

Architecture Analysis Report

TreeWayz Ride-Sharing Application (Sprint 2)

1. Introduction

The TreeWayz mobile app connects riders and drivers through a clean, responsive interface supporting ride posting, booking, payment, live tracking, and feedback.

This report analyzes three candidate architectures—Layered, MVC, and Microservices—to select the best fit for TreeWayz based on scalability, security, maintainability, and team constraints.

All evaluations reference the functional and non-functional user stories and the Frontend Design from Sprint 1.

2. Architectural Style 1 – Layered Architecture

Overview

Divides the system into independent layers, each responsible for a specific concern.

Layer	Responsibility	TreeWayz Examples
Presentation	Mobile UI – login, signup, ride lists, receipts	<i>Prototype 5 screens 1–4</i>
Application	Coordinates booking, posting, matching workflows	Handles “Book Seat,” “Post Ride” use cases
Domain	Business logic & rules	Fare calculation (5 BD), seat updates, ride state
Data	Storage & external services	Firebase / SQL DB, Maps API, Benefit API

Advantages

- **Scalability:** Vertical scaling and caching support *NF1 Performance* (≤ 2 s load time).
- **Security:** Centralized auth layer aligns with *NF2 Security* and encrypted credentials.
- **Maintainability:** Each layer can be updated without breaking others (supports *NF4 Reliability*).
- **Team Fit:** Four-member academic team can divide by layer (UI, Logic, Data) while maintaining one codebase.

Disadvantages

- Limited horizontal scaling at very high load.
- Requires discipline to prevent cross-layer coupling.

3. Architectural Style 2 – Model–View–Controller (MVC)

Overview

Separates interface, logic, and data.

Component	Responsibility	TreeWayz Examples
Model	Data & state management	Ride, User, Booking entities
View	Visual representation	“Available Rides,” “Rate Driver,” “Receipts”
Controller	Handles input, updates model/view	Login, Booking, Payment handlers

Advantages

- Parallel UI and backend development; good for *NF3 Usability*.
- Controllers validate inputs (login, payment).
- Clearer testability for booking and rating flows.

Disadvantages

- “Fat Controller” risk as business logic grows.
- Less suited for distributed scalability.

4. Architectural Style 3 – Microservices Architecture

Overview

Decomposes the app into independent services (User, Ride, Booking, Payment).

Advantages

- **Scalability:** Each service scales individually (future growth).
- **Resilience:** Ride service failure doesn’t crash payments.

- **Flexibility:** Different tech stacks per service.

Disadvantages

- **Security:** Many endpoints increase attack surface (*NF2 risk*).
- **Maintainability:** Debugging is complex due to distributed state.
- **Team Fit:** High DevOps overhead unsuitable for 4-week student sprint.

5. Decision Matrix

To ensure an objective architectural choice, TreeWayz's requirements were evaluated using a weighted decision matrix. Each criterion (scalability, security, maintainability, and team fit) was assigned a weight percentage based on its relative importance to the project goals and user stories.

Scoring Explanation (1–5 scale)

Score	Meaning
1 – Poor	Fails to meet TreeWayz's minimum requirement for this criterion
2 – Fair	Meets some requirements but introduces significant challenges
3 – Good	Sufficient performance for MVP but with moderate limitations
4 – Very Good	Performs well and aligns with most project needs
5 – Excellent	Fully satisfies TreeWayz's needs with long-term scalability

Criteria Weighting

- Scalability (30%) – TreeWayz must handle increasing booking activity (*NF1 Performance*).
- Security (25%) – All user credentials and payment data must be protected (*NF2 Security*).
- Maintainability (25%) – Architecture must support easy feature updates during the semester (*NF4 Reliability*).
- Team Fit (20%) – Must align with the 4-member academic team's time and skill constraints.

Weighted Decision Matrix

Criteria	Weight	Layered (Score × Weight)	MVC (Score × Weight)	Microservices (Score × Weight)	Explanation
Scalability	0.30	$4 \times 0.30 = 1.20$	$3 \times 0.30 = 0.90$	$5 \times 0.30 = 1.50$	Microservices scale independently; Layered supports expected MVP load.
Security	0.25	$4 \times 0.25 = 1.00$	$4 \times 0.25 = 1.00$	$3 \times 0.25 = 0.75$	Layered and MVC use centralized authentication; Microservices require distributed endpoint protection.
Maintainability	0.25	$5 \times 0.25 = 1.25$	$4 \times 0.25 = 1.00$	$3 \times 0.25 = 0.75$	Layered enables isolated updates for features like Booking and Payment.
Team Fit	0.20	$5 \times 0.20 = 1.00$	$4 \times 0.20 = 0.80$	$2 \times 0.20 = 0.40$	Layered suits small academic teams using one GitHub repo and shared toolchain.
Total Weighted Score	1.00	4.45 / 5	3.70 / 5	3.40 / 5	—

Interpretation

- The Layered Architecture achieves the highest weighted average score (4.45/5), indicating a strong balance across all critical dimensions.
- It performs especially well in maintainability and team fit, two factors crucial for an agile student project.
- Although Microservices outperform in scalability, the overhead and DevOps complexity make it unsuitable for TreeWayz's current scope.

Result: Layered Architecture provides the best trade-off between technical capability, team practicality, and project timeline, while still allowing gradual evolution into a microservices model as the platform scales.

6. Final Choice & Justification

Chosen Architecture: Layered Architecture

The Layered Architecture received the highest overall score (4.5/5) and best matches TreeWayz's functional requirements, team capabilities, and project timeline.

User Story / Requirement	Layered Architecture Support
1.1 Sign-Up & 1.2 Login	Centralized Application Layer validates credentials, enforces encryption (NF2).
2.1 Post Ride (Driver)	Domain Layer handles fare calculation and seat count; Data Layer stores ride info.
4.1 View Rides / 4.2 Book Seat	Presentation Layer shows available rides in real time; Application Layer updates seat availability via repositories.
4.3 Payment Method	Application Layer handles selection workflow; Data Layer records method for receipts.
6.1 Rate Ride / Driver	Domain Layer records feedback logic; Presentation Layer shows rating prompt (<i>NF3 Usability</i>).
8.1 RideTrack (Safety)	Observer Pattern within Domain Layer pushes live alerts to UI (NF4 Reliability).
Team Constraint	Five-person team using GitHub benefits from a single, maintainable codebase.

Future Scalability Plan

TreeWayz can later transition selected layers (e.g., Booking Service) into microservices, if there are more traffic and time allocated for the project (if we decided to expand from this being a small-scale university project).

Example Folder Structure

```
apps/
  mobile/      → UI (Presentation)
  api/
    controllers/ → MVC controllers
  src/
    app/        → Application logic (Booking, Matching)
    domain/     → Business rules (Ride, User, Payment)
    infra/      → Database + external APIs
```

7. Design Patterns Mapped to TreeWayz

1. Repository Pattern (Data Layer)

Purpose: Separate persistence from logic.

Usage: RideRepository, UserRepository, BookingRepository manage CRUD operations.

Benefit: Supports system maintainability by ensuring convenience of app development during testing and update phases.

```
Ride ride = rideRepository.findById(rideId);
ride.start();
rideRepository.save(ride);
```

2. Observer Pattern (Notification & RideTrack System)

Purpose: Enable real-time updates between components.

Usage: When driver accepts a ride, NotificationService and RideTrackUI are notified.

Benefit: Fulfils NF4 Reliability and 8.1 RideTrack Safety alert story.

```
ride.attach(NotificationService())
ride.change_status("Accepted") # Notifies observers instantly
```

8. Conclusion

The **Layered Architecture** provides the optimal balance of scalability (4.5/5), security, maintainability, and team fit for the *TreeWayz* application.

It aligns tightly with the user stories (1.1 → 8.2) and non-functional goals (NF1–NF6), ensuring both a smooth frontend experience and a robust backend structure.

Paired with the Repository and Observer patterns, this architecture ensures a secure, maintainable, and scalable foundation for TreeWayz's real-time ride-sharing ecosystem.

Final Decision: Adopt a **Layered Architecture** integrated with MVC controllers and Repository/Observer patterns to balance clarity, performance, and future growth.