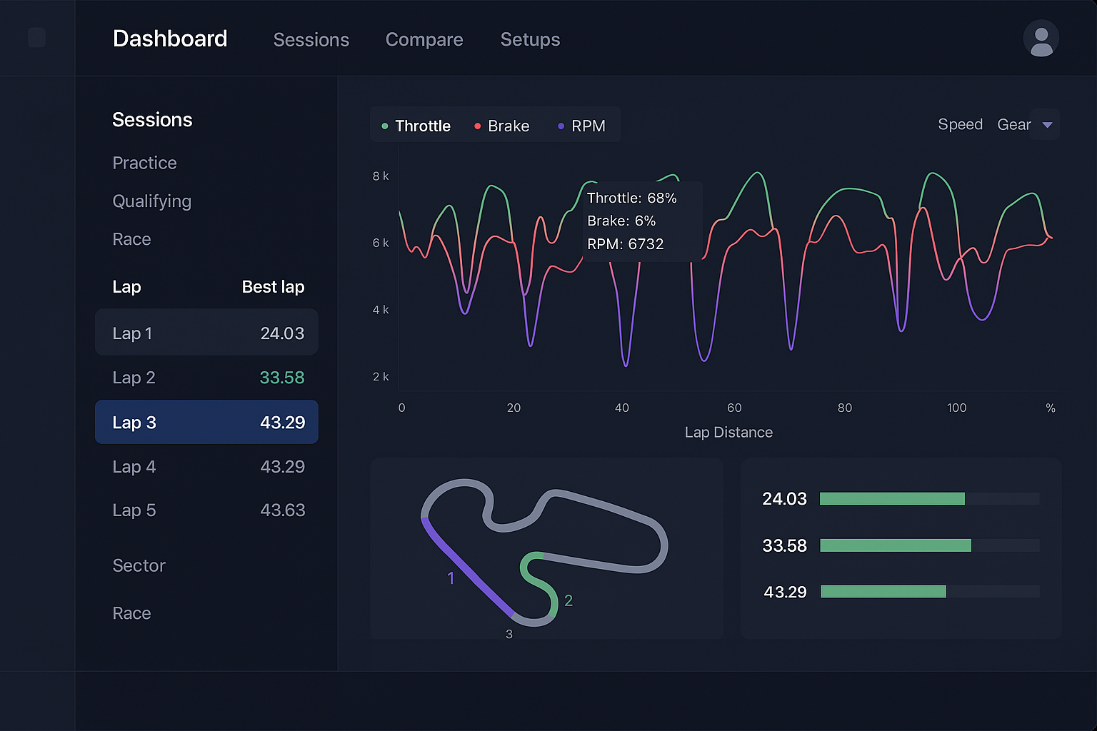
NegDelta (name not confirmed)



That’s a general look of it. Definitely going to stray from this a little bit, theming is about right, navigation on the left, lap comparisons, sessions, and setups available.

# Data Structures

Laps are stored in a stint, which is stored in a session.

* Each lap is pretty self explanatory. Once the driver crosses the start/finish the lap starts, and once they cross it again that lap ends, and the next lap starts immediately. However, a lap should NOT start when the driver is leaving the pits. No point in recording that one since the first sector is gone. A lap should also not be recorded when entering the pits.
  + Laps store list of telemetry points (throttle, brake, steering, timestamp), list of position points (x, y, z, timestamp), lap time, sector times (not exhaustive, there may be more)
  + Events? Such as oversteer and understeer events and maybe crash events
* Each stint starts when the driver leaves the pits, and it ends when the driver enters the pits by either driving to pitlane or being towed.
  + All things the stint will hold
    - Id
    - Fastest lap time
    - Laps
    - Lap count
  + This could result in lots of stints if the driver is crashing a lot in practice sessions.
  + Stints store a list of laps
  + Stints will also have the fastest lap of that specific stint
    - Sessions will also hold the fastest lap of the entire session.
* Each session starts once the driver joins a session of any type. It ends when the driver leaves the session they are currently in.
  + All things the session will hold
    - Id
    - Fastest lap time
    - Car name
    - Track name
    - Session Type
    - Stints
    - Stint count
    - Lap count
  + Sessions can take on several different types. They are as follows
    - Race, practice, qualifying, time attack.
  + Session types will likely be determined by the SessionType found in the yaml string of the ibt file.
  + Race sessions will hold practice, qualifying and the race session.
    - A race session should be able to hold each of the other session types (except time attack)
      * Composite pattern could work here
  + Sessions will also have the fastest lap time for that session.

## Tracks

I want to implement tracks at some point. It would be really cool to have a dynamic track map that moves much like the telemetry graphs do, however they follow the line that the driver took throughout the lap. I’m not entirely sure how to implement something like this, but I’m sure good ole chat gippity will give me hand.

In order to have this track object I’ll need to have a set of points that correspond to the track layout. I’m not sure how to get this data, but I’ll find it.

Tracks will also have to be stored in a database or a Json file.

## SessionBuilder

The session builder creates all the sessions. A builder instance will be created, then a session will be created with that builder.

### Issues

Having an issue where I need to clarify when a stint really needs to be made. Let’s say someone is just practicing and they’re crashing a lot. I need a way to make sure that a bunch of stints aren’t being created for each new lap a driver goes out on. If a stint is created every single time a driver leaves the pits, then there’s going to be a ton of pointless stints that only hold partial lap data. That’s just not efficient or really necessary. There has to be some way to check that the stints are valid.

Maybe I could have a function that checks if the stints have more than so many laps. If so then leave that stint alone. However, if the number of laps in a stint is less than some threshold, I can create a “failed stint” or something that holds all the “crash” laps or something. Basically this stint will be filled with all the laps that are inside of a stint that has less than a specific number of laps.

* Consensus:
  + Add a List<Lap> Junkyard to Session. This is where the laps that were part of a small stint go. Might need to also add a method to the builder for when crashes are detected. This could be like AddCrashLap(Lap l) where it would add it straight to the junkyard laps.

# Data Visualization

I want to visualize telemetry data. The main dashboard (lap analysis) will have the telemetry for brake, throttle, and steering. The dashboard should also be expandable. Meaning, the dashboard should be able to hold either just throttle position, or hold throttle and brake and steering and line what have you. If you don’t want to see steering, go ahead and uncheck it. Basically there should be a “view” tab or something that allows you to modify the dashboard to be however you want.

* Could have a dashboard “layout” so every time you open your dashboard to view a lap it’s already how you left it.
* Could possibly save multiple layouts for different types of analyzation.
* Name the layouts
* Each telemetry component will be inside a telemetry group of some kind this way there can be shared information between them.

## Components

Likely going to be using plotly for the data vis. The throttle, brake, and steering angle will be a typical chart that has panning abilities. Also, the center of the plot will be like the “timestamp” and the other charts should synchronize with the movement.

* As the user moves a chart up and down the time of the lap, the other charts should follow suit and move so that the same timestamp is always in the middle.
  + I’m not sure how this will be done, maybe by having a timestamp that’s currently “selected” and then navigating to that timestamp in real time within the other charts. I’ll need to investigate plotly and see if that’s possible (automated chart navigation)
* Drop downs on the side of the screen for session, stint, lap selection. Maybe an overarching session is selected, and then there are the stints which have dropdowns for the laps.

## Lap Comparisons

Should be able to compare two laps. This could be like a specific dashboard that’s for lap comparisons. This could be accessed by a button click or a drag and drop technique.

* Lap data could be stored in the cloud, and any friends or randos could look at your lap time to compare with their own.
* Laps should also be able to be removed from the comparison. An easy-to-understand user interface for changing laps for comparison.

# Theming

Allow ability for light and dark mode? Idk Dark theme goes pretty hard. Don’t need any plebs requesting light mode, though. I may just give it to them.

# Navigation

Navigation for the overarching app should be on the top. I like the idea of having different tabs available for the different pages.

Then once inside of that page, if there’s a need for it, have navigation to the left of the page. By navigation I mean like drop downs, kind of like a folder structure or the solution explorer in visual studio. This wouldn’t be as much navigation as it would be selecting a lap to look at.

# Storage

I think it would be great to use a relational database for this project. Maybe even tie it into the cloud to get some experience with it. This way setups could be saved and shared across the internet. As well as lap comparisons.

* Laps will be stored within stints, which are stored within sessions.
  + Sessions will be stored in the relational database.

## SessionDbContext

The SessionDbContext object is an object that holds EntityFramework (EF) compatible objects. These objects, in my case, are the Sessions, Stints, and Laps. Whether or not I want to make the TelemetryPoints, and other point objects, also EF compatible will determine on my querying needs.

## SessionStorageService (SSS)

The *SessionStorageService* will save and load sessions. This will be used after a session has been created, filled, and exited by the user. The *SSS* will save that session for later analysis. The actual creation of sessions will be done by a different object.

Once a session has been saved, it should then be able to be loaded into the UI. The SSS will be the object to load up the session object from the database via *SessionDbContext*. It will return the session object to the caller.

Might be worth looking into a cache setup so that previously queried sessions are quickly available.

The *SSS* is a interface provider (wrapper) of the *SessionDbContext* object. The *SessionDbContext* object provides an interface for interacting with the database, but the *SSS* will make it a bit easier for us in the long run.

# Authentication

People would need to sign in with an account. You can add friends that you can share setups with, compare lap times, etc.

* There might be some sort of copyright issue that I would be putting in motion if people share paid setups. I should investigate that.