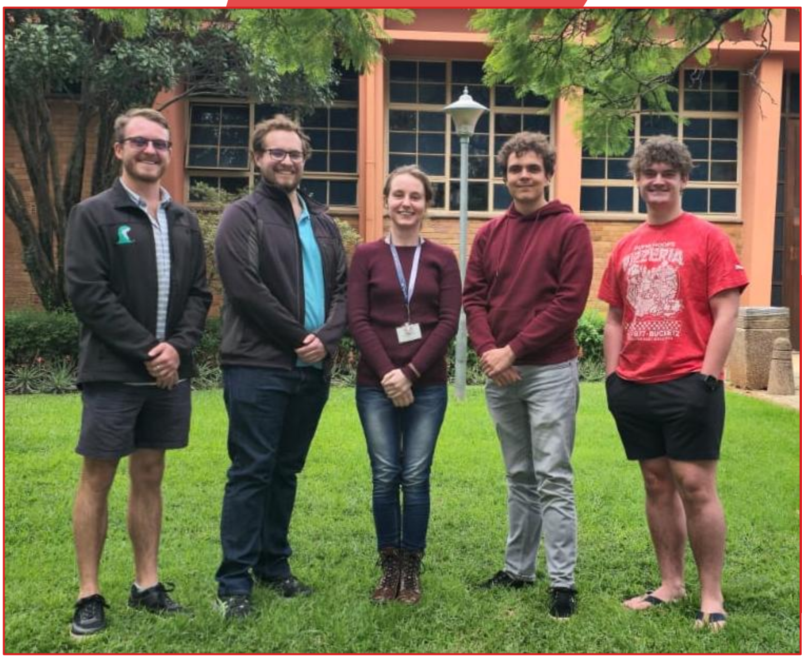
# SuperLap Racing Line Optimization System

EPI-USE




## Quintessential

Amber Ann Werner [u21457752]

Milan Kruger [u04948123]

Qwinton Knocklein [u21669849]

Sean van der Merwe [u22583387]

Simon van der Merwe [u04576617]

# Contents

# Introduction

**SuperLap Racing Line Optimization System** is an innovative AI-driven platform designed to help superbike riders identify the fastest possible racing line on a given racetrack. By combining Reinforcement Learning (RL) with Computer Vision, SuperLap analyses a top-down image of the track, learns optimal paths through thousands of simulations, and visually overlays the ideal racing line back onto the map. The system is built for accessibility – especially for students, amateur riders, and motorsport enthusiasts – eliminating the need for expensive telemetry systems or real-world trials. With a focus on usability, performance, and precision, SuperLap enables smarter, data-driven race training and performance optimization.

# User Characteristics

## Amateur & Hobbyist Racers

**Characteristics:**

- **Skill Level:** Novice to intermediate riders.
- **Goals:** Improve lap times, learn optimal racing lines, and understand track dynamics.
- **Technical Proficiency:** Basic (comfortable with apps but not deep technical knowledge).
- **Usage:**
    - Uploads track images from local circuits.
    - Uses AI-generated racing lines as training aids.
    - Compares different lines for self-improvement.
- **Motivation:** Cost-effective alternative to professional coaching/telemetry.

**Example:** A track-day rider at Kyalami Circuit who wants to shave seconds off their lap time.

## Motorsport Coaches & Instructors

**Characteristics:**

- **Skill Level:** Advanced (former/current racers).
- **Goals:** Teach students optimal racing strategies using AI insights.
- **Technical Proficiency:** Moderate (understands racing physics but not AI/ML).
- **Usage:**
    - Validates AI suggestions against their experience.
    - Generates visual training materials for students.

    - Compares different rider lines for debriefs.
- **Motivation:** Enhances coaching efficiency with data-backed insights.

**Example:** A riding instructor at a racing school who uses SuperLap to show students braking points.

### Sim Racing Enthusiasts

**Characteristics:**

- **Skill Level:** Varies (casual to competitive sim racers).
- **Goals:** Optimize virtual racing performance in games like *Assetto Corsa* or *Gran Turismo*.
- **Technical Proficiency:** High (comfortable with mods/data analysis).
- **Usage:**
    - Imports game track maps for AI analysis.
    - Compares SuperLap's line against in-game telemetry.
    - Shares optimized lines with sim racing communities.
- **Motivation:** Gain a competitive edge in online races.

**Example:** An iRacing league player who wants the perfect Monza line.

### Professional Racing Teams (Small/Privateer)

**Characteristics:**

- **Skill Level:** Expert (professional riders/engineers).
- **Goals:** Fine-tune bike setup and validate strategies.
- **Technical Proficiency:** High (understands AI, telemetry, and vehicle dynamics).
- **Usage:**
    - Cross-checks AI predictions with real-world data.

    - Tests "what-if" scenarios (e.g., wet vs. dry lines).
    - Integrates with existing telemetry tools (if API available).
- **Motivation:** Affordable alternative to high-end motorsport analytics.

**Example:** A privateer Moto3 team optimizing cornering lines on a budget.

### Engineering & Motorsport Students

**Characteristics:**

- **Skill Level:** Academic (learning racing dynamics/AI).
- **Goals:** Study racing line theory, RL applications, and vehicle physics.
- **Technical Proficiency:** Medium (some coding/math knowledge).
- **Usage:**
    - Experiments with different AI models (e.g., DQN vs. PPO).
    - Validates academic theories against SuperLap's simulations.
- **Motivation:** Research and project-based learning.

**Example:** A mechanical engineering student analyzing Suzuka's "S-curves" for a thesis.

## User Stories

### Core User Stories (Functionality & User Experience)

1. As a rider, I want to upload a top-down image of my racetrack so that the system can analyse it for racing line optimization.
2. As a user, I want to see the AI-generated optimal racing line overlaid on the track so that I can compare it to my existing racing strategy.
3. As a motorsport enthusiast, I want the system to simulate multiple racing lines using reinforcement learning so I can see which line performs the best under different conditions.
4. As a rider, I want to compare my recorded lap times with the AI's optimal lap time so I can identify areas for improvement.

5. As a user, I want the app to visually simulate the lap with a bike animation in Unity so I can better understand the racing line's logic.
6. As a beginner racer, I want simple guidance such as "brake here" or "turn in here" based on the AI's racing line, so I can apply it in real life.
7. As a user, I want to toggle between 2D and 3D views of the track to better analyse racing lines.
8. As a data scientist, I want to inspect the AI's learning process and convergence so I can understand and tweak the training algorithm.
9. As a developer, I want to train the AI model using data from racing games like Gran Turismo or F1 202x to simulate high-fidelity data scenarios.
10. As a researcher, I want to see how the AI adapts to different weather conditions (wet/dry), so I can evaluate its robustness.
11. As a developer, I want to integrate simulated sensor data (like speed and grip levels) so that the AI racing line feels more realistic.

## Visualization & Comparison Stories

1. As a racer, I want to switch between different racing line strategies (e.g., heuristic vs. AI-optimized) so I can decide which one is best suited for my skill level.
2. As a user, I want the ability to scrub through a lap simulation to analyze key moments like braking zones and apex points.
3. As a coach, I want to export performance data and AI-generated lines for further analysis outside the app (e.g., in Excel or MATLAB).

## Interface & User Experience Stories

1. As a mobile user, I want to access racing line data from my smartphone to review my training between sessions.
2. As a VR/AR user, I want to see the AI racing line overlaid on my real track via augmented reality so I can train on the spot.
3. As a casual user, I want a guided tutorial on how to interpret AI racing lines and use the app effectively.
4. As a user, I want to switch between light and dark modes for better visibility depending on the time of day.

### Backend & Performance Stories

1. As a backend developer, I want the system to efficiently process large track images to reduce wait time for the user.
2. As a developer, I want cloud integration so that users can save, retrieve, and compare racing sessions across devices.
3. As a power user, I want to configure AI training parameters (e.g., epsilon decay, learning rate) for custom experiments.
4. As a team, we want to store training sessions and model states securely in a database so that progress isn't lost between runs.

### Gamification & Community Stories

1. As a user, I want to share my best lap and AI-optimized strategy with others to compare and compete.
2. As a community member, I want to vote on or comment on AI racing lines that others have shared to collaborate and learn.
3. As a racer, I want leaderboards showing AI lap times vs. user lap times to motivate improvement.

## Use Case

# Functional Requirements

**R1: Track Image Processing**

**R1.1: Binary Conversion**

- The system shall convert top-down racetrack images into binary maps for AI analysis.

**R1.2: Boundary Detection**

- The system shall accurately detect and distinguish track boundaries from off-track areas.

**R2: Racing Line Optimization**

**R2.1: Reinforcement Learning**

- The system shall apply Reinforcement Learning (RL) to simulate and refine racing lines.

**R2.2: Path Evaluation**

- The system shall iterate through multiple paths to determine the fastest racing line.

**R3: AI Training and Simulation**

**R3.1: Training Data Input**

- The system shall train AI agents using simulated or game-based datasets.

**R3.2: Physics Modelling**

- The system shall incorporate physics-based models to ensure realistic performance.

**R4: Result Visualization**

**R4.1: Line Overlay**

- The system shall overlay the optimized racing line on the track image.

**R4.2: Performance Metrics**

- The system shall display key performance indicators such as estimated lap time and braking zones.

**R5: Infrastructure Integration**

**R5.1: Computation Support**

- The system shall support sufficient computational resources (e.g., GPU) for RL training.

### R5.2: Cloud Compatibility

- The system shall optionally integrate with cloud services to allow for scalability and extended computation.

## R6: Adaptive AI Strategies

### R6.1: Dynamic Track Conditions

- The system shall adjust racing lines based on simulated track conditions (e.g., wet/dry surfaces).

## R7: Enhanced Visualization & User Interaction

### R7.1: Interactive 3D Simulation (Optional)

- The system shall provide a 3D interactive visualization of the track and optimized racing line.

### R7.2: Dynamic Line Adjustment

- The system shall allow users to manually adjust the racing line and re-simulate performance.

### R7.3: Heatmap of Speed/Acceleration Zones

- The system shall generate a speed/acceleration heatmap overlay for performance analysis.
- The system shall allow users to provide feedback on AI-generated lines for iterative improvement.

# Non-Functional Requirements

### NF1: Performance Requirements

- **NF1.1:** The system shall process and analyze a racetrack image (≤10MB) in under 5 seconds.
- **NF1.2:** AI training simulations shall run at ≥30 FPS for real-time feedback during optimization.
- **NF1.3:** Lap time predictions shall be computed within 1 second after track processing.
- **NF1.4:** The system shall support at least 50 concurrent users in cloud-based mode.

### NF2: Security Requirements

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

### NF3: Reliability & Availability

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

### NF4: Usability Requirements

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

### NF5: Scalability Requirements

- **NF5.1:** The system shall scale horizontally to support up to 10,000 simulations/day via cloud resources.
- **NF5.2:** Modular architecture shall allow integration of new physics models or RL algorithms without major refactoring.
- **NF5.3:** GPU-accelerated training shall dynamically allocate resources based on workload.

### NF6: Compatibility Requirements

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

### NF7: Maintainability Requirements

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

### NF8: Cost & Resource Constraints

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

# Architectural Requirements

## High-Level Architectural Style

**Requirement:**

- **AR1.1:** The system shall follow a **microservices architecture** for modularity, with separate services for:
  - Image processing (OpenCV/Python)
  - Reinforcement Learning (RL) training (PyTorch/TensorFlow)
  - Visualization (Web-based frontend)
  - User management (Auth0/Custom JWT)
- **AR1.2: Event-driven communication** (e.g., Kafka/RabbitMQ) shall connect services to handle async tasks (e.g., RL training completion triggers visualization updates).

**Justification:**

- Decouples resource-intensive tasks (e.g., RL training) from user-facing components.
- Enables independent scaling of services.

## Core Components & Interactions

**Requirement:**

- **AR2.1:** The system shall consist of:
  - **Track Processing Service:**
    - Input: Top-down track image (JPEG/PNG).
    - Output: Binary map + detected boundaries (stored in Redis for fast retrieval).
  - **RL Training Service:**

        - Input: Binary map + physics parameters (e.g., tire grip, bike specs).
        - Output: Optimized racing line (stored in PostgreSQL with versioning).
    - **Simulation Engine:**
        - Physics model (e.g., PyBullet/Custom) for realistic dynamics.
    - **API Gateway:**
        - REST/GraphQL endpoints for frontend communication.
    - **Frontend:**
        - Web-based (React/Three.js for 3D) + optional desktop (Electron).

**AR2.2:** Data flow shall adhere to:

User Upload → Track Processing → RL Training → Simulation → Visualization.

## Data Management

**Requirement:**

- **AR3.1:** Track images and metadata shall be stored in **AWS S3/Blob Storage** (cost-effective for large files).
- **AR3.2:** Simulation results (racing lines, lap times) shall use **PostgreSQL** (structured queries) + **Redis** (caching).
- **AR3.3:** Training data from games/simulators shall be ingested via **parquet files** (columnar storage for efficiency).

## Integration Requirements

**Requirement:**

- **AR4.1:** The system shall support APIs for:
    - **Racing Games** (e.g., Assetto Corsa via UDP/Telemetry APIs).
    - **Cloud GPU Providers** (e.g., AWS SageMaker for distributed RL training).

- **AR4.2:** Third-party auth (Google/OAuth) shall integrate via **Auth0** or **Firebase**.

### Scalability & Performance

**Requirement:**

- **AR5.1:** RL training shall scale horizontally using **Kubernetes** (auto-scaling GPU nodes).
- **AR5.2:** Image processing shall offload to **AWS Lambda** during peak loads.
- **AR5.3:** Frontend shall use **CDN caching** (e.g., Cloudflare) for static assets.

### Fault Tolerance & Recovery

**Requirement:**

- **AR6.1:** Training jobs shall checkpoint progress **every 15 minutes** (prevent data loss).
- **AR6.2:** Database failover shall be automated (PostgreSQL replica in standby mode).
- **AR6.3:** User uploads shall retry **3 times** before error reporting.

### Security Architecture

**Requirement:**

- **AR7.1:** Zero-trust model:
  - **JWT tokens** for API auth.
  - **VPC isolation** for training workloads.
- **AR7.2:** Data encryption:
  - **At rest** (AES-256 for S3/PostgreSQL).
  - **In transit** (HTTPS/mTLS for microservices).

### Deployment & DevOps

**Requirement:**

- **AR8.1:** Infrastructure-as-Code (IaC) via **Terraform/Ansible**.
- **AR8.2:** CI/CD pipeline (GitHub Actions/Jenkins) with:
    - **Testing:** Unit tests (PyTest), integration tests (Selenium).
    - **Rollback:** Automated if error rate >5% in canary deployments.

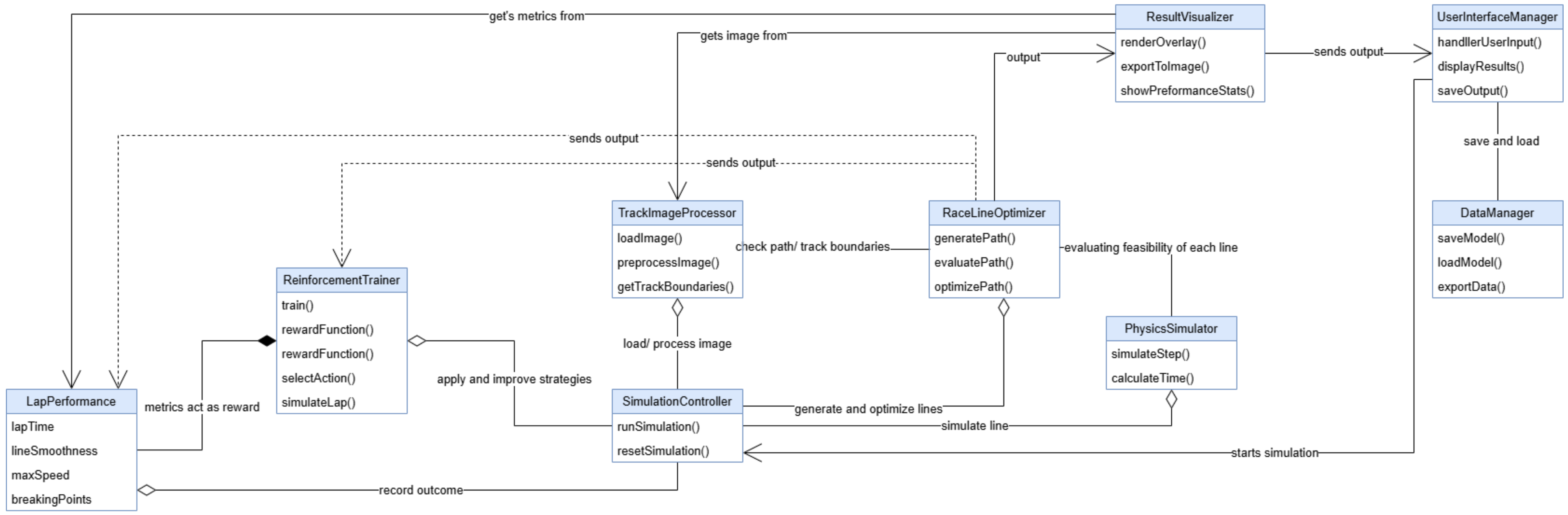
## Cross-Cutting Concerns

**Requirement:**

- **AR9.1:** Observability:
    - **Logging:** ELK Stack (Elasticsearch, Logstash, Kibana).
    - **Monitoring:** Prometheus/Grafana for GPU usage, API latency.
- **AR9.2:** Compliance with **GDPR** for user data deletion requests.

# Architecture Diagram

# Class Diagram



## Deployment Diagram

# Installation Manual

[Needs to be created]

# Technical Installation Manual

[Needs to be created]

# User Manual

[Needs to be created]

# Machine Learning Specification

[Needs to be created]

# API Documentation

[Needs to be created]

# Coding Standards

[Needs to be created]

# Testing Policy

## Testing Scope & Levels

| Level | Focus | Tools/Methods | Owners |
|---|---|---|---|
| **Unit Testing** | Individual functions (e.g., track image processing, RL reward function). | Pytest (Python), JUnit (Java). | Developers |
| **Integration Testing** | Interaction between services (e.g., track processor → RL engine). | Postman, Jest (API tests), Selenium (UI flows). | QA Team |
| **System Testing** | End-to-end workflows (e.g., upload image → simulate → visualize). | Cypress, Robot Framework. | QA Team |
| **Performance Testing** | Scalability (e.g., 50 concurrent users), RL training speed. | Locust (load testing), NVIDIA Nsight (GPU profiling). | DevOps |
| **Security Testing** | Data encryption, auth vulnerabilities. | OWASP ZAP, SonarQube. | Security Team |
| **User Acceptance (UAT)** | Real-world usability (by target users). | Beta releases, A/B testing. | Product Team |

## Testing Types & Frequency

| Test Type | Description | Frequency |
|---|---|---|
| **Automated Regression** | Validate existing features after updates. | On every Git commit (CI/CD). |
| **Manual Exploratory** | Unscripted UX/edge-case testing. | Before major releases. |
| **Physics Validation** | Compare AI racing lines against known heuristics (e.g., apex accuracy). | Per RL model update. |
| **Hardware Compatibility** | GPU/CPU performance benchmarks. | Quarterly. |

## Entry & Exit Criteria

**Entry Criteria (Tests Start When):**

- Requirements are documented (e.g., FR/NFRs).
- Code is merged to the test branch.
- Test environment mirrors production (GPU-enabled).

**Exit Criteria (Tests Pass When):**

- **Unit/Integration:** ≥90% code coverage (measured via Coveralls).
- **Performance:** <2s response time for track processing; RL training FPS ≥30.
- **Security:** Zero critical OWASP vulnerabilities.
- **UAT:** ≥80% positive feedback from beta testers.

## Defect Management

- **Severity Levels:**

  - **Critical** (Crash/data loss): Fixed within 24h.
  - **Major** (Feature failure): Fixed in next sprint.
  - **Minor** (UI glitch): Backlogged for prioritization.
- **Tracking:** Jira/Linear with labels (bug, reproducible, blocker).

## Environments

| Environment | Purpose | Access |
|---|---|---|
| **Development** | Feature development. | Engineers only. |
| **Staging** | Pre-production (mirrors prod). | QA/Product Team. |
| **Production** | Live user-facing system. | Automated deployments only. |

## Test Data Management

- **Realistic Datasets:**
  - 10+ sample tracks (F1, MotoGP circuits).
  - Synthetic data from racing sims (Assetto Corsa).
- **Anonymization:** User-uploaded tracks scrubbed of metadata.

## Compliance & Reporting

- **Audits:** Monthly test coverage/review meetings.
- **Reports:** Dashboards (Grafana) for:
  - Test pass/fail rates.
  - Performance trends (e.g., lap time prediction accuracy).

## Policy Exceptions

- **Emergency Fixes:** Hotfixes may bypass some tests but require:

- Post-deployment regression testing.
- Retrospective review.

# Contributing

## Project Manager

### Amber Werner

[list contributions]

## Backend Developers

### Qwinton Knocklein

[list contributions]

### Sean van der Merwe

[list contributions]

## Front End Developers

### Simon van der Merwe

[list contributions]

### Milan Kruger

[list contributions]