

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 ($\leq 10\text{MB}$) in under 10 min.
- **NF1.2:** AI training will be completed for the system, so that simulations will run giving real-time feedback during optimization.

NF2: Security Requirements

- **NF2.1:** Sensitive user data (e.g: login credentials) will be stored using salted hashing (bcrypt).

- **NF2.2:** 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.

NF4: Usability Requirements

- **NF4.1:** The interface will be intuitive for non-technical users.
- **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 will require user confirmation.

NF5: Scalability Requirements

- **NF5.1:** GPU-accelerated training will dynamically allocate resources based on workload.

NF6: Compatibility Requirements

- **NF6.1:** The system will support Windows 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 ($\leq 10\text{MB}$).

NF7: Maintainability Requirements

- **NF7.1:** Code will be documented with API specs, inline comments, and version control (Git).

NF8: Cost & Resource Constraints

- **NF8.1:** Costs will not exceed R5000 (aligned with project budget).

Quality Requirements Testing (Non-Functional Testing)

NF1: Performance Requirements

- **NF1.1:** The system will process and analyse a racetrack image ($\leq 10\text{MB}$) in under 10 min.

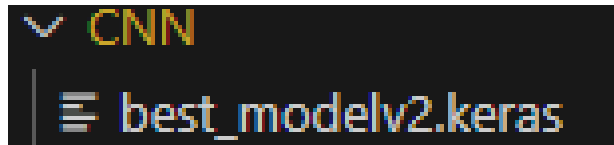
Has this Requirement been met?

Yes, this requirement is met. On average the system takes < 5 min once built. This can be seen once the system is used.

- **NF1.2:** AI training will be completed for the system, so that simulations will run giving real-time feedback during optimization.

Has this Requirement been met?

Yes, our AI has been trained before hand and this greatly improves the runtime of the system. Our CNN is saved to a .keras file that is used by the system when it is run.



NF2: Security Requirements

- **NF2.1:** Sensitive user data (e.g: login credentials) will be stored using salted hashing (bcrypt).

Has this Requirement been met?

Yes, we use salted hashing within our system to keep user data save once it is stored within our database.

```

1  const express = require('express');
2  const bcrypt = require('bcrypt');
3  const { ObjectId } = require('mongodb');
4
5  module.exports = function(db) {
6    const router = express.Router();
7
8    async function hashPassword(password) {
9      const saltRounds = 10;
10     return await bcrypt.hash(password, saltRounds);
11   }
12
13   async function comparePassword(password, hashedPassword) {
14     return await bcrypt.compare(password, hashedPassword);
15   }
16 }

```

- **NF2.2:** AI models and training data will be protected against unauthorized modification.

Has this Requirement been met?

Yes, our system has ben built with the AI already trained, so the user is unable to manipulate this. The data is also not uploaded to our database, so if the user

somehow managed to pollute their AI it would not affect anyone else using the system.

```
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from scipy import ndimage

class CNNTrackProcessor:
    def __init__(self, model_path="best_modelv2.keras"):
        self.model = load_model(model_path, compile=False)
        self.original_image = None
        self.track_mask = None
        self.track_boundaries = None
```

NF3: Reliability & Availability

- **NF3.1:** The system will maintain 95% uptime under normal operating conditions.

Has this Requirement been met?

Yes, our system has been deployed and is now downloadable to the user. Our API is currently hosted on an oracle VM.

The Website where users are able to download the application from is being hosted from our GitHub pages, so it will also be available to the users so long as our GitHub is active.



Build and deployment

Source

Deploy from a branch ▾

Branch

Your GitHub Pages site is currently being built from the /docs folder in the main branch. [Learn more about configuring the publishing source for your site.](#)

 main ▾  /docs ▾ Save

id	name	namespace	version	mode	pid	uptime	cpu	status	mem	user	watching
0	superlap_api	default	N/A	fork	90891	48D	4	online	0%	33.2mb	disabled

ubuntu@instance-20250804-2226:~\$ |

NF4: Usability Requirements

- **NF4.1:** The interface will be intuitive for non-technical users.

Has this Requirement been met?

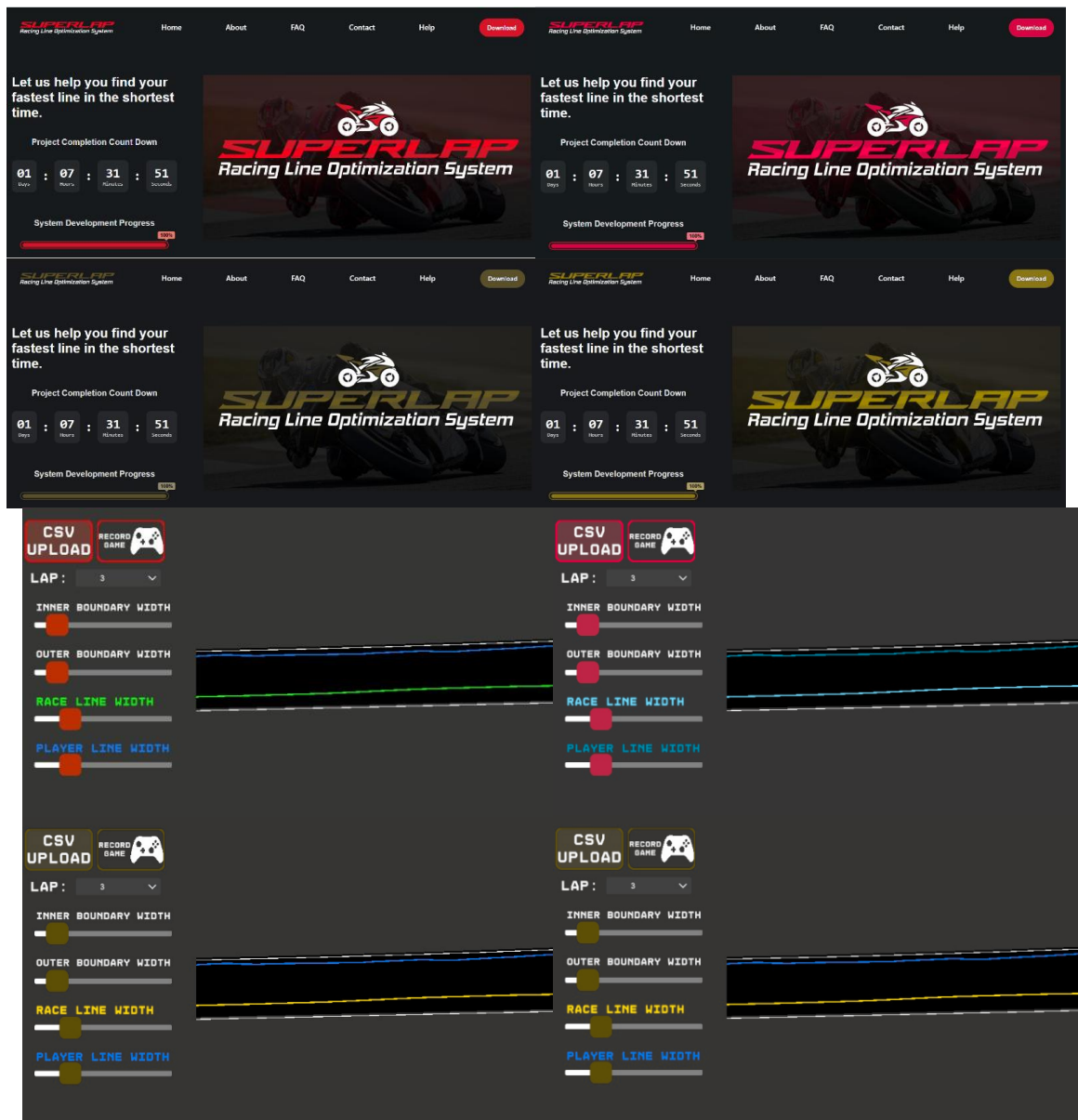
Yes, our interface is clearly laid out and easy to “play around with” to learn how to use the system. It is set up in an intuitive way so that if the user can easily navigate around the dashboard.

- **NF4.2:** Visualizations (racing line overlays, metrics) will adhere to colourblind-friendly palettes.

Has this Requirement been met?

Yes, we have a cohesive palate, but one that is easily recognisable in various formats making navigation easy for the users. The screenshots below showcase the website and track extractor in the actual application when seen through different types of colour-blindness:

Normal *Tritan (blue)*
Protan (red) *Deutan (green)*



NF4.3: The system will provide tooltips/guided tutorials for first-time users.

Has this Requirement been met?

Yes, once a user enters the system they can click on the little “?” at the top right and tool tips will appear for them on the screen to guide them.



- **NF4.4:** All critical actions will require user confirmation.

Has this Requirement been met?

We no longer have any critical actions within our system, such as update, delete or replace. So the system is safe from user error that may occur from something like this.

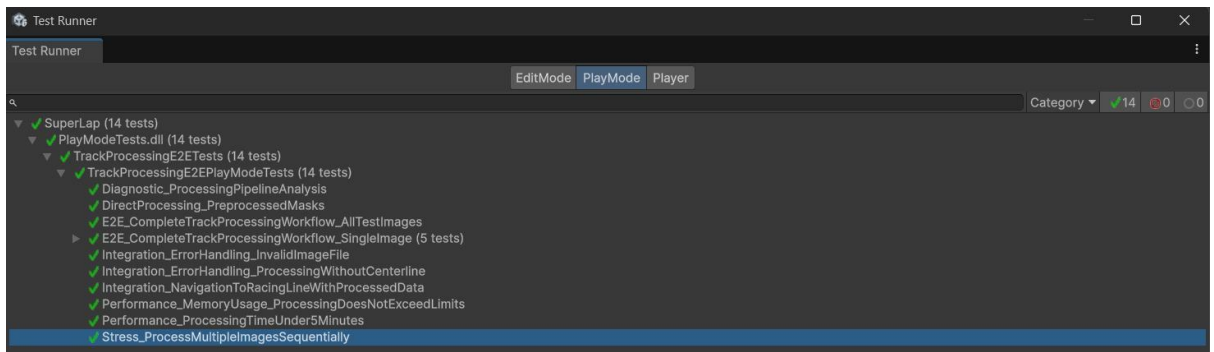
NF5: Scalability Requirements

- **NF5.1:** GPU-accelerated training will dynamically allocate resources based on workload.

Has this Requirement been met?

Yes, during training we utilized the GPU for the CNN image training to cut down on training time. Within our U-Net training we use the GPU.

There is no easy way to showcase this, but when we were using the CPU it would take approximately 3hr on 3000 images and once we started to use the GPU it took 10 min. We then increased the number of images so it now takes 3hr on 30 000 images, but it would have been substantially higher if we were using the CPU.

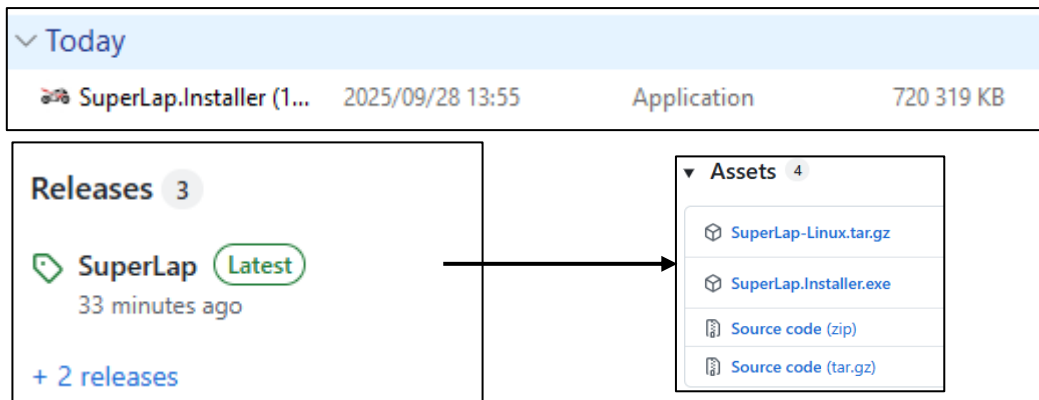


NF6: Compatibility Requirements

- **NF6.1:** The system will support Windows and Linux for desktop applications.

Has this Requirement been met?

Yes, we have a downloadable .exe file that will run on Windows for the system and we have a build that works on Linux as well.



NF6.2: Web-based access will be compatible with Chrome, Firefox, and Edge (latest versions).

Has this Requirement been met?

Yes, our website can be viewed through any browser.

Yes, currently we have only used R2 351 of our budget and the remaining budget is more than enough to complete the project.

We have used the budget on t-shirts, a Unity package and the MotoGP 18 game.

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).

<i>NF Category</i>	<i>Strategy</i>
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.

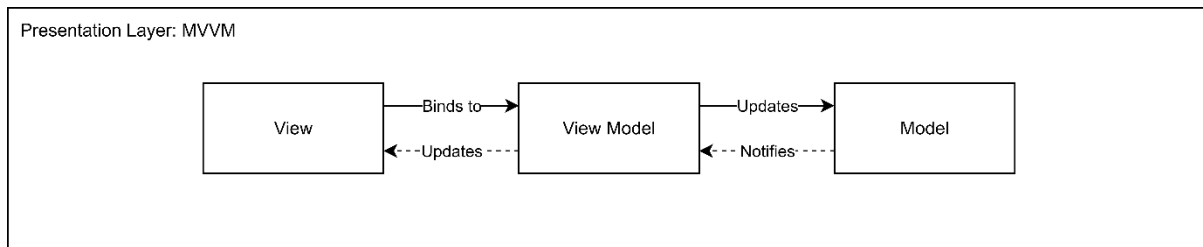
Where Used:

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

Event-Driven Architecture (EDA)

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

Why:



Repository Pattern for Data Layer

NF Category

Strategy

NF3: Reliability & Availability,
NF7: Maintainability, NF2:
Security

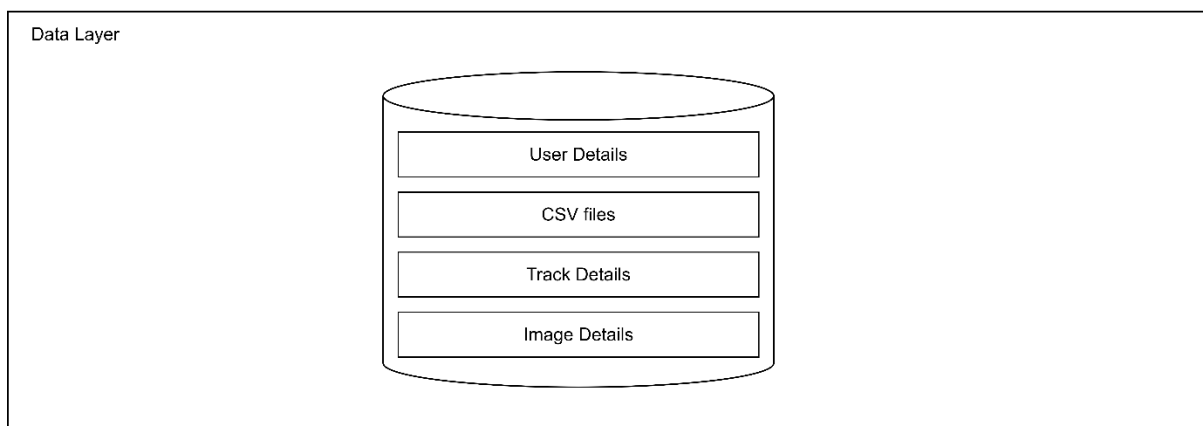
Abstract data access logic behind a repository layer to decouple business logic from persistence concerns.

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.

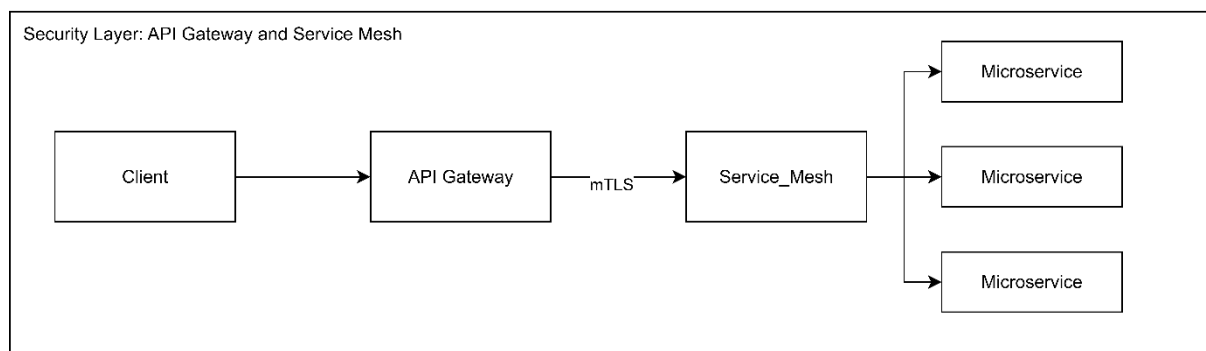
Why:

- 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

