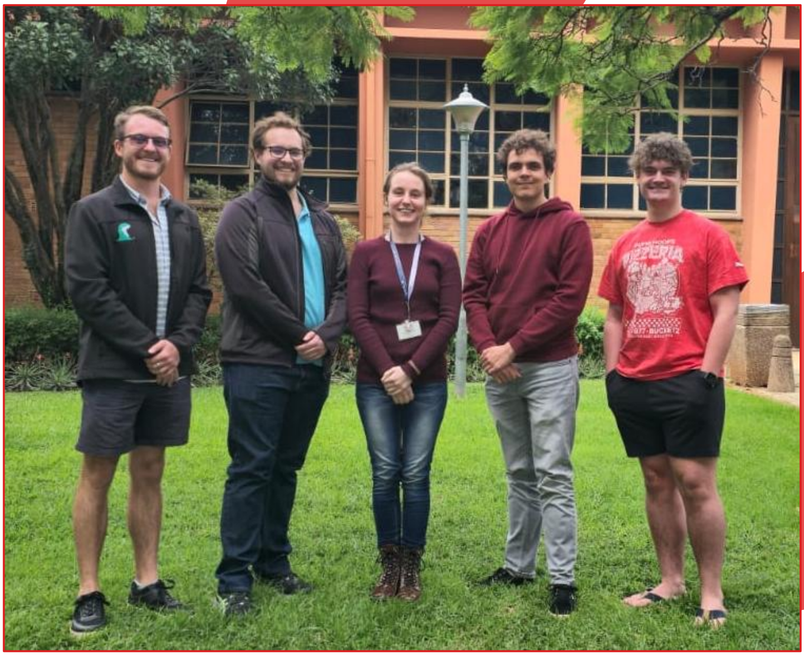
SuperLap Racing Line Optimization System

EPI-USE



Quintessential

Amber Ann Werner [u21457752]

Milan Kruger [u04948123]

Qwinton Knocklein [u21669849]

Sean van der Merwe [u22583387]

Simon van der Merwe [u04576617]

Architectural Requirements

Architectural Design Strategy

This system adopts a **Design Based on Quality Requirements** strategy to guide architectural decisions. A diverse set of non-functional requirements – including real-time performance, security, scalability, and constraints related to maintainability, availability, and cost – necessitated a quality-driven approach.

These requirements informed key architectural choices such as:

- The adoption of an Event-Driven Architecture to support responsiveness and decoupling,
- The use of GPU offloading and model caching to meet real-time performance goals,
- Implementation of API gateways and role-based access control for security enforcement,
- A microservices-based structure combined with infrastructure-as-code for maintainability and scalable deployment.

By deriving architectural patterns from quality attributes, the system maintains alignment with both stakeholder expectations and technical constraints from the outset. This strategy ensures the architecture remains robust, adaptable, and performance-optimized under real-world conditions.

Quality Requirements

NF1: Performance Requirements

- **NF1.1:** The system will process and analyse a racetrack image (≤10MB) in under 5 seconds.
- **NF1.2:** AI training simulations will run at ≥30 FPS for real-time feedback during optimization.
- **NF1.3:** Lap time predictions will be computed within 1 second after track processing.

- **NF1.4:** The system will support at least 50 concurrent users in cloud-based mode.

NF2: Security Requirements

- **NF2.1:** All user-uploaded track images and telemetry data will be encrypted in transit (HTTPS/TLS 1.2+).
- **NF2.2:** Sensitive user data (e.g: login credentials) will be stored using salted hashing (bcrypt/PBKDF2).
- **NF2.3:** The system will enforce role-based access control (RBAC) for admin vs. end-user privileges.
- **NF2.4:** AI models and training data will be protected against unauthorized modification.

NF3: Reliability & Availability

- **NF3.1:** The system will maintain 95% uptime under normal operating conditions.
- **NF3.2:** Critical failures (e.g: RL training crashes) will recover automatically within 10 minutes.
- **NF3.3:** Backup procedures will ensure no more than 1 hour of data loss in case of system failure.
- **NF3.4:** The offline mode will retain core functionality (track processing, pre-trained AI suggestions) without cloud dependency.

NF4: Usability Requirements

- **NF4.1:** The interface will be intuitive for non-technical users (e.g: drag-and-drop track uploads, one-click simulations).
- **NF4.2:** Visualizations (racing line overlays, metrics) will adhere to colourblind-friendly palettes.
- **NF4.3:** The system will provide tooltips/guided tutorials for first-time users.
- **NF4.4:** All critical actions (e.g: deleting data) will require user confirmation.

NF5: Scalability Requirements

- **NF5.1:** The system will scale horizontally to support up to 10,000 simulations/day via cloud resources.
- **NF5.2:** Modular architecture will allow integration of new physics models or RL algorithms without major refactoring.

- **NF5.3:** GPU-accelerated training will dynamically allocate resources based on workload.

NF6: Compatibility Requirements

- **NF6.1:** The system will support Windows, macOS, and Linux for desktop applications.
- **NF6.2:** Web-based access will be compatible with Chrome, Firefox, and Edge (latest versions).
- **NF6.3:** Track images will be accepted in JPEG, PNG, or SVG formats (≤10MB).

NF7: Maintainability Requirements

- **NF7.1:** Code will be documented with API specs, inline comments, and version control (Git).
- **NF7.2:** The system will log errors with timestamps, severity levels, and recovery suggestions.
- **NF7.3:** Third-party dependencies (e.g: PyTorch, OpenCV) will be pinned to stable versions.

NF8: Cost & Resource Constraints

- **NF8.1:** Cloud computing costs will not exceed R5000 (aligned with project budget).
- **NF8.2: Offline mode will operate on consumer-grade hardware (e.g: NVIDIA GTX 1060+ for GPU acceleration).**

Architectural Strategies

NF1: Performance Requirements

- Microservices
- Microservices allow isolating performance-intensive tasks (e.g., image processing, AI inference), while event-driven

NF2: Security Requirements

- Service-Oriented Architecture (SOA)
- SOA is often chosen in enterprise systems for built-in security practices (e.g., HTTPS, RBAC, authentication layers across services).

NF3: Reliability & Availability

- Microservices
- Microservices support fault isolation and recovery (e.g., container restarts)

NF4: Usability Requirements

- Layered Architecture
- Layered architecture separates UI from business logic, supporting clean, intuitive interfaces and interaction layers.

NF5: Scalability Requirements

- Microservices
- Microservices allow independent scaling of services

NF6: Compatibility Requirements

- Client-Server
- Client-server supports access from different OS and browsers.

NF7: Maintainability Requirements

- Layered Architecture
- Layered and component-based architectures promote modularity, code isolation, and easier debugging/logging.

NF8: Cost & Resource Constraints

- Microservices
- Microservices support cost-effective scaling and offline deployment scenarios.

Architectural Designs and Patterns

Microservices Pattern (with Service-Oriented Architecture principles).

NF Category	*Strategy*
NF1: Performance, NF3: Reliability & Availability, NF5: Scalability, NF8: Cost & Resource Constraints	Decompose the system into small, autonomous services that can be developed, deployed, and scaled independently.

Why:

- Breaks the system into independently deployable services (e.g., Track Processing, Racing Line Optimization, AI Training, Visualization).
- Facilitates **independent scaling** of compute-heavy services such as AI training or image preprocessing.
- Supports **fault isolation** — a failure in one service does not crash the entire system.

Where Used:

- Backend services for track image processing, AI model training, and visualization.
- Node.js API Gateway orchestrates calls to microservices.

Event-Driven Architecture (EDA)

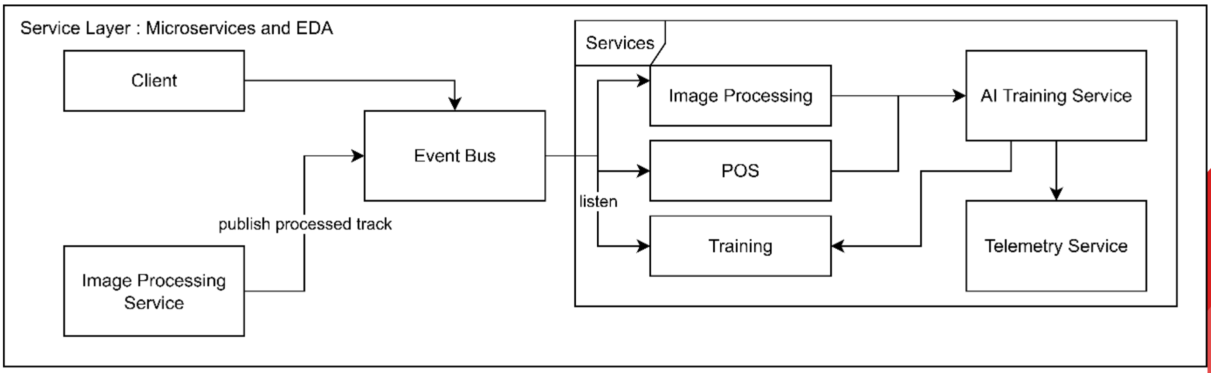
NF Category	*Strategy*
NF1: Performance, NF3: Reliability & Availability, NF5: Scalability	Use asynchronous messaging to decouple services, enabling parallel processing and reactive updates.

Why:

- Handles asynchronous workloads like reinforcement learning model training or simulation execution without blocking other system functions.
- Allows the backend to process tasks in parallel and notify the frontend upon completion.

Where Used:

- Training jobs queued in a message broker (e.g., RabbitMQ, Kafka).
- Events trigger updates in the visualization service when new results are ready.



MVVM (Model-View-ViewModel) in Unity Frontend

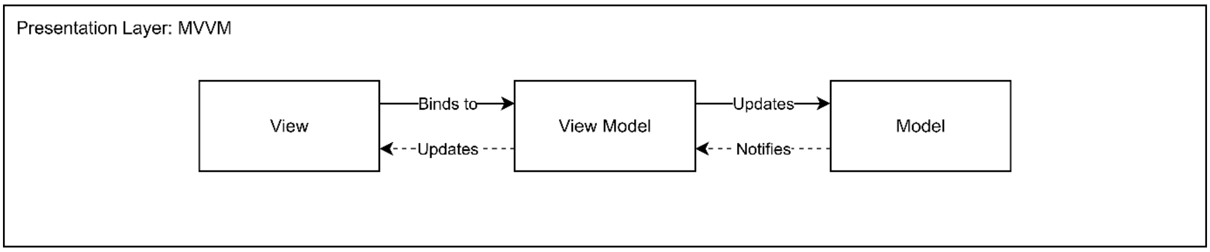
NF Category	*Strategy*
NF4: Usability, NF6: Compatibility, NF7: Maintainability	Separate presentation logic from business/application logic to simplify UI maintenance and enable responsive updates.

Why:

- Clean separation between UI logic (View) and application logic (ViewModel/Model).
- Easier maintenance of the Unity-based front end while supporting **real-time visualization** of racing lines.

Where Used:

- Frontend UI for uploading track images, controlling simulations, and viewing results in 2D/3D.



Repository Pattern for Data Layer

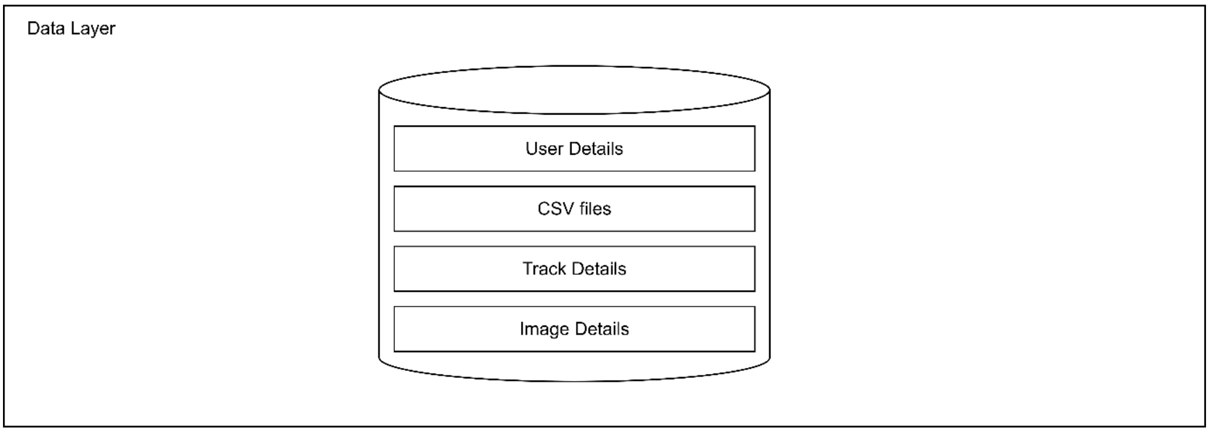
NF Category	*Strategy*
NF3: Reliability & Availability, NF7: Maintainability, NF2: Security	Abstract data access logic behind a repository layer to decouple business logic from persistence concerns.

Why:

- Centralizes data access logic, ensuring **business logic remains decoupled** from database queries.
- Improves maintainability and testing by abstracting database details.

Where Used:

- Persistent storage of user profiles, track data, AI models, and simulation logs.



API Gateway + Service Mesh for Security and Communication

NF Category	*Strategy*
NF2: Security, NF3: Reliability & Availability, NF5: Scalability	Centralize entry points for external requests and manage secure internal communication between services.

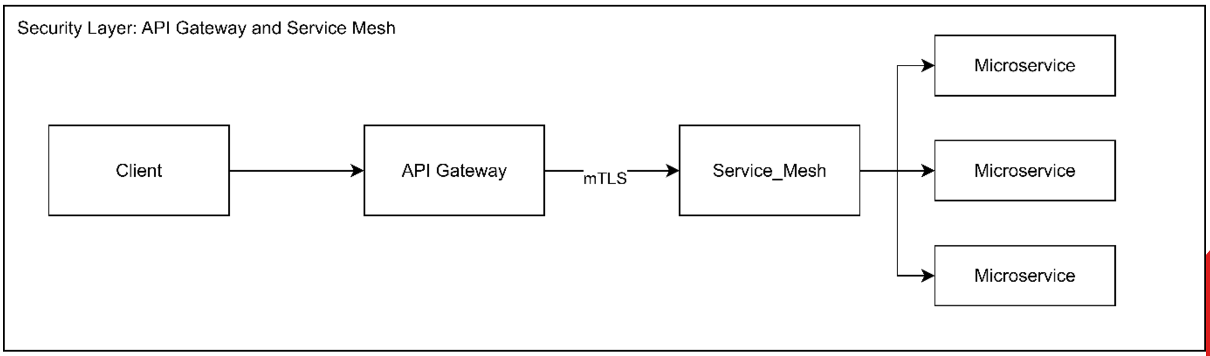
Why:

- API Gateway enforces authentication, authorization, rate limiting, and request validation.
- Service Mesh (e.g., Istio, Linkerd) ensures secure service-to-service communication with mTLS.

Where Used:

- All external requests pass through the API Gateway before reaching microservices.

Internal service communication is routed through the Service Mesh.



Layered Architecture

NF Category	*Strategy*
NF1: Performance, NF7: Maintainability	Break down image processing into sequential, independent stages to allow optimization and replacement of steps.

Why:

- Breaks image analysis into discrete, reusable steps (e.g., normalization → edge detection → track mapping).
- Allows independent optimization or replacement of each processing step.

Where Used:

- Track Image Processing microservice.

Architectural Constraints

Limited Real-World Telemetry Data

Obtaining authentic racing telemetry for supervised learning is challenging. Consequently, the system relies primarily on simulated or gaming data, which may not fully capture real-world nuances.

Model Reliability and Accuracy

AI outputs must be rigorously validated against established racing strategies to ensure accuracy and dependability, preventing flawed decision-making.

Image Processing Complexity

The system must accurately interpret 2D track images, correctly detecting circuit boundaries and optimal racing lines. Errors at this stage could compromise the entire prediction pipeline.

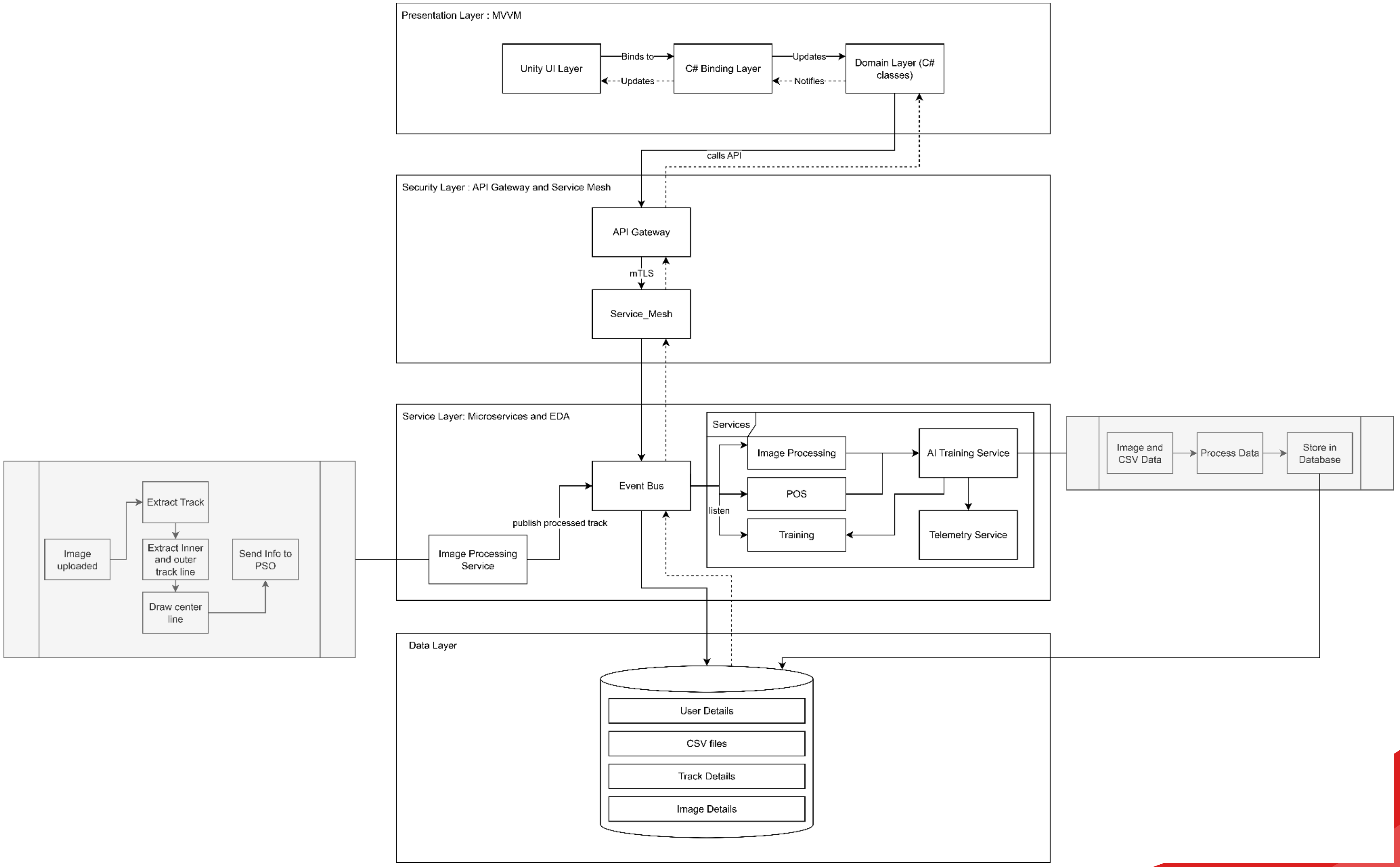
Computational Resource Demands

Reinforcement learning requires significant hardware resources, such as GPUs or cloud infrastructure, to train models effectively within reasonable timeframes. This may limit deployment on less powerful devices.

Focus on 2D Data for Initial Development

Due to time constraints, the system emphasizes 2D image data import and analysis rather than full 3D simulation. This prioritizes core functionality and simplifies early development.

Architectural Diagram


Presentation Layer : MVVM
Unity UI Layer
Binds to
Updates
C# Binding Layer
Updates
Notifies
Domain Layer (C# classes)
calls API
Security Layer : API Gateway and Service Mesh
API Gateway
mTLS
Service_Mesh
Service Layer: Microservices and EDA
Event Bus
publish processed track
Image Processing Service
Services
Image Processing
POS
Training
listen
AI Training Service
Telemetry Service
Image and CSV Data
Process Data
Store in Database
Extract Track
Image uploaded
Extract Inner and outer track line
Draw center line
Send Info to PSO
Data Layer
User Details
CSV files
Track Details
Image Details