TeLIX - real-time race data overlay and cloud-powered insights, your personal race engineer.

James Ash, ashj@wit.edu

Cole Fitzgerald, fitzgeraldc7@wit.edu

Version 1

GitHub Repository: https://github.com/TeLIX/TeLIX

May 19, 2025

Contents

1 Introduction

1.1 Purpose

The purpose of this project is to serve the client with an all-purpose SaaS (Software as a Service) that creates a real-time HUD (Heads up Display) overlay for real-time telemetry and analytics for a racing simulation game called iRacing. iRacing is a very popular racing simulation game that is used by professional real and simulation drivers. iRacing is branded as an ultra-realistic simulation device to perfect performance for drivers even when they do not have access to a track or car. Telemetry is the process of collecting data from remote locations and transmitting it to a central location for monitoring and analytics. Our goal is to create an all-in-one solution for real-time monitoring and analytics to improve driver performance over time.

1.2 Scope

The scope of this software includes the overlay software, responsible for recording telemetry data and sending it to our database and our dashboard which utilizes collected data to display statistics and suggest improvements to users.

1.3 Definitions, Acronyms, Abbreviations

- iRacing - A subscription-based simulation racing service.
- SaaS - Software as a Service.

2 Project Overview

2.1 Description

TeLIX is a telemetry data acquisition and visualization platform built specifically for iRacing. It consists of two main components:

- A real-time Heads-Up Display (HUD) overlay, which displays current telemetry data to the driver in-game.
- A web dashboard, which provides deep insights into performance metrics after a session.

The system captures live telemetry data (e.g., speed, RPM, gear, throttle, brake, G-force, lap times), stores it, and serves it to both components through a central API and database.

2.2 Impact & Market Potential

TeLIX targets simulation racers, competitive esports drivers, and coaching organizations. The project addresses a growing need in sim-racing for accessible, insightful analytics and real-time driving feedback. Potential future expansion could include:

- Support for other simulators (Assetto Corsa, rFactor 2)

- Community sharing of telemetry data
- Machine learning-based feedback

2.3 Objectives

1. Real-time telemetry capture from iRacing
2. Transparent and non-intrusive overlay for key metrics
3. Persist telemetry data in a structured database
4. Dashboard to visualize session statistics
5. Scalable backend for future expansion

3 System Architecture

3.1 Overlay Architecture

The overlay architecture shall be a native Windows application that connects to the iRacing shared memory API. The transparent window shall be rendered with ImGUI displaying selected metrics in real time with minimal delay.

3.2 Dashboard/API Architecture

A web application (React.js) will serve as the user dashboard and will handle the client. A backend (Express.js) will handle all API and Database calls from the dashboard and the Overlay software.

3.3 Network Architecture

Data will be streamed from the Overlay software to our Express API and data will be stored in our PostgreSQL database.

3.4 Testing

Automated unit tests will validate overlay logic, API data integrity, and dashboard render logic. Integration testing ensures end-to-end flow from iRacing to the web dashboard. Manual testing will include QA under race conditions.

4 Components

4.1 Overlay Components

- Telemetry Reader / iRacing SDK
- Overlay Renderer
- API Client

4.2 Dashboard/API Components

- Authentication API Module
- Data API Module
- Dashboard Module
- Static Page Module
- API Client

4.3 External API's

- iRacing SDK

5 Software Design

5.1 Languages

- Overlay: C# (.NET)
- API: TypeScript (Express.js)
- Dashboard: TypeScript (React.js)
- Database: PostgreSQL

5.2 Frameworks/Tech

- Overlay: ImGUI.NET, iRacing SDK
- API: Express.js, Docker
- Dashboard: React.js, Chart.js, Docker

5.3 API's

- REST API for telemetry submission and querying.
- Auth API for user authentication.

5.4 Database Design

5.4.1 User Information

*Include UML diagrams for a User ”class” in db.

5.4.2 Race/Lap Data

*Include UML diagrams needed for a race, a lap, a car, etc.

6 Requirements

6.1 Functional Requirements

6.1.1 Overlay

6.1.2 Dashboard/API

6.2 Non-Functional Requirements

6.2.1 Overlay

6.2.2 Dashboard/API

7 Diagrams

7.1 Class Diagrams

7.1.1 Overlay Classes

7.1.2 Dashboard/API Classes & Interfaces

7.2 Use Case Diagrams - Dashboard/API

7.3 Database Relations

7.4 UI/UX Mockups

8 Implementation Plan

8.1 Milestones

8.2 Timeline

9 Project Goals

9.1 Course Project Goals

9.2 EOL/Commercial Extension Goals

10 Appendices

This section is optional. Appendices may be included, either directly or by reference, to provide supporting details that could aid in the understanding of the Software Design Document.

11 References

References