f1nsight Systems Report

Gianfranco Manieli

Contents

1. Identifying and defining

1.1. Define and analyse problem requirements

Problem context

Students **analyse** the problem by **describing** each of its individual components and **explaining** how each of these components contribute to the problem needing resolution.

F1 fans have a fragmented experience when it comes to finding easy-to-access, aesthetically pleasing race data, performance insights, and news.

These key informations are spread across multiple platforms, making it time-consuming to scrape data and news from so many different sources. There is no central hub that offers personalised analytics, desktop notifications and interactive visualisations in one place.

This lack of integration limits fan engagement and creates an opportunity for a unified solution like f1nsight.

Needs and opportunities

Students **describe** the needs of the new system to be built based on the problem context and using the table given below.

Need	Description
1. Central dashboard for race data	The dashboard will combine multiple data sources, saving time and frustration from having to search multiple places
2. API news integration + notifications	News updates will help keep fans updated within one place
3. Driver insights	Help fans understand trends and form within teams

Boundaries

Students **analyse** any limitations or boundaries in which this new system will need to operate. Boundaries can include but are not limited to: hardware, operating systems, security concerns etc.

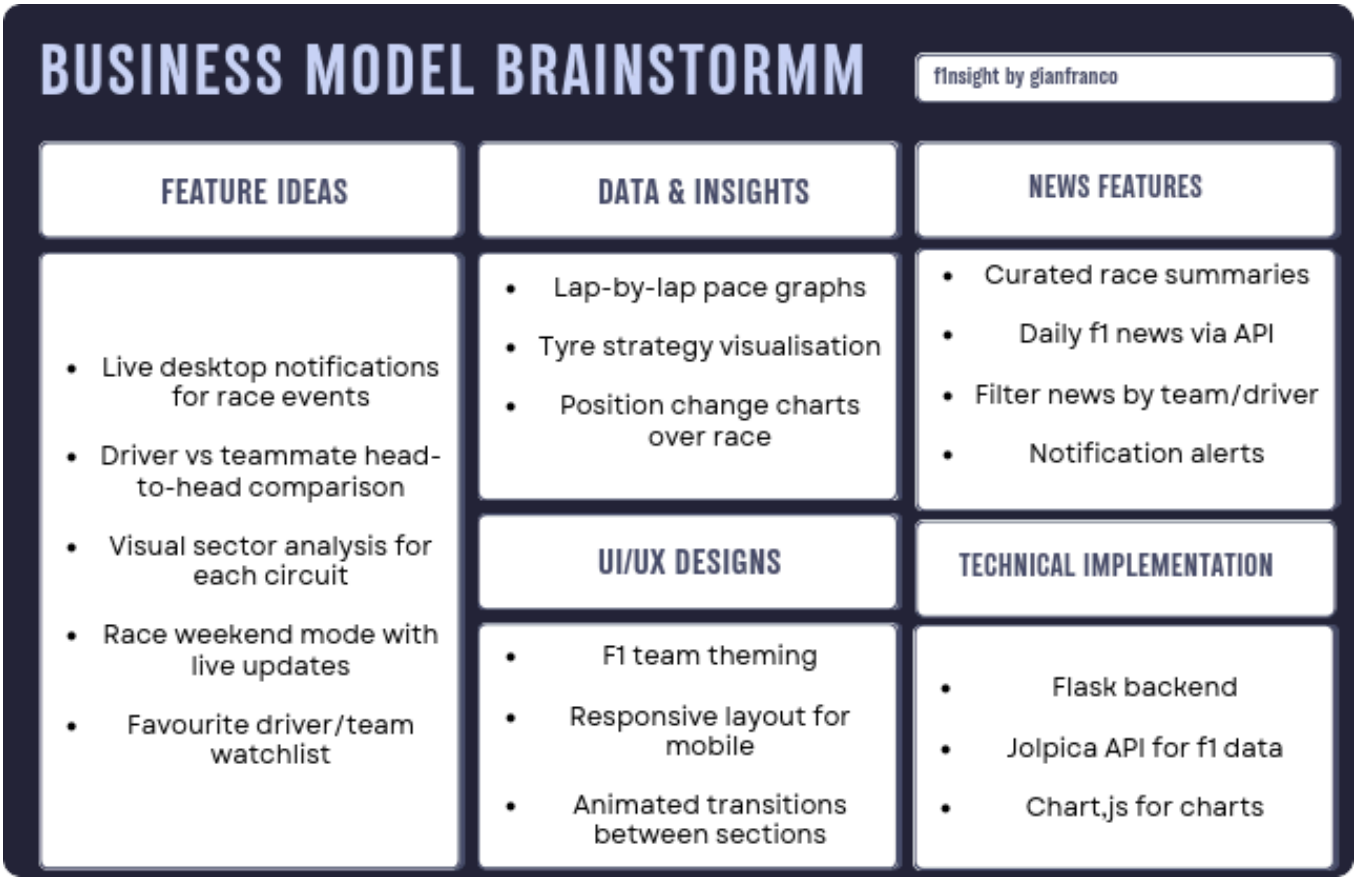
f1nsight must operate as a browser-based platform that is compatible across multiple devices and operating systems. It relies on 3rd party APIs with rate limits, requires secure logins, and must run efficiently so it can be used on standard consumer hardware. The project is budget-conscious, limited use of any premium infrastructure or tools.

Notification system must be tested on multiple devices and browsers.

1.2. Tools to develop ideas and generate solutions

Application of appropriate software development tools

Students **apply** appropriate tools such as brainstorms, mind-maps, storyboards and prototypes.



Storyboard

1.3. Implementation method

Students **explain** the applicability of the implementation method for the current project. These methods are normally direct, phased, parallel, or pilot.

f1nsight follows an phased implementation method, well suited to iterative software development. Each stage (sprint) will be tested and finalised before moving onto the next

Each sprint will deliver a working portion of the system

- Sprint 1 will focus on identifying requirements, setting up the backend, and building core pages (e.g. login, standings).
- Sprint 2 will handle research and planning, adding more complex features like news scraping, compare graphs, and UI refinements.
- Sprint 3 will be focused on production and testing, implementing HTTPS, polishing the theme, and debugging.
-

This phased method was appropriate because:

- It allowed for progressive development, with feedback after each sprint.

- It reduced risk, since bugs or failures were contained within small phases.

- It ensured that by the final sprint, the system was both functional and tested.

Each stage will be documented, tracked VIA Git branches, and conclude with a working system state. This approach supports quality control and is manageable for a solo project.

1.4. Financial feasibility

Students are to **conduct** a financial feasibility study, including producing an opening-day balance sheet, to assess whether their application is financially viable.

SWOT Analysis

Study	Go or No Go?	Assessment and evidence
Market feasibility	GO	Growing f1 fanbase with high interest in data content. No major free alternatives.
Cost of development	GO	use of flask and open-source tools make total development costs low $3000 for actual codebase
Cost of ownership	GO	Ongoing monthly costs for the website are relatively low excluding labour, it sits at $45/month
Income potential	GO	ads, donations and potential for premium features will cover monthly costs and could provide profit
Future expansion opportunities	GO	can expand data archive and website to f2, motogp and potentially support fantasy f1. potential for mobile app, integrated fantasyf1 analysis

f1nsight Balance Sheet [AUD $]	2025	2026	2027
Assets			
Current assets:			
Labour	5,000	1,000	1,000
Hosting	350	350	350
Prepaid API subscription	100	100	100
Maintenance	100	100	100
Total current assets	**5,550**	**1,550**	**1,550**
Fixed Assets:			
Domain name (3 Years)	100		
Development computer	1,000		
Application codebase	3,000		
Total Assets	**9,650**	**1,550**	**1,550**
Liabilities			
Current liabilities:			
Credit Card Payable	750	250	250
Personal Loan	550	550	550
Total Liabilities	**1,300**	**800**	**800**
Equity			
External Investment	8,550	750	750
Total Liabilities & Shareholder's Equity	**9,850**	**1,550**	**1,550**
Check	$ 8,550.00	$ 750.00	$ 750.00

2. Research and planning

2.1. Project management

Software development approach

Students **explain** the software development approach most applicable for this current project. These are normally: Waterfall, Agile and WAgile.

f1nsight utilises an agile development approach, using sprint–based planning and iterative development.

This approach was most appropriate because:

- It will likely evolve over time with client feedback, meaning new features and fixes will be added progressively based on testing and feedback
- Agile approach allows for continuous integrating and testing, issues can be identified and resolved early through this approach.
- Features like performance graphs, live news will benefit from incremental improvements, rather than all being built at once.

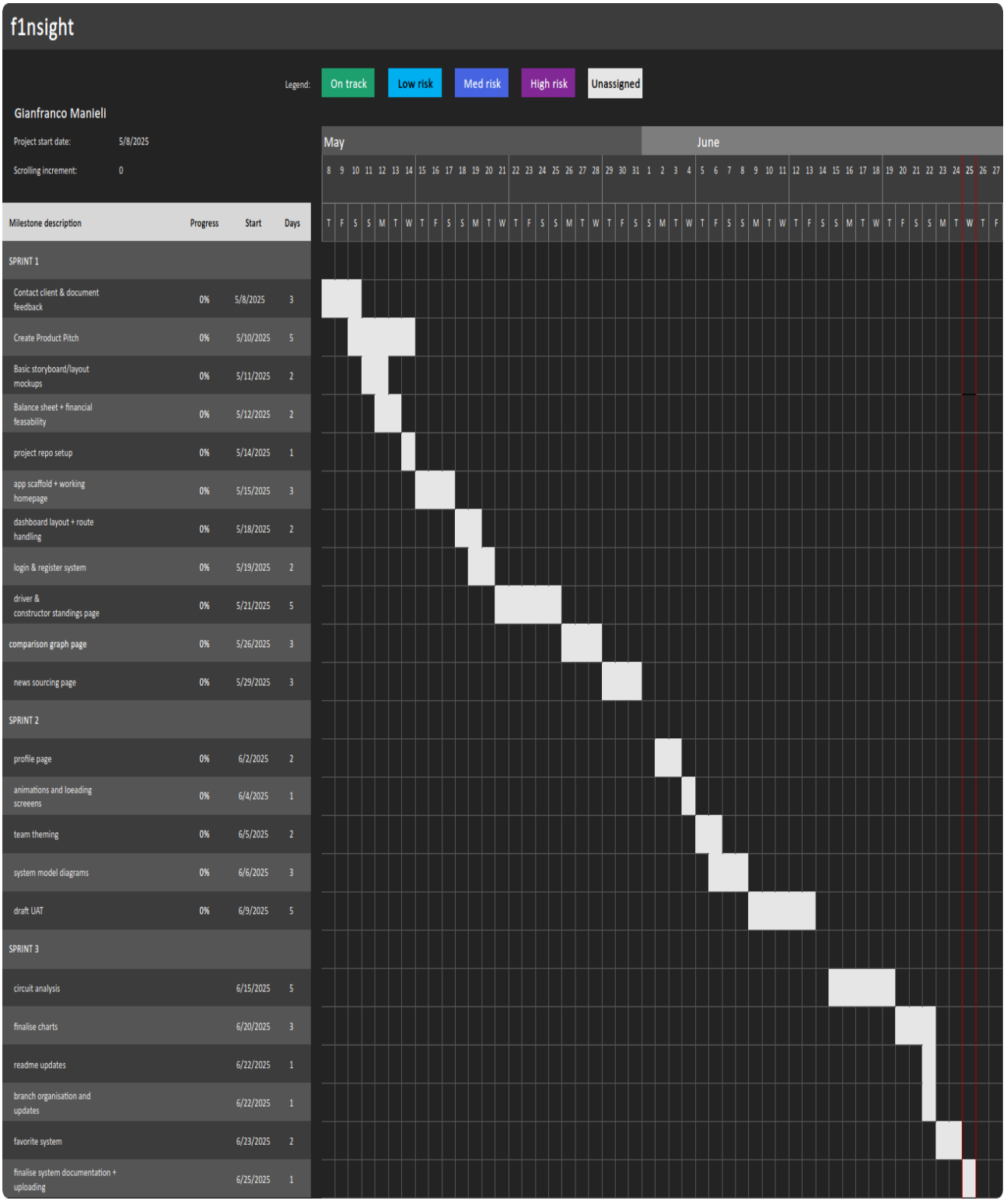
Unlike a rigid waterfall approach, agile supports:

- evolving and changing client needs
- ongoing refinement
- regular delivery of working components

Agile will essentially allow f1nsight to remain adaptable and client-focused, with ongoing testing and feedback at the end of each sprint, evolving the scope and features of the project. This method will help deliver a more functional product that fits the needs of the client.

Scheduling and task allocation

Students **develop** a Gantt Chart that details the tasks required to be completed, person or people assigned to each task, timeline that does not exceed the project due date, resources required. In addition, students **identify** any collaborative tools used. For example Repl.it, GitHub and so on.



2.2. Quality assurance

Quality criteria

Students **explain** quality criteria based upon the needs from Section 1.1. These quality criteria should contain qualities, characteristics or components that need to be included or visible – based on Section 1.1. – by the end of the current project.

Quality criteria	Explanation
Secure login/ authentication system	Users should be able to log in and out securely with encrypted credentials, utilising hashing and secure storage of passwords.
Responsive and intuitive interface	The dashboard should be easy to navigate on both desktop and mobile, with clear data visualisations and smooth UI.
Fast loading and stable performance	Pages should load quickly, with minimal lag during interactions like chart updates or news loading.
Accurate race data and insights	Historical data and comparison must be factually correct and consistently retrieved from reliable APIs
Modular code structure	Codebase should follow clean, modular design to allow for future expansion
Error handling and fallbacks	If APIs fail, system should properly handle this and display cached data or fallback message without crashing

Compliance and legislative requirements

Students **explain** compliance and legislative requirements their projects need to meet and how they plan to mitigate them where possible. For example, projects that deal with

sensitive personal data being publicly available may fall under the Australian NSW Privacy and Personal Information Act (1998) and/or Federal Privacy Act (1988). Alternatively, international standards on information security management such as ISO/IEC 27001 may also be applicable.

Compliance or legislative issue	Methods for mitigation
User authentication and data handling must comply with Privacy and Personal Information Protection Act (1998) and Federal Privacy Act (1988).	Ensure secure login using encrypted passwords, HTTPS for all communication, and limit personal data collection to only what's necessary
Accessibility standards (eg WCAG 2.1 compliance) to ensure equal access for all users.	Follow accessibility guidelines in layout and design, such as appropriate contrast, alt text and keyboard-friendly navigation
Spam and notification compliance	Allow users to opt in or out of desktop notifications and don't send notifications without user consent.
Copyright Act 1968 means API data usage rights and copyright for race data, news content, and logos must be considered	Only use APIs with usage rights. and attribute news sources to original source. Avoid using copyrighted images or logos incorrectly and follow their press kit, etc.

2.3. Systems modelling

Students are to **develop** the given tables and diagrams. Students should consult the Software Engineering Course Specifications guide should they require further detail, exemplars or information. Each subsection below should be completed with Section 1.1. in mind.

Data dictionaries and data types

Students take the needs identified in Section 1.1. of this Systems Report. For each need, students **identify** the variables required, data types, format for display, and so on.

Need							
1. Central dashboard							
Variable	**Data type**	**Format for display**	**Size in bytes**	**Size for display**	**Description**	**Example**	**Validation**
race_name	string	text	50	15 characters	name of f1 race	monaco gp	must not be empty

Need							
race_date	date	yyyy-mm-dd	8	10 characters	date of f1 race	2024-05-26	must be valid date
race_location	string	city, country	60	20 characters	location of race track	monte carlo, monaco	must not be empty
total_laps	integer	whole number	4	3 digits	total number of laps in the race	78	between 40 and 100
driver_standings	string	name, points	60	20 characters	driver points	leclerc, 120	must match driver name
constructor_standings	string	name, points	60	20 characters	constructor points	ferrari, 197	must match constructor name

Need
2. API news integration and notifcations

Need							
Variable	**Data type**	**Format for display**	**Size in bytes**	**Size for display**	**Description**	**Example**	**Validation**
article_title	string	text	100	10-40 characters	Title of news article, will be used to filter teams, etc.	‘Aston Martin launches F1 driver academy, signs Mari Boya ‘	not empty
article_url	string	hyperlink	200	text link	link to article	‘https://au.motorsport.com/f1/news/aston-martin-launches-f1-driver-academy-signs-mari-boya-/10735821/’	must be valid url format
publish_date	date	yyyy-mm-dd	8	10 characters	date the article was published	2025-06-25	valid date, not future

Need							
source_name	string	text	30	15 characters	name of news source	motorsport.com	not empty
notify_user	boolean	checkbox	1	1 character	boolean of whether user opted for notifications	true	true/false only

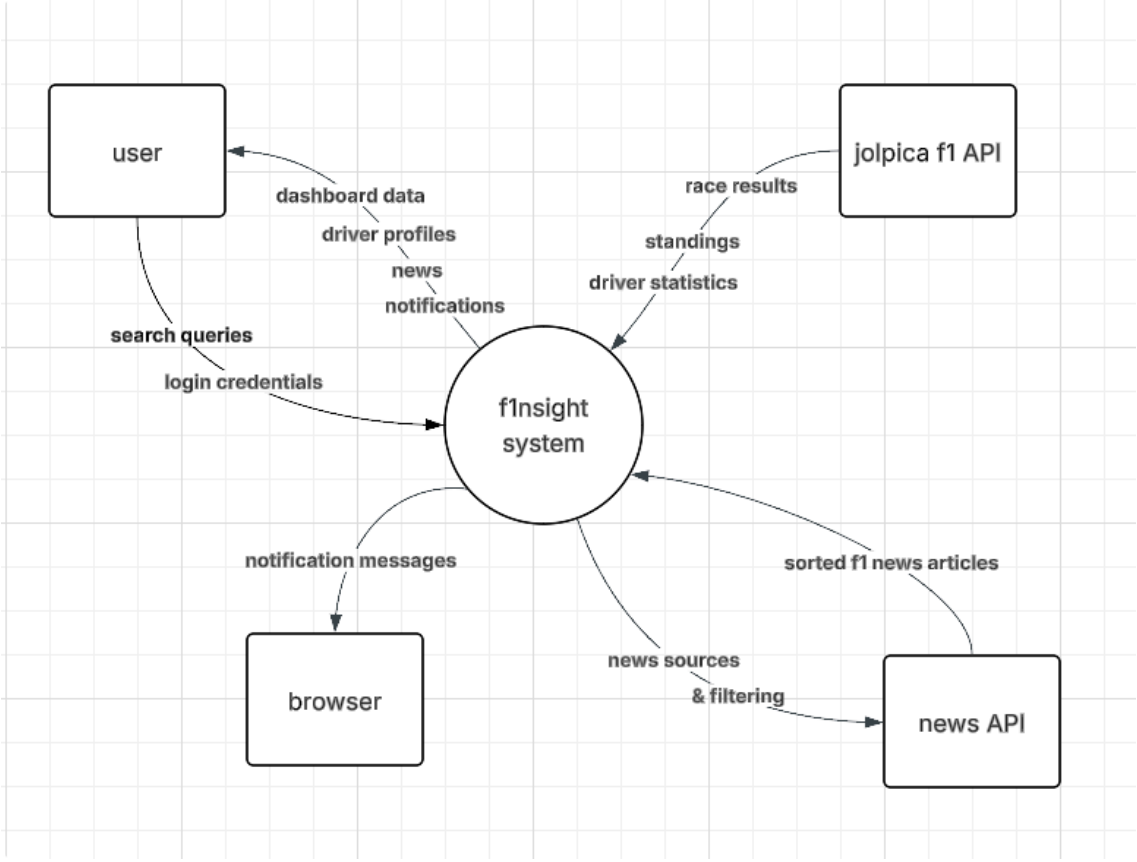
Need							
3. Driver insights							
Variable	**Data type**	**Format for display**	**Size in bytes**	**Size for display**	**Description**	**Example**	**Validation**

Need							
driver_name	string	first name, last name	50	30 characters	full name of f1 driver	Charles Leclerc	match database
constructor_name	string	text	50	30 characters	drivers current team	Ferrari	must exist in constructor list
avg_finish_pos	float	1 decimal place	4	4 digits	drivers average finishing position	3.7	between 1.0 and 20.0
driver_standings	integer	whole number	4	1-3 digits	drivers total points in season	120	greater than 0
consistency score	float	percentage	4	4 digits	calculated consistency metric	85.7%	between 0 %and 100%

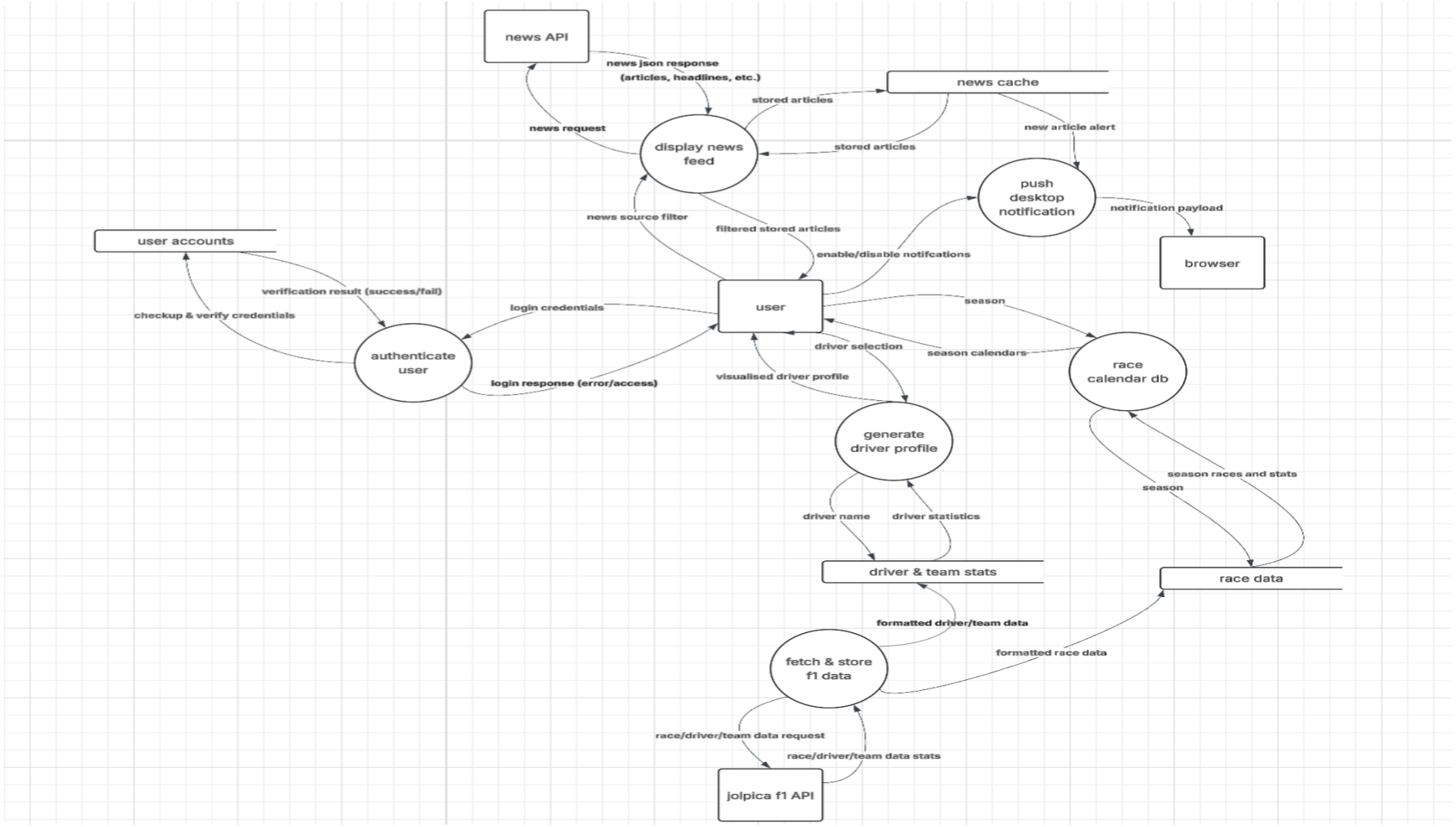
Data flow diagrams

Students **develop** data flow diagrams (DFDs) at Level 0 and Level 1. These diagrams should explicitly include the variables from the data dictionaries previously identified as well as the needs identified in Section 1.1.

LEVEL 0

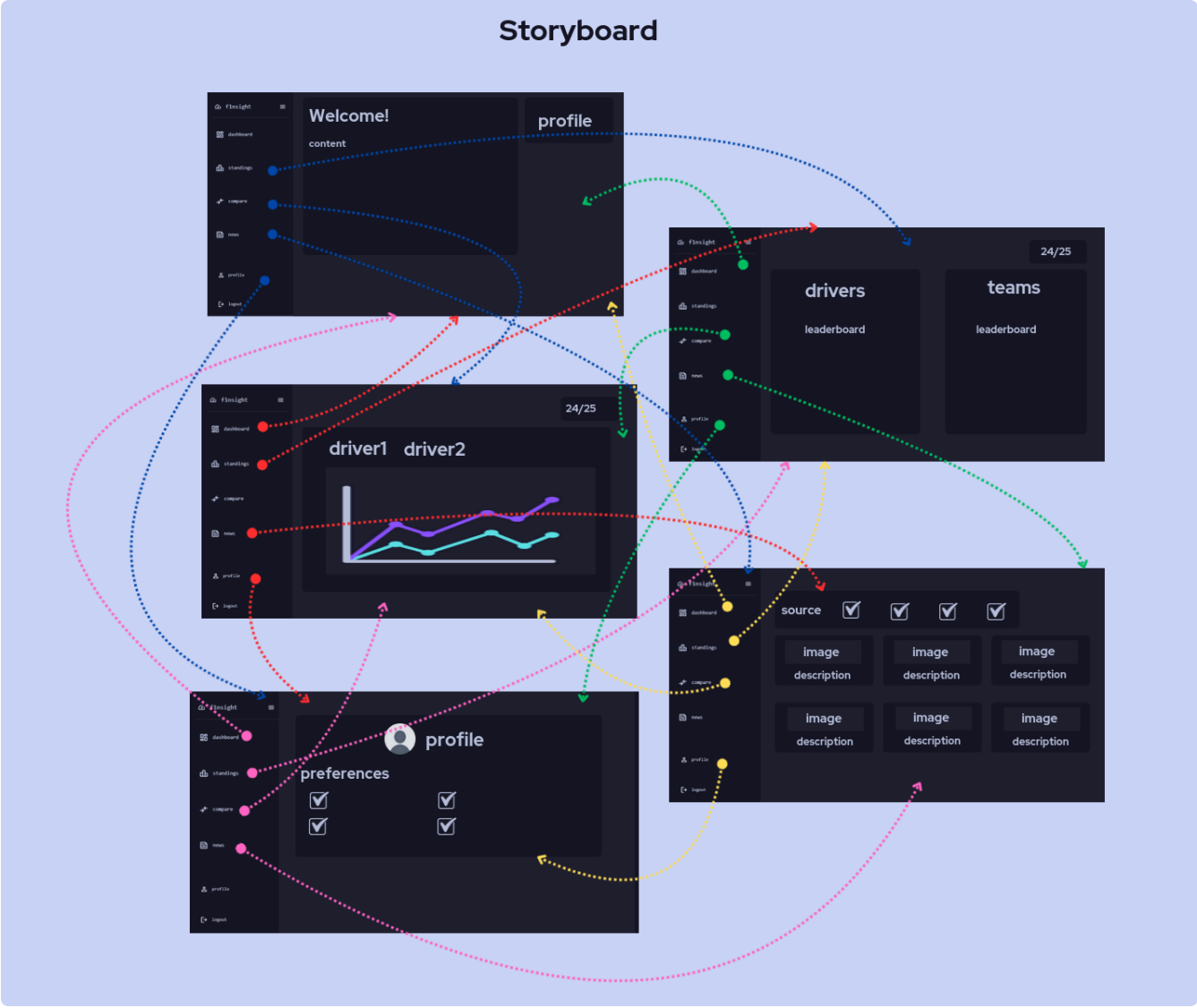


LEVEL 1



Storyboards

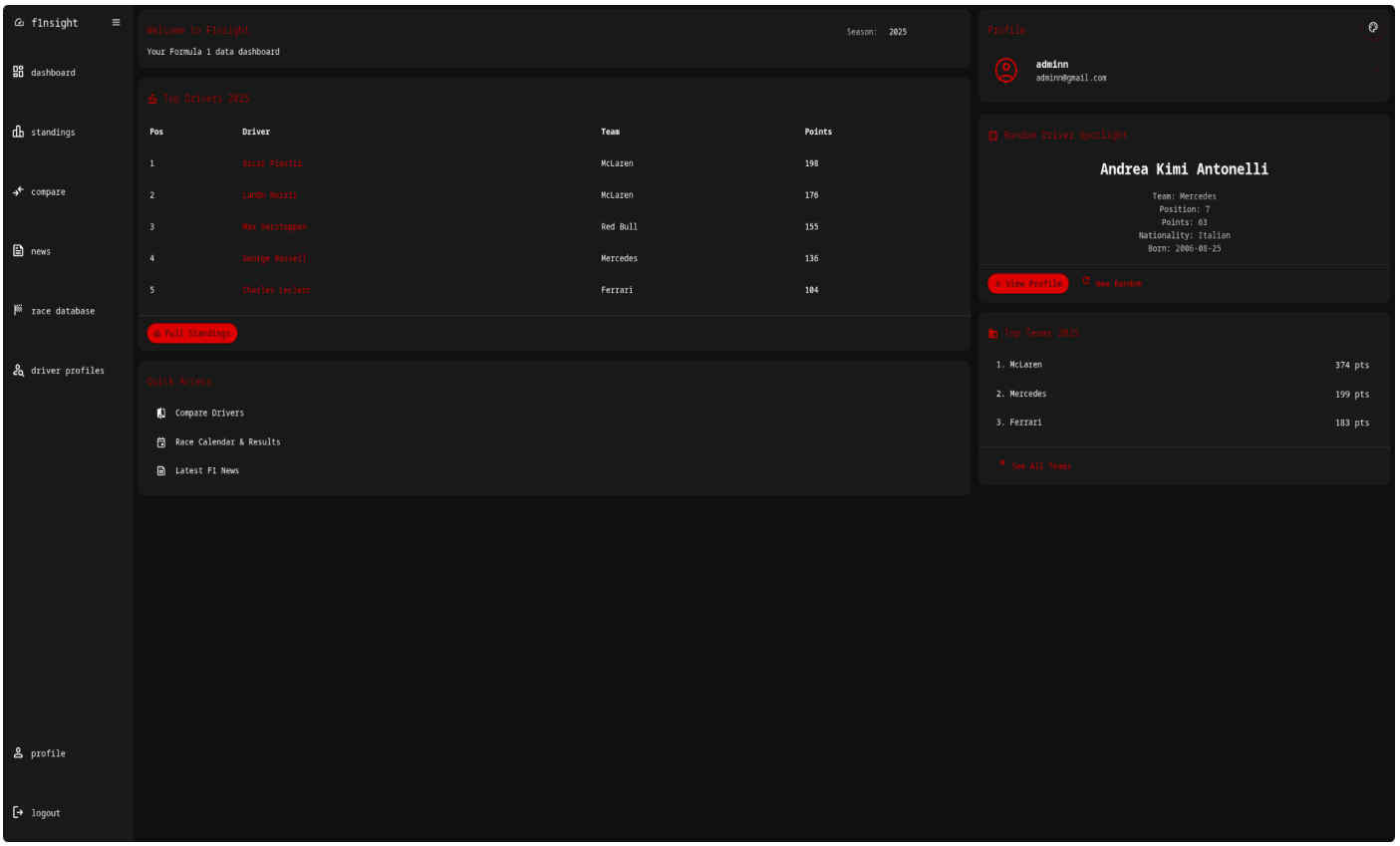
Students **develop** storyboards, visually representing the software solutions they will build.



3. Producing and implementing

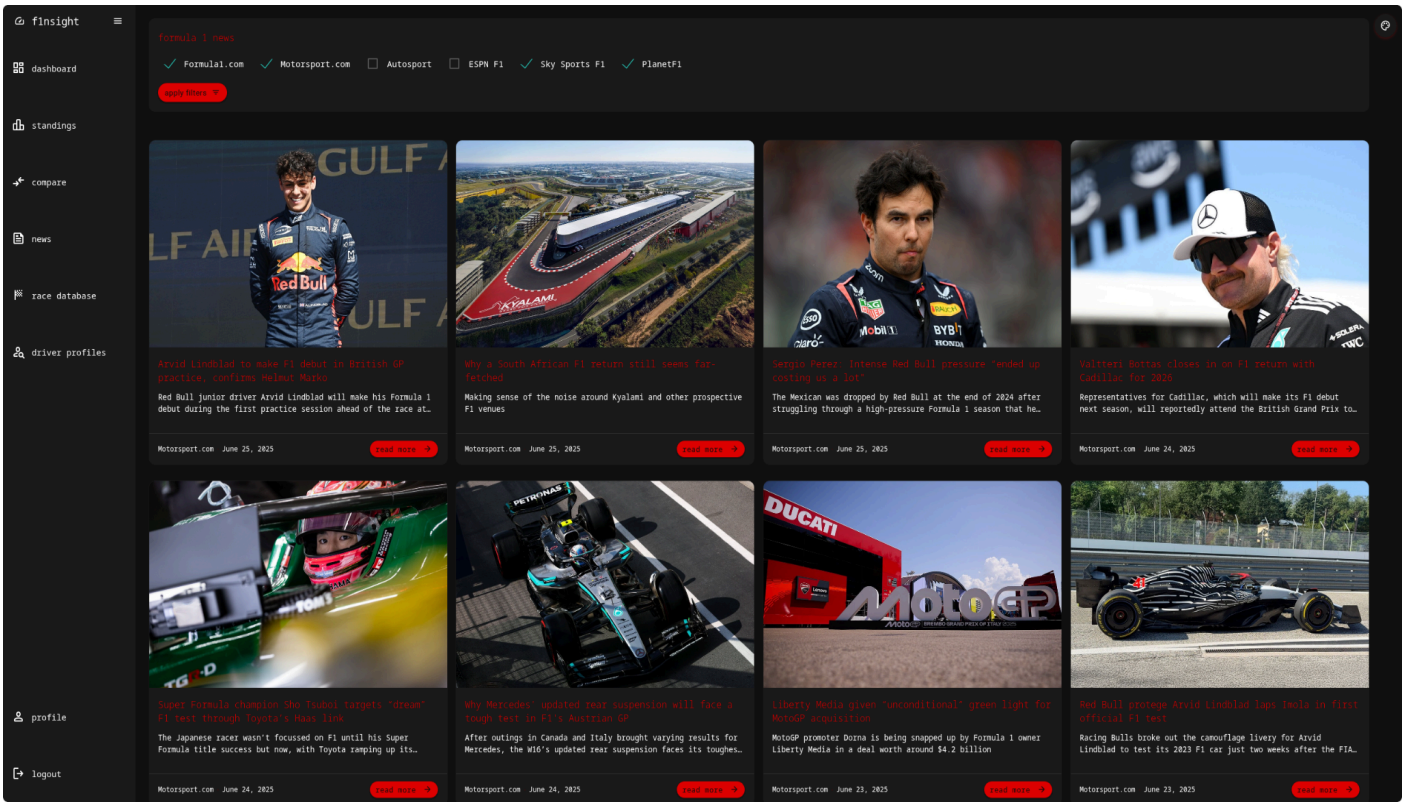
Solution to software problems

1. DASHBOARD



This dashboard page addresses the client's need (section 1.1) for a centralised dashboard that visually organises multiple pieces of information with hyperlinks and quick access. This aligns with client frustration of having to search and scrape through multiple data sources. All pieces of information in the dashboard are referenced in the LVL.0 DFD and LVL.1 DFD for visual reference of how the processes work and interconnect with each other.

2. NEWS INTEGRATION



News integration through an API fulfills clients' general needs of a central f1 hub. Users can easily toggle sources using a checkbox system (which was prototyped in the storyboard), providing a heightened sense of control and personal selection within the website. The news here also links to the news cache data store shown in the LVL.1 DFD

3. DRIVER PERFORMANCE INSIGHTS

The points comparison chart satisfies client request's for performance insights while also reflecting storage of driver information/stats as seen in the LVL.1 DFD. This helps users visually assess trends such as mid-season slumps or race-to-race consistency.

Additionally, this page reflects functionality designed in the storyboard prior.

Version control

Students **describe** what version control system or protocol was implemented.

f1nsight utilised the Github platform and Git commands to handle version control.

Specifically, a branch-based structure reflects my use of an agile development approach.

Deviating slightly from original plans:

- sprint-1 used to build core functionality such as authentication, comparison graphs and standings tables

- sprint-2 used for UI/UX improvements, notifications, theming, and advanced features like race calendar database and driver profiles

- main branch used for final merges and testing.

Each commit included descriptive messages to keep logs of important changes and know where to rollback to if needed. Github version control also ensured:

- collaboration across devices (codespaces)

- clear history of changes and testing processes

4. Testing and evaluating

4.1. Evaluation of code

Methodology to test and evaluate code

Subsystem testing:

Subsystem testing for f1nsight focused on validating interactions between connected components. The authentication subsystem went through testing to ensure proper registration & login flows, aswell as proper session management. My UAT tester Alex was responsible for testing out authentication (specifically any security issues with changing passwords and registration exception handling)

Places where the jolpica API is integrated are also tested independently from base UI components, making sure data flows properly within the API data subsystem, before later testing interactions with the whole system. I did this testing independently through thorough limit testing the API's capabilities and abilities to receive data (finding rate limits)

System testing:

Whole system testing for f1nsight validates the website as a complete integrated product from a user perspective. This includes real-world scenario testing from UAT tester Brandon, who is responsible for end-to-end testing of core workflows and components like comparing driver performance and utilising the dashboard. This testing also accounts for different network speeds, browser environments and user scenarios (Brandon is not very technologically skilled, so it will simulate a new user experience).

Special attention will also be given to error recovery and exception handling, ensuring if there any failures (eg. jolpica API timing out), f1nsight provides the user with meaningful error messages relating to which part of the system failed. This informs the user that errors are not on their end, and can contact the developer (me) for solutions

Code optimisation

Dead code elimination:

f1nsight includes cache clearing to ensure fresh data is received, effectively identifying and removing code that is either unreachable or has no effect on the program's outputs..

Here I implement explicit cache clearing before fetching new driver data to eliminate any stale code paths

‘# CLEAR CACHE TO ENSURE FRESH DATA
datacache.delete_memoized(driverStandings.get_driver_points)’

This prevents execution of unnecessary API calls and calculations when data is already available, improving response times.

Code movement:
f1nsight uses alot of code movement to improve both performance, and user experience. A prime example of this is strategic loading of CSS and JS resources. Critical CSS is loaded in the document head to prevent fouc (flash of unstyled content):
‘<script>
    document.documentElement.style.visibility = 'hidden';
    window.addEventListener('load', function() {
        document.documentElement.style.visibility = 'visible';
    });
</script>’

In addition to this, non-critical JS is deferred to improve initial page load time
‘<!-- DEFER NON-CRITICAL JAVASCRIPT -->
<script defer
src="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/js/materialize.min.js"></script>
<script defer src="{{ url_for('static', filename='js/main.js') }}"></script>’

Strength reduction:

strength reduction is utilised to simplify and optimise code within the f1nsight codebase. more specifically simplifying DOM operations in JS:

before optimization:

```
document.querySelectorAll('.select-options, .select-search').forEach(el => {
    if (el !== options && el !== search) {
        el.style.display = 'none';
    }
});
```

after optimization: efficient direct references

```
options.style.display = options.style.display === 'none' ? 'block' : 'none';
search.style.display = search.style.display === 'none' ? 'block' : 'none';
```

I have optimised JS performance by replacing expensive lengthy DOM queries with direct element references, improving overall speed of the website and enhancing the user experience.

4.2. Evaluation of solution

Analysis of feedback

Interface & design

Feedback surrounding the aesthetics and theming of the website was great and design was consistently praised. Navigation was said to be intuitive and clean and good for user flow. However some users (Gordon and Mathieu) who were randomly chosen for feedback, requested additional theming options of their favorite themes (Redbull & Mercedes respectively). This feedback supports my ideals for a user-centric design with a sense of personal touch through theming.

Data analytics & charts

The feedback surrounding the existing charts are mixed. Normal casual users like Brandon San were very intrigued and appreciative of these graphs. However power users like Christine Nguyen demand more insightful, in-depth graphs. In order to appeal to this demographic, I will look into expanding the comparison charts. Including an estimated score graph, consistency score, and driver ratings per race.

Historical data & content

The large scale of content archiving for races and previous drivers is a clear strength of f1nsight. Testers expressed strong positive feedback for the comprehensive nature of the historical archives and the presentation of driver information (such as career statistics).

The race calendar in particular impressed testers as particularly valuable with its reach from 1950 till current year 2025. Christine said ‘I could easily visualise the Ferrari domination of the early 2000s. I will move forward with this feedback, and continue development of data archiving.

Testing methods

Students **identify** the method or methods of testing used in this current project. For each they use, students are to **explain** how and why it was used.

Method	Applicability	Reasoning
Functional testing	Used extensively to test feature functionality during development	Functional testing helped verify functionality of each component/subsystem and ensure they will work when used. Unit testing for the jolpica API and data processing helped catch regression issues when implementing new functionality and keep core functionality worked
User Acceptance testing	Used during later stages of development, when product was near launch	Group of 3 UAT testers of varying backgrounds were chosen to test functionality and gain feedback from a whole wide range of individual contexts/experience. It also allowed for real word testing scenarios to ensure f1nsight met real user needs. This method verified whether or not the f1nsight system satisfied actual user needs and if UI/UX was of high quality.
Live data	Used all throughout development process	Testing both news and jolpica f1 APIs live and in a real world environment helped to visualise if my data processing algorithms could handle real world race data where formats often varied across seasons.

Method	Applicability	Reasoning
		(eg. jolpica API driverstandings formatting changed from 1957 to 1958)
Simulated data	N/A	I relied on real world data to simulate actual user usage of the website. I avoided using simulated datasets as they would not realistically depict how APIs would provide info
Beta testing	N/A	Time restraint of project did not allow time for proper beta testing, so I chose to avoid that and go straight into agile sprint development with feedback at the end of each cycle.
Volume testing	tested jolpica API to see if it'll handle lots of requests	would need to test if too many requests would break the API. jolpica has a relatively high rate limit and i was able to iteratively search years through learning that from volume testing.

Security Assessment

Students are to **perform** an extensive security assessment of their final application and e**xplain** the countermeasures implemented.

Threat	Countermeasure
SQL injection means attackers can manipulate input fields, potentially exposing sensitive data or compromising the entire database.	I implemented SQLAlchemy ORM which used parameterised queries to prevent direct SQL injection. Also input validation has been added using WTforms validator library. Also regular scanning of code will be done through automated Github security tools
Cross site scripting (xss) means attackers could inject malicious scripts into webpages	I used jinja2 automatic escaping of html characters. Also sanitised all dynamic content before rendering, and employed strict input validation for all user-provided inputs
Cross-site request forgery (csrf) attacks trick authenticated users into executing unwanted actions on a website they're currently authenticated.	form.hidden_tag includes a CSRF token in each form submission <form method="POST"> {{ form.hidden_tag() }} </form>

Test data tables

Variable	Maximum	Minimum	Default Value	Expected Output	Actual Output	Reason for Inclusion
season	2025	1950	2025 (current year)	valid data for selected year	valid data	test historical data boundaries and see if the whole range works
password	50 char	8 char	n/a	account creation	account creation	ensure there are password length restrictions, preventing both weak passwords and buffer overload attacks
comparedrivers	2 drivers	0 drivers	disabled	valid comparison chart for driver/s should show	valid comparison chart for driver/s	make sure the comparison feature visualisation works with both single and multiple driver selections
searchquery	100 characters	0 char	empty string	matching results	matching results	search functionality should handle input lengths, making it important for path testing different search scenarios.

Analysis of solution against quality success criteria

Quality criteria	Met?	Analysis
Secure login/ authenticatio n system	Yes	the authentication system works well with bcrypt password hashing and proper session management. rate limiting has also been added to block brute force attempts. flask-login helps manages user sessions securely. everything from login to password resets follows good security practices.
Responsive and intuitive interface	Yes	the ui adapts nicely to different screen sizes. The sidebar collapses on mobile while keeping the sidebar button accessible. cards, buttons and navigation also have consistent styling throughout the app.
Fast loading and stable performance	Not very	still issues with comparison graph loading speed, current algorithm goes through each race and scans for points. This is too slow and I could probably find a better more efficient solution by utilising other API methods
Accurate race data and insights	Yes	the app pulls correct f1 data from jolpica APIs. the historical stats match official records, and all the visualisations correctly show performance trends. when API calls fail, f1nsight shows proper error messages instead of wrong data.
Modular code structure	yes	I built the code using flask blueprints to separate different concerns like auth, API data, and frontend. this made adding new features much easier during our sprints. the consistent naming and organisation also made it easy to develop

Quality criteria	Met?	Analysis
Error handling and fallbacks	yes	the app handles API failures well with proper user messages. i have exception handling for the important stuff to prevent crashes, and validation gives good feedback for bad inputs.