Practical Exercise: Architectural Patterns for Mobile Applications

MVC, MVP, MVVM, Microservices in Practice

Course: Mobile Application Development (Chapter 4)
Level: Technical University Students
Duration: 6–8 hours self-study + 1–2 hours online discussion
Format: Individual Analysis Report with Design Decisions (2–3 pages)

Exercise Overview

In this exercise, you will analyze and compare four major architectural patterns used in mobile and backend development: **MVC, MVP, MVVM, and Microservices**. You'll learn when to apply each pattern and make design decisions for real-world scenarios.

Key Learning Outcomes:

- Understand the core separation of concerns in each architectural pattern
- Recognize strengths, weaknesses, and trade-offs of each approach
- Apply patterns to appropriate mobile app scenarios
- Design layered architectures suitable for different project scales
- Develop strategic thinking about architecture evolution

Part A: Pattern Comparison & Analysis

Task 1: Pattern Matrix (Self-Study: 2–3 hours)

Objective: Build a comprehensive comparison table of architectural patterns.

Instructions:

Complete the following comparison matrix for **MVC, MVP, MVVM, and Microservices**:

Aspect	MVC	MVP	MVVM	Microservices
Layers	Model, View, Controller	Model, View, Presenter	Model, View, ViewModel	Independent services (API Gateway, Services, Databases)
Data Flow	Bidirectional (View ↔ Controller ↔ Model)	Unidirectional (View → Presenter → Model)	Bidirectional with data binding (View ↔ ViewModel ↔ Model)	Asynchronous (Service-to-Service via APIs/Events)
Best For	Simple apps, prototypes, MVPs	Medium apps, complex UI logic, testing	Complex UI, data binding, MVVM frameworks (e.g., .NET, WPF)	Large systems, multiple teams, scalable backends
Testing Difficulty	Medium (tight View-Controller coupling)	Easy (Presenter is testable in isolation)	Easy (ViewModel logic independent of View)	Medium (requires service mocking, integration tests)
Team Scalability	Poor (bottleneck: shared Controller)	Good (Presenter logic isolated)	Good (ViewModel-View separation)	Excellent (independent service teams)
Mobile Platform Use	Android (older), some iOS	Android, iOS (some)	Android (recent), iOS (MVVM-C)	Backend only (consumed by mobile clients)
Main Strength	Simple, quick to build	Clean separation, testable	Two-way data binding reduces boilerplate	Scalability, independent deployment, fault isolation
Main Weakness	Poor testability, god objects	Extra layer overhead for simple apps	Overkill for simple screens, ViewModel complexity	Distributed complexity, network latency, debugging difficulty

Deliverable: Completed comparison table with 1–2 sentence explanations for any cells you find ambiguous.

Task 2: Architecture Diagram Sketches (Self-Study: 1.5–2 hours)

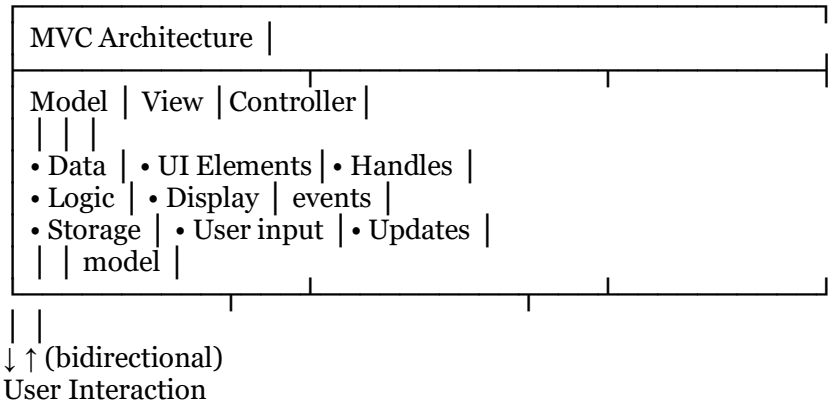
Objective: Visualize component relationships in each pattern.

Instructions:

For **each of the four patterns**, draw or describe a simple block diagram showing:

- Primary components (layers/services)
- Data flow direction (arrows labeled with examples)
- Communication method (direct call, event, API, etc.)

Example: MVC Pattern



Create similar diagrams for:

- MVP (showing Presenter isolation)
- MVVM (showing ViewModel & data binding)
- Microservices (showing service decomposition)

Deliverable: Four architecture diagrams with labeled components and data flows.

Part B: Scenario-Based Decisions

Task 3: Pattern Selection for Three Scenarios (Self-Study: 2–3 hours)

Objective: Match architectural patterns to real-world project needs.

Instructions:

For **each of the three scenarios**, decide on:

1. Which UI pattern to use (MVC/MVP/MVVM)
2. Whether to use Microservices on the backend (Yes/No)
3. Justify your choice in 3–5 sentences

Scenario A: Simple Habit-Tracker Mobile App

Context:

- Feature: Users log daily habits (meditation, exercise, water intake)
- Screens: Home (list of habits), Detail (log entry), Settings
- Backend: Simple REST API (Node.js/Python)
- Team: 2 developers, 1 backend engineer
- Scale: ~10,000 monthly active users (MAU)
- Data: Offline-first (sync when online)

Your Decision:

- **UI Pattern:** _____
- **Microservices:** Yes / No
- **Justification:** (3–5 sentences)

Example Answer:

> **UI Pattern:** MVP
> **Microservices:** No
> **Justification:** MVP provides good separation of UI logic from business logic without excessive overhead. A simple monolithic backend suffices for this scale—habit logging, user auth, and sync are straightforward. Two mobile developers benefit from clean Presenter layer for testing UI flows. Backend logic (habit aggregation, analytics) remains simple enough for a single Node.js/Python service.

Scenario B: Social Chat Application

Context:

- Features: Real-time messaging, contacts, media sharing (photos/video), notifications, user presence
- Platforms: iOS + Android native
- Backend: WebSocket server, multiple microservices possibility
- Team: 4–5 mobile developers, 3–4 backend engineers
- Scale: ~1 million MAU, complex real-time requirements
- Data: Cloud-based, real-time sync

Your Decision:

- **UI Pattern:** _____
- **Microservices:** Yes / No
- **Justification:** (3–5 sentences)

Scenario C: Large Ride-Hailing Platform (e.g., Uber-scale)

Context:

- Features: Ride matching, real-time GPS tracking, payments, driver/passenger ratings, multiple product tiers (UberX, UberPool, etc.)
- Platforms: iOS, Android, web, driver app, passenger app
- Backend: Critical scalability & reliability needs
- Team: 50+ engineers (multiple teams: iOS, Android, backend, ML, infrastructure)
- Scale: 10+ million MAU, globally distributed
- Data: Complex transactions, high availability required

Your Decision:

- **UI Pattern:** _____
- **Microservices:** Yes / No
- **Justification:** (3–5 sentences)

Deliverable: Three scenarios with clear architectural choices and well-reasoned justifications (3–5 sentences each).

Part C: Refactoring & Evolution

Task 4: Architecture Migration (Self-Study: 1–1.5 hours)

Objective: Understand how architectures evolve as apps grow.

Instructions:

A startup built its Android app in **MVC** two years ago. Now:

- The app has grown to 15+ screens
- UI logic is increasingly complex (animations, state management, offline sync)
- Testing is difficult—UI and business logic are tightly coupled
- The team has grown from 2 to 6 developers
- New developers struggle to understand the "god Controller" patterns

Your Task:

1. **Identify the Problem (1–2 sentences):**

    - Why is MVC becoming inadequate?
2. **Propose a Target Architecture (1 sentence):**
    - Which pattern (MVP or MVVM) would you migrate to? Why?
3. **Outline Migration Steps (5–7 steps):**
    - Step 1: Extract UI event handlers from Activities/Fragments into a new Presenter/ViewModel
    - Step 2: [Your steps…]
    - …and so on
4. **Discuss Trade-offs (3–4 sentences):**
    - What benefits will you gain?
    - What risks or costs might the migration introduce?

Example (Partial):

> **Problem:** MVC tightly couples View and business logic in Activities, making unit testing impossible and new features increasingly difficult to add.
>
> **Target Architecture:** MVVM with LiveData/StateFlow
>
> **Migration Steps:**
>
> **Trade-offs:** Developers must learn ViewModel patterns and reactive programming (learning curve). Initial refactoring takes 2–3 sprints. However, testing becomes much easier, code is more maintainable, and new features are faster to implement long-term.

Deliverable: Problem statement, target architecture choice, 5–7 migration steps, and 3–4 sentence trade-off analysis.

Part D: Strategic Reflection

Task 5: Architecture Decision Framework (Online: 0.5–1 hour)

Objective: Develop strategic thinking about architecture choices.

Instructions:

Answer **all three questions** (250–350 words total):

1. **Why Not Microservices Everywhere?**
    - The Habit Tracker scenario is small; why not use Microservices on the backend from day one?
    - What costs does premature microservices introduce? (Hint: think about distributed debugging, DevOps, testing complexity)

2. **Pattern Evolution:**
    - Imagine the Habit Tracker grows to 10 million MAU with complex social features (friend recommendations, activity feeds, leaderboards).
    - How might the architecture evolve? (e.g., MVC → MVVM, monolith → microservices)
    - What triggers each migration?
3. **Your Project Hypothesis:**
    - If you were starting a **new mobile app project today**, what combination of UI pattern + backend architecture would you choose as your **default starting point**?
    - Why is this combination a good "first bet" for most projects?

Deliverable: Answers to all 3 questions, 250–350 words total.

Evaluation Criteria

Criteria	Excellent (9–10)	Good (7–8)	Acceptable (5–6)	Needs Improvement (<5)
Pattern Understanding	All patterns clearly explained; distinctions are precise	3 of 4 patterns well understood; minor confusion	Basic understanding of patterns; some gaps	Vague or incorrect explanations
Comparative Analysis	Insightful trade-offs identified; well-reasoned table	Good comparison; mostly accurate	Basic comparison; some oversimplifications	Incomplete or inaccurate
Scenario Decisions	All 3 scenarios have well-justified choices; considers trade-offs	2–3 scenarios well-justified; clear reasoning	Scenarios addressed; reasoning is shallow	Weak or unjustified choices
Migration Planning	Clear steps with rationale; realistic assessment of effort & risk	Good step sequence; adequate trade-off analysis	Steps present; trade-offs partially addressed	Incomplete or unrealistic plan

Strategic Thinking	Thoughtful reflection on architecture maturity levels and scaling	Good reflection; considers alternatives	Basic reflection; limited depth	Minimal or missing reflection
Communication	Clear, well-organized, professional diagrams & narrative	Generally clear; minor organization issues	Readable but lacks polish	Disorganized or unclear

---

Deliverable Checklist

Submit as **PDF or DOCX (2–3 pages)** containing:

- [ ] **Task 1:** Completed comparison matrix (MVC, MVP, MVVM, Microservices)
- [ ] **Task 2:** Four architecture diagrams with labeled components and data flows
- [ ] **Task 3:** Three scenario decisions with 3–5 sentence justifications each
- [ ] **Task 4:** Migration plan (problem → target → 5–7 steps → trade-off analysis)
- [ ] **Task 5:** Strategic reflection (3 questions answered, 250–350 words total)
- [ ] **References:** Cite sources on architecture patterns (optional but recommended)

Resources & References

Architecture Pattern References:

- Fowler, Martin. *Patterns of Enterprise Application Architecture* (Chapter 2: Layered Architecture)
- Microsoft Docs: MVVM Pattern
- Google Android Architecture Guide: Guide to app architecture

Microservices:

- Newman, Sam. *Building Microservices* (Chapter 1: Microservices)
- Martin Fowler: Microservices
- AWS: What are Microservices?

Mobile Architecture Best Practices:

- Apple Human Interface Guidelines: App Architecture

- Android Architecture Components: https://developer.android.com/jetpack/guide

Tools for Diagramming:

- Draw.io, Lucidchart, Miro, or paper sketch (any method acceptable)

Timeline Guidance

Self-Study (6–8 hours):

- Hours 1–3: Task 1 (pattern comparison & matrix completion)
- Hours 3–5: Task 2 (architecture diagrams & data flow visualization)
- Hours 5–7: Task 3 (scenario analysis & decision justification)
- Hours 7–8: Task 4 (migration planning & trade-off analysis)

Online Sessions (1–2 hours):

- 0.5 hours: Task 1–2 presentation (pattern overview, diagram review)
- 0.5 hours: Task 3 discussion (scenario decisions, class debate on trade-offs)
- 0.5 hours: Task 4 peer review (migration plans, feasibility assessment)
- 0.5 hour: Task 5 reflection & Q&A (architecture maturity, scaling decisions)

Extension Tasks (Optional, Advanced)

1. **Real Codebase Analysis:**
    - Download an open-source Android/iOS app (e.g., from GitHub)
    - Identify its current architectural pattern (MVC/MVP/MVVM)
    - Propose refactoring steps to improve architecture
2. **Backend Architecture Deep Dive:**
    - Design a microservices architecture for the Ride-Hailing scenario
    - Identify individual services (Auth, Matching, Payments, Notifications, etc.)
    - Document API contracts between services
3. **Performance & Scalability:**
    - Compare the performance implications of each UI pattern (UI rendering, memory usage, battery drain)
    - Research architectural patterns for high-scale backends (CQRS, Event Sourcing)
4. **Architecture Decision Records (ADRs):**

    - Write formal ADRs documenting why you chose each pattern for your scenarios
    - Follow the ADR template: Status, Context, Decision, Consequences
5. **Code Migration Exercise:**
    - Take a small MVC codebase and refactor it to MVVM or MVP
    - Document the refactoring steps and improvements

Notes for Instructors

This exercise bridges **theory (architectural patterns) and practice (real-world design decisions)**. Students develop:

- Deep understanding of layered architecture principles
- Ability to match patterns to project scale and complexity
- Strategic thinking about technical debt and architecture evolution
- Appreciation for trade-offs (simplicity vs. testability vs. scalability)

Discussion prompts for online session:

- "When is 'over-engineering' with microservices a mistake?"
- "How do team size and code complexity drive architectural choices?"
- "What signs indicate an app has outgrown its current architecture?"
- "How do mobile and backend architectures influence each other?"
- "What architecture would you choose for a startup with uncertain requirements?"

Common pitfalls to address:

- Choosing Microservices prematurely (distributed complexity without gain)
- Ignoring team skill levels when selecting patterns
- Migrating architectures without clear business drivers
- Confusing UI patterns with backend architecture
- Assuming one pattern fits all screens/features

Assessment Notes

Grading rubric integration:

- Emphasizes *reasoning* over perfect technical knowledge
- Values recognition of trade-offs and constraints
- Recognizes that multiple valid designs exist for most scenarios

- Encourages pragmatism (context-driven decisions)

Partial credit guidance:

- Pattern understanding: 25% (depth of knowledge)
- Scenario analysis: 40% (quality of decision-making)
- Migration planning: 20% (realism and feasibility)
- Strategic thinking: 15% (depth of reflection)