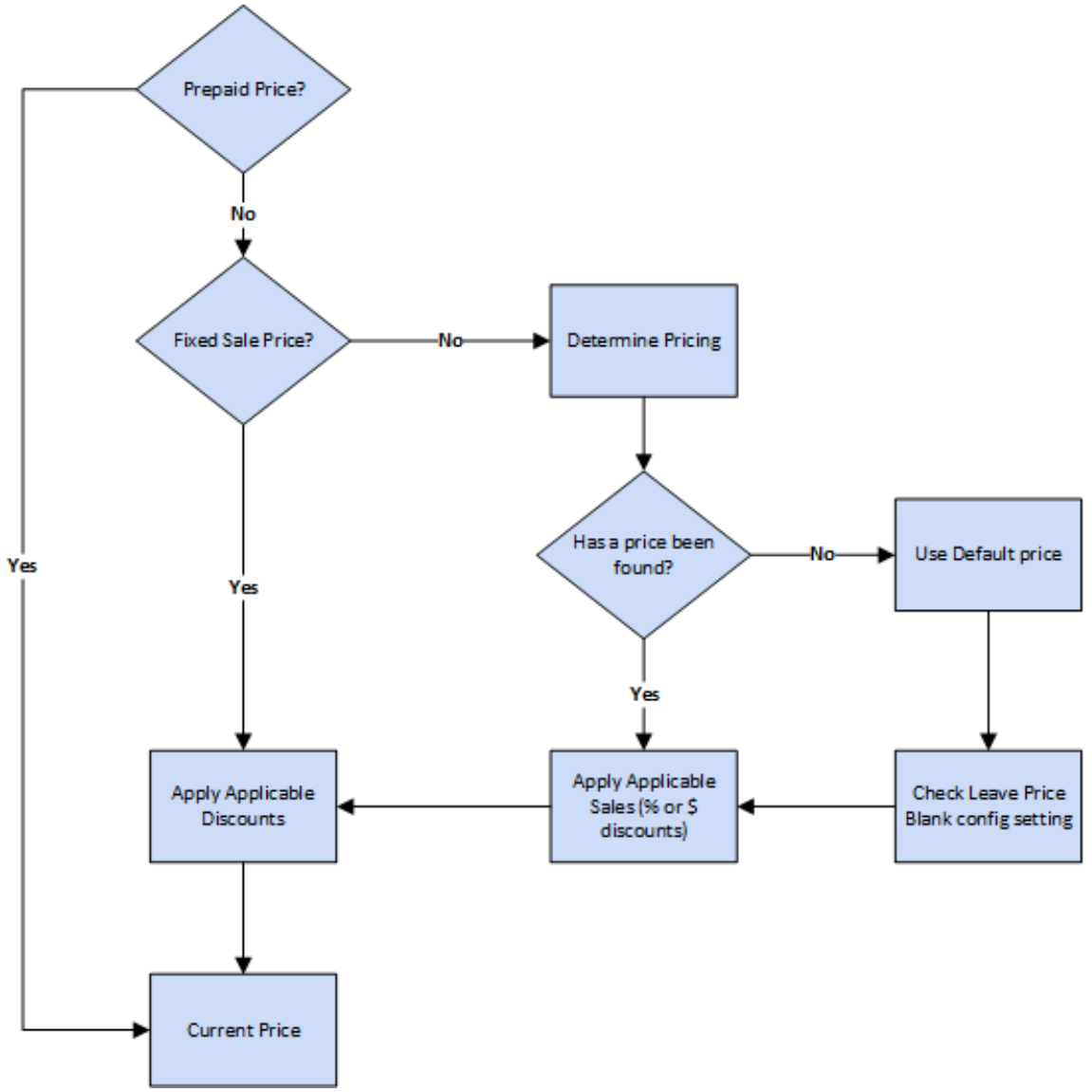
- **Data Retrieval:**
  - **Interactions with databases or web services are used** to fetch data.
- **Data Storage**:
  - Storing and maintaining data integrity.
- **Business Logic**:
  - Computation and operations on data.
- **Data Validation**:
  - Ensuring the correctness and validity of data.
- **Communication**:
  - Sending processed data to the ViewModel.

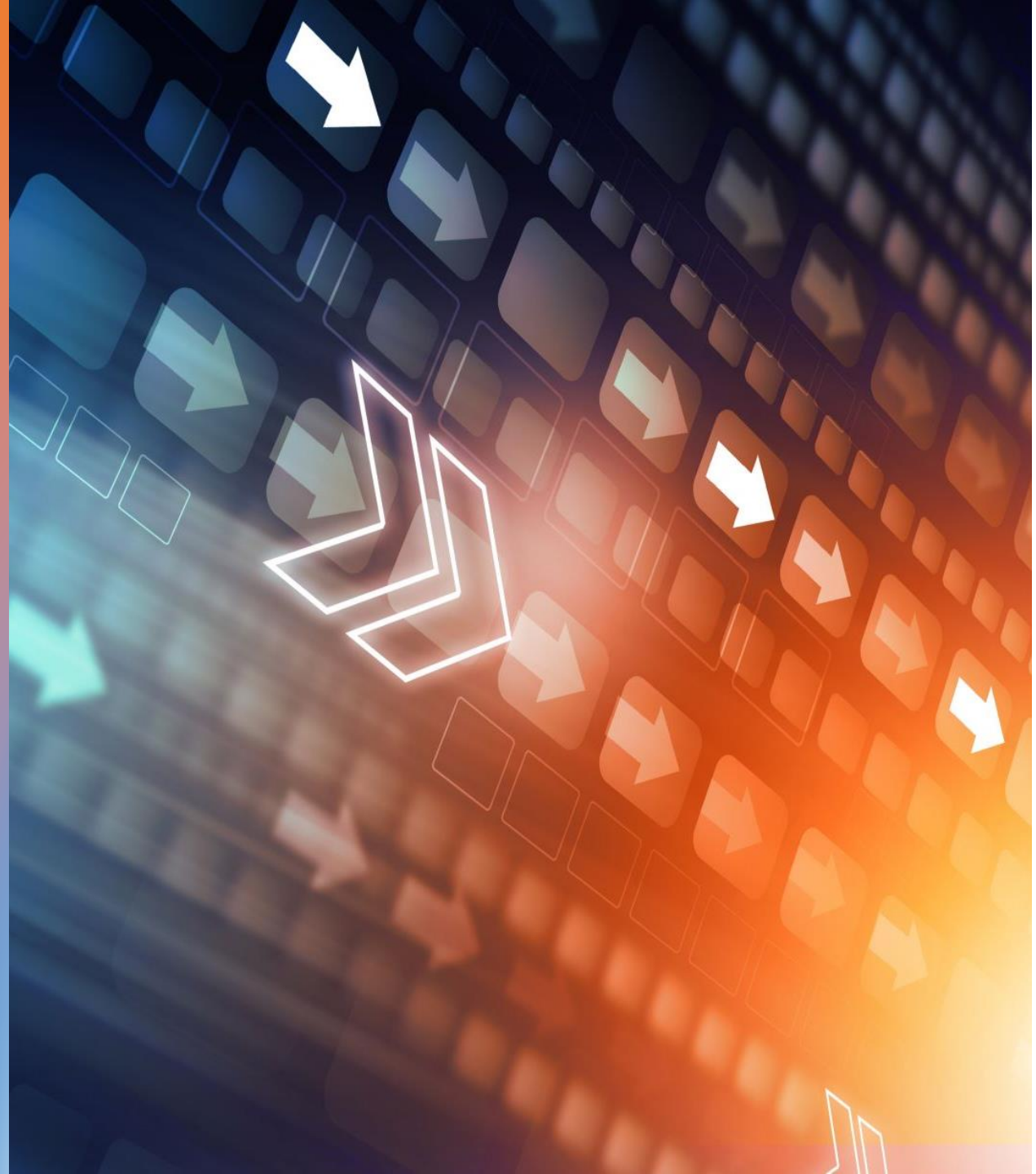
Data Sources

- "Models can retrieve data from a variety of sources, translating them into a format suitable for the application.“
  - Databases
  - APIs
  - Files
  - Other Sources

Business Logic

- Refers to the systematic set of guidelines followed by a system.
- Can involve both simple and complex operations, depending on the application.
- Establishes the mechanisms through which existing data can be modified or updated.
- Business Logic is crucial for ensuring data integrity, consistency, and accuracy within a system.
- Often, Business Logic is distinct from the user interface, ensuring that the core processes of a system remain consistent regardless of the UI changes

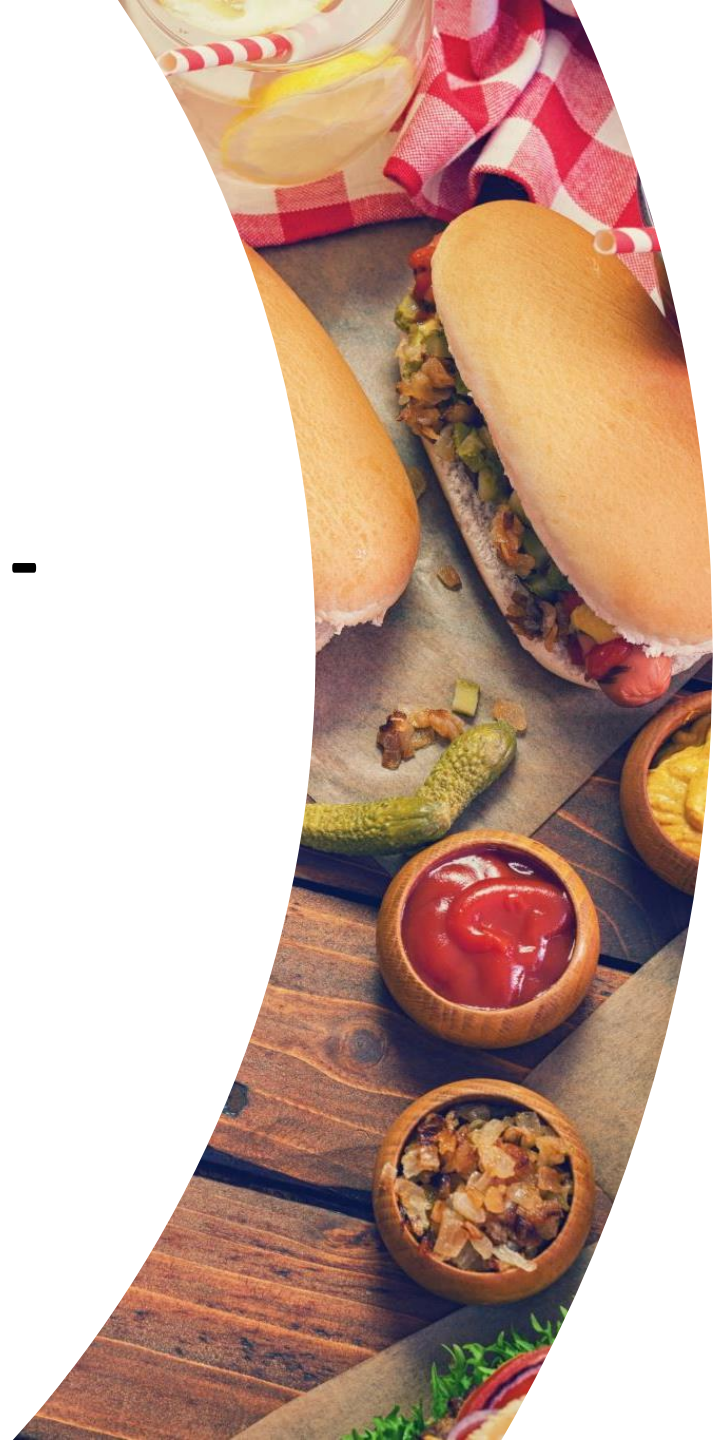




Data Validation

- **Ensure Data Integrity**: Keeping the data consistent and error-free.
- **User Feedback**: Informing users of incorrect inputs or errors.
- **Security**: Preventing malicious inputs and attacks.

Communication - Asynchronous Operations



- **Using tasks, promises, or other asynchronous patterns to fetch and process data.**

- **Why Asynchronous?** To avoid freezing the UI during long data operations.

Imagine you're in a cafeteria with your friends. You're all hungry and want to order some food.

- **Synchronous (the opposite of Asynchronous)**: It's like there's only one food counter. When one friend goes to order, everyone else has to wait in line until that friend gets their food and comes back. Then, the next friend can order. If the food counter is slow or one of your friends orders a lot, everyone else must wait a long time!
- **Asynchronous**: Now, imagine there are many food counters. Each of your friends can go to a different counter simultaneously. This way, even if one friend is slow or one counter is busy, the others can still get their food without waiting. Everyone is doing their thing at their own pace, and no one is making everyone else wait.

Interaction with ViewModel

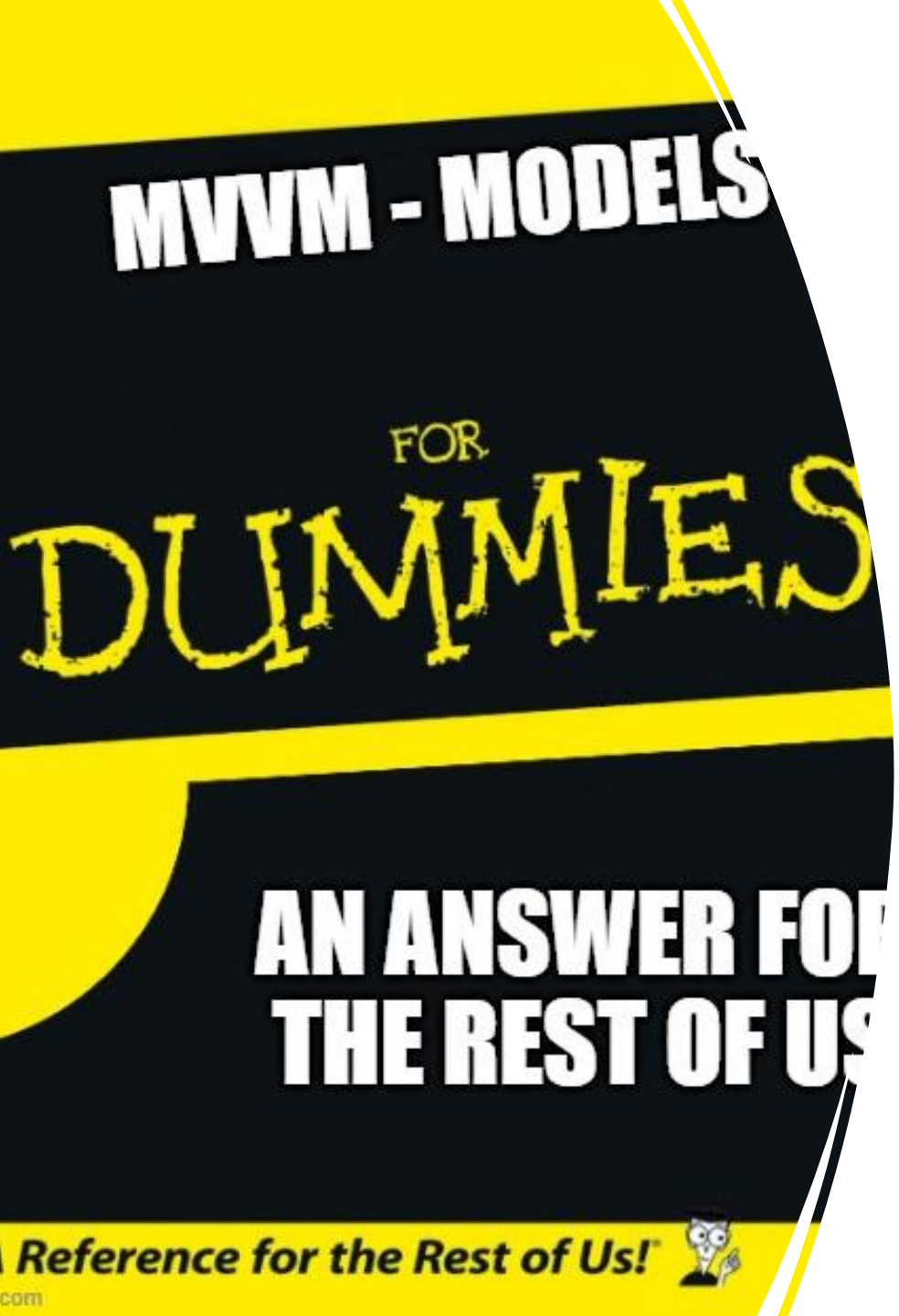
Principles for Creating an Effective Model

Ensure separation of concerns – The Model shouldn't be aware of UI specifics.

Make it scalable – Anticipate the growth of data and business rules.

Implement efficient data operations, including interactions with databases, APIs, and other external systems.

Always prioritize data integrity and security.



- **Data Storage**:
  - Think of the Model as your backpack. It's where you keep all your stuff (data) like books (information) and notes (details).

- **No Fancy Looks**:
  - The Model is like the plain text in a book. It doesn't care about the book's cover design or illustrations. It's just focused on storing and providing the actual content.

- **Independent Worker**:
  - Imagine a chef in a kitchen. The chef (Model) prepares the food but doesn't serve it to the customers. Others (View and ViewModel) handle that. The chef just ensures the food is ready and tasty.

Statement Regarding Slide Accuracy and Potential Revisions

- Please note that the content of these PowerPoint slides is accurate to the best of my knowledge at the time of presentation. However, as new research and developments emerge, or to enhance the learning experience, these slides may be subject to updates or revisions in the future. I encourage you to stay engaged with the course materials and any announcements for the most current information

Importance of Soft Deletes

- **Soft Deletes vs. Hard Deletes**
  - Preservation of historical data
  - Facilitates data recovery
  - Reduces risks of accidental data loss
- **Relevance of RemoveFromViewFlag**
  - Enables filtering out "deleted" records
  - Maintains database integrity
  - Streamlines user experience by hiding "deleted" data without actual data loss