COLORADO SCHOOL OF MINES
EARTH • ENERGY • ENVIRONMENT

CSCI 370 Final Report

BITT

Bjorn Karlen
Ivan Lopez Rubio
Thomas Collyer
Tobin Ford

Revised December 10, 2025

CSCI 370 Fall 2025
Mr. Caleb Bartel

Revision	Date	Comments
New	09/03/2025	Document was created
Rev - 2	09/05/2025	Sections I through IV were initially populated
Rev - 3	09/12/2025	Revise sections I through IV, and add to section VI
Rev - 4	09/19/2025	Revised sections I through VI and added section VII
Rev - 5	09/21/2025	Revised sections I through VII
Rev - 6	10/12/2025	Added section VIII
Rev - 7	10/26/2025	Revised sections I through VIII and added sections IX & X
Rev - 8	11/30/2025	Revised all sections and added section XII
Rev - 9	12/2/2025	Revised all sections based on peer feedback
Rev - 10	12/10/2025	Final revisions and edits were made to the report

Table 1

Table of Contents

I. Introduction

The project aims to build an extension for the current Ready-Set-Sim website, specifically by developing a backend recommendation engine for the “Simfigurator.” This new engine will take questionnaire input from users and, based on their responses, recommend parts that best fit their needs. After the Simfigurator provides its recommendations, users will be directed to a basket containing the recommended items, with the ultimate goal of encouraging users to utilize the RSS affiliate links associated with these items.

Ready Set Sim (RSS) is a company dedicated to helping racing simulation enthusiasts achieve their ideal racing setups. Currently, RSS offers a tool, the Sim Rig Builder, that assists users in selecting compatible parts. Now, they want to enhance this capability by not only ensuring compatibility but also actively recommending specific parts to users. RSS recognizes the need for a simpler way to suggest customized sim rigs based on each user's needs, current equipment, and skill levels. By implementing the Simfigurator, RSS hopes to lower the entry barrier for those wanting to set up a new sim racing rig, providing a recommended build tailored to users’ questionnaire responses.

As for data sources, the team will utilize site analytics available through the Django admin panel. User carts, known as “baskets,” reflect the setups being selected, offering valuable information about which brands and products are most popular among the community. This data will help inform product recommendations. The client has also provided a document summarizing survey responses they have collected. While this document outlines the process for using survey data in recommendations, it lacks actual recommendations or actionable guidance, so without further clarification or inquiries, its usefulness remains limited.

The primary stakeholders and users of the software will be members of the sim racing community, who represent the largest group benefiting from the new system. Additionally, people considering entering the sim racing scene are seen as another substantial group of stakeholders.

Regarding software maintenance, Ready Set Sim will be solely responsible for maintaining the software, as per the client’s request. After the conclusion of the Fall 2025 field session (12/10/2025), all project artifacts will be transferred to RSS’s GitHub repository, and BITT will no longer be involved in the software's maintenance or updates.

II. Functional Requirements

1. **Information gathering**
    - From users, receive answers to relevant questions to narrow the scope of possible products.
    - Understanding what makes a good recommendation.

2. **Recommendation**
    - The system should recommend a product or multiple products.
    - The system should respond with rational outputs within some error bound as defined by RSS and BITT.

3. **API Behavior**
    - The system shall expose an API to generate recommendations based on:
        i. A single product for similar or complementary items.
        ii. A set of products for complementary items.
        iii. A set of questionnaire responses for a minimal functional set of products.

4. **Incorporation and synchronization**
    - The system's recommendation tool should be able to communicate well with other implemented backend tools.
    - The system should utilize internal APIs to understand the compatibility of products while searching for points of recommendation.

III. Non-Functional Requirements

1. **Understanding SIM Racing**
    - The system should weigh SIM racing configurations and their tradeoffs.
    - The system should be able to translate questionnaire responses into configurations as an experienced SIM racer would.

2. **Security**
    - The recommender tool should not expose any training data to users.
    - RSS-owned IPs should remain private, inaccessible to the public, and unadulterated.

3. **Streamline questionnaire**
    - Redundant questions should be eliminated.
    - Redundant recommendations should be eliminated as the sample space decreases.

IV. Risks

Risk	Mitigation Plan	Likeliness	Impact
Lack of quality data	Review existing data for inconsistencies, identify gaps, and create community questionnaires to gather the necessary information.	Likely	Moderate
Disconnect from Clients/ Goals	Maintain regular, structured communication, including weekly meetings and continuous assessment of client requirements to ensure alignment.	Likely	Major
Scope Creep	Clearly define project steps and requirements upfront. Both parties must explicitly specify and agree upon any additional features.	Very Likely	Moderate / Major
Schedule Conflicts	Use a combination of in-person and remote collaboration to maintain flexibility and ensure progress despite conflicting schedules.	Extremely Likely	Major
System Complexity and Technical Debt	Conduct regular code reviews, maintain clear documentation, and refactor code when necessary to keep the system maintainable.	Likely	Moderate
Compatibility Errors Between Recommended Items	Implement rigorous item compatibility testing and data validation to ensure recommended items (e.g., wheels and wheelbases) work together.	Likely	Major

List of Possible Risks and Respective Mitigation Plans

Table 2

V. Definition of Done

1. **Minimal Useful Feature Set**

   Completion of backend product recommendation feature utilizing a mixture of logic and machine learning to provide relevant product recommendations to Ready Set Sim users from three entry points: Single product, multiple products, and survey intake.

2. **Client Testing**
   - Given a single product, the API should be able to recommend a set of similar products.
   - Given a single product, the API should be able to recommend a set of complementary products.
   - Given a set of products, the API should be able to recommend a set of complementary products.
   - Given a set of questionnaire responses, responses should be able to recommend a minimal functional set of products.
   - The recommendation API should never recommend a set of incompatible products.

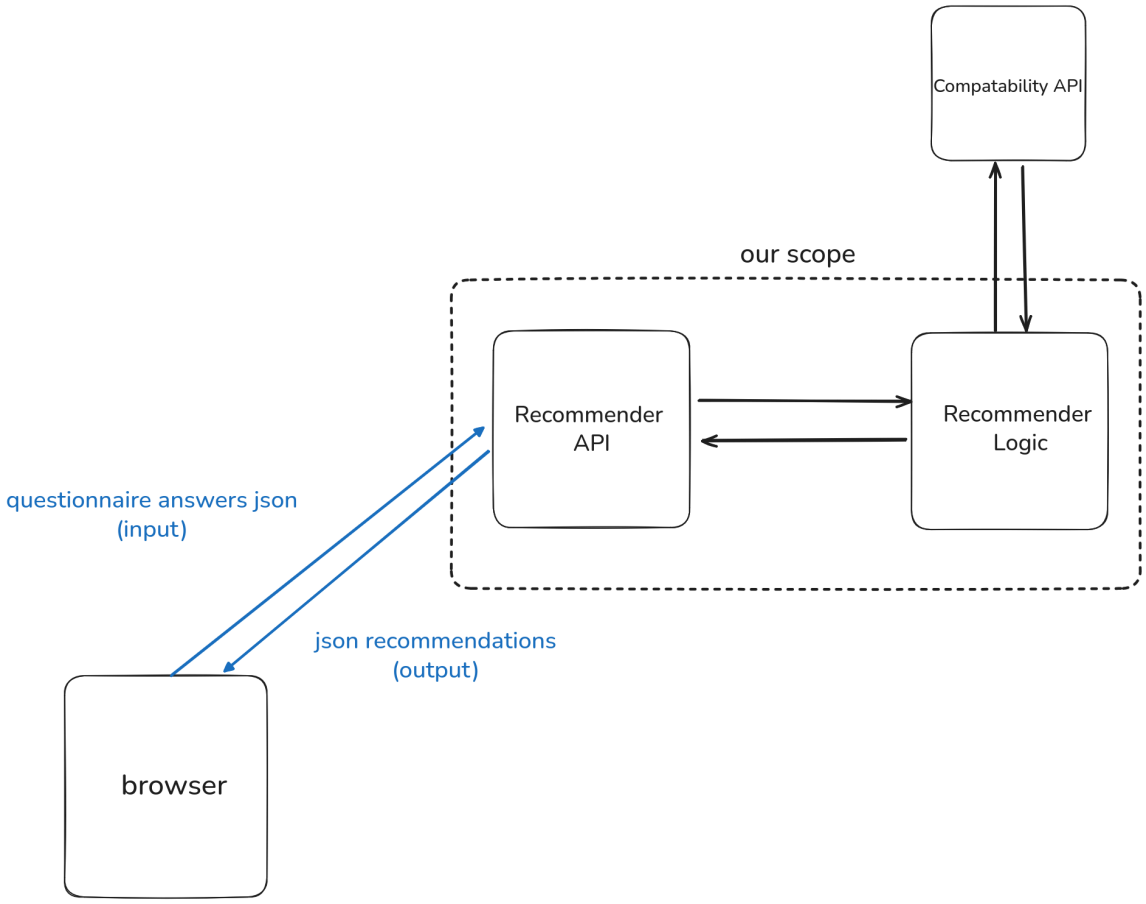
3. **Product Delivery**

   Upon the completion of the Fall 2025 Field Session (12/10/2025), the product will be completed and in a git repository under RSS's possession. On the aforementioned date, the product will be out of the possession of BITT and will officially be a product of RSS.

VI. System Architecture

1. High-level design

The existing Ready Set Sim (RSS) Web App will call the new recommender API endpoint to obtain a ranked list of compatible products. The backend service will validate requests, map inputs to features, recall candidates from Postgres, and enforce hard compatibility constraints via the existing Compatibility API to ensure all components fit without error. Surviving candidates will be scored by an in-process logistic-regression model and filtered by simple business rules to produce a top-K list with brief "why" explanations suitable for the UI. If the compatibility dependency returns no matches, the service will gracefully degrade and mark its responses accordingly. The API follows existing auth and returns a request identifier and model version for observability, and displays machine-readable errors for invalid inputs or unsatisfiable constraints.



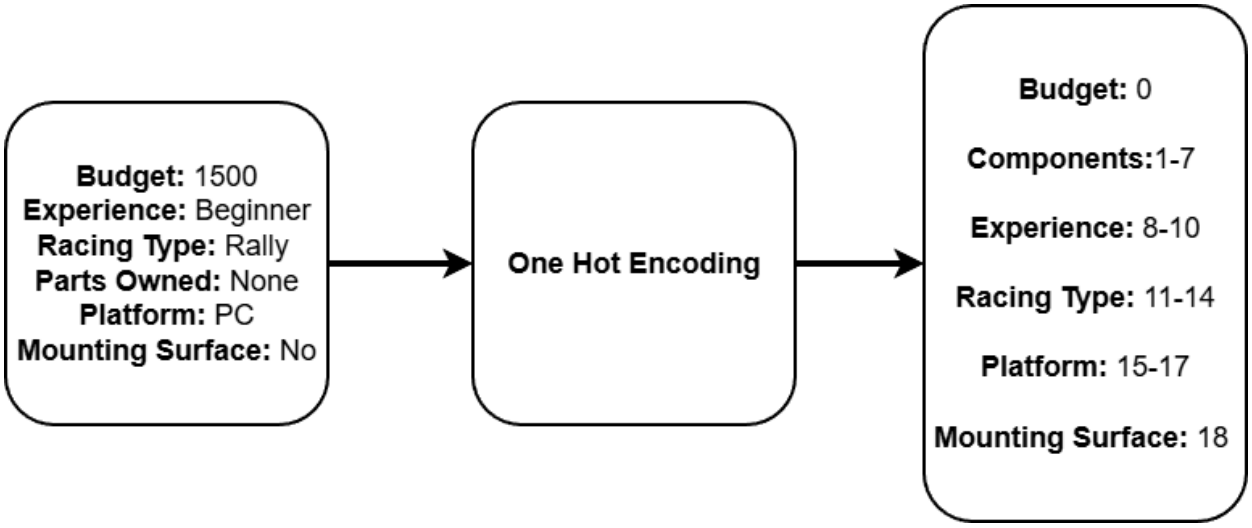
Overall Project Service Architecture

Figure 1

2. Detail Design

This backend API is a RESTful service. Questionnaire responses are formatted into a JSON response in the front end and then validated to ensure the response has appropriate contents. The questionnaire responses are then vectorized and normalized so that they can be forwarded into the recommendation engine. Ensuring that the model has been trained and has mapped the questions to specific features (as shown in Figure 2), the engine should then generate outputs.

In this design, the backend is running on a single physical computer. In deployment, this is an Azure Virtual Machine (B1 Bucket).



Survey Response Vectorization Logic (High Level)
Figure 2

3. Recommender API

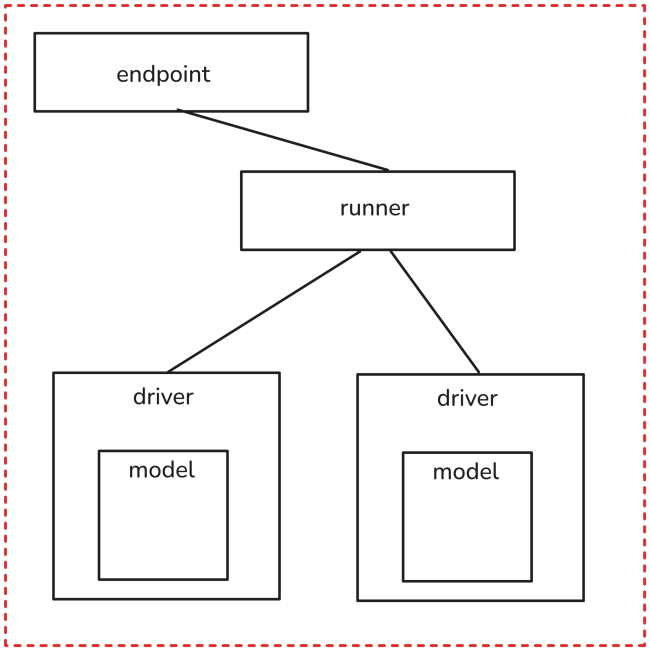
The web app sends recommendation requests to the server, where the backend recommender API makes a recommendation and forwards it to the basket manager. The resulting basket is then returned to the frontend, and React hooks redirect the user to the existing Sim Rig Builder.

The client first submits a POST request to /v1/recommend/ and includes a request ID for tracing. The view validates the JSON payload containing the survey responses, then passes it to the runner service layer, which decides which model to use for the recommendation.

The runner service layer calls a series of drivers that contain business logic used to process information before invoking the models.

The model layer runs inference using pre-computed weights, then applies external “policies” to check compatibility, mounting surface availability, and budget constraints. If any of these checks fail, the service updates model inputs and re-runs inference with a relaxed search to generate a recommendation that satisfies our policies. Information about product availability and other constraints is encoded into the trained weights, so the model does not need to call external services during inference.

The model returns suggested product IDs, metadata, considered IDs, and fallback logic for traceability. The API adds suggested product IDs to a user basket and returns a basket ID to the frontend, which redirects the user to the interactive Sim Rig Builder, where they can continue to iterate on their rig.



Simfigurator Backend Logic (High Level)
Figure 3

4. Technical Design Issues

The biggest challenge our team faced was accessing and understanding the database. The issue wasn't with the connection itself failing, but rather a lack of clear information on locating, connecting to, and using the existing database. Documentation about the database setup, credentials, and structure was limited, which made it difficult to verify where the database was hosted and what data it contained.

Because of this, the team spent additional time trying to identify the correct connection details and understand how the database was intended to be used. Without clear guidance, testing and development that depended on live data were delayed.

To address this, we clarified database ownership, documented connection steps, and created a consistent process for future team members to connect and query the data. This process has been documented in a separate helper document sent to the client.

VII. Software Test and Quality

Activity	Contribution to Software Quality
Item Compatibility Testing	Item compatibility testing ensured that the system did not recommend physically/ functionally incompatible products. The provided compatibility API endpoint ensured that the wheel and wheelbase recommended were compatible. Compatibility checks written into the testing process contributed to providing trustworthy and consistent results.
Unit Testing	Unit testing focused on verifying that components of the recommendation system worked correctly individually and with one another. This included ensuring that user inputs are captured properly and that the recommendation logic produces expected outputs for different answer sets.
Integration Testing	Integration testing helped pinpoint any issues that might come into play when adding to our existing solution. This will ensure that any piece that we add will not change the interaction of any existing code. User responses will flow into our engine and then be processed accordingly; no step should change the other steps' functionality.
Code Reviews	To ensure that harmful code was not pushed into production, in-depth group code reviews were completed. After that, the code was sent to the head developer, and he approved any changes made to the codebase.

Testing Used and Impacts on Software Quality

Table 3

VIII. Project Ethical Considerations

Several principles from the ACM Code of Ethics are especially relevant to our project. Protecting user privacy is critical, following ACM 1.6 Respect privacy and 1.7 Honor confidentiality. Survey responses must remain inaccessible to the public, and data should be anonymized to protect sensitive information. In addition, users should be informed about what is being collected and how long it will be stored to maintain transparency and trust. These principles have the biggest potential risk.

Our system's recommendation feature also raises important ethical considerations. According to 2.1, we should strive to achieve high quality in both the processes and products of professional work. Recommendations should be designed to genuinely benefit users, not simply to maximize profit for the platform. Nudging users toward higher spending without a clear benefit would undermine the integrity of the product and could cause harm.

Security is essential, especially since the recommendation API is more exposed than other application parts. Following 2.4, accepting and providing appropriate professional review, it is important to ensure this access point is robust against misuse. Preventing malicious actors from exploiting the API protects both the integrity of the system and the safety of its users. All of our code will be reviewed by both peers and stakeholders to achieve this goal.

The Simfigurator follows the Michael Davis Harm Test to choose the option that causes the least harm. To reduce risk, PII is not collected, eliminating the possibility of accidental disclosure. The Michael Davis Publicity Test was performed; the only data requested is such that users would be comfortable seeing it on the front page of a newspaper.

IX. Project Completion Status

The goal of this project was to develop a machine learning based algorithm for a sim racing parts/ build recommendation system based on a user survey. Users will answer *n* questions, and an in-house mapping function will generate vectors to input into a logistic regression algorithm. A list of product IDs is outputted, and those IDs are then used to build a shopping cart that is sent out of the recommender API.

Once the shopping cart is sent out of the recommender API, it integrates with their existing basket API and produces an output of products in their custom shopping basket UI. Users can then switch out products they don't like, further customize with additional products not included in the recommendation, or proceed with purchasing the products they were recommended.

For testing, unit tests were performed for schema validation, user input validation, user analysis testing for our machine learning approach, and fallback testing for the algorithm. Schema validation consisted of ensuring that all JSON responses being sent into our API were valid and would not produce errors. Input validation consisted of ensuring that the responses that followed the JSON schema also contained responses/ answers that were within the scope of sim racing parts and/or sim racing as a whole. Fallback testing consisted of triggering model fallbacks through inputting questionnaire edge cases into the model to ensure that the appropriate model fallbacks would occur to still generate an acceptable output.

Clients were asked to develop part recommendations based on generated survey responses to test the algorithm's output against theirs. Specific survey responses that would elicit a "no_compatible_combo_within_budget" fallback response were inputted to ensure that the necessary circumventing logic functioned as needed to output client-defined shopping baskets.

X. Future Work

Possible future work can consist of expanding the recommendation algorithm to encompass more products that are offered by Ready Set Sim. Currently, only wheels, wheelbases, cockpits, pedals, monitor stands, shifters, seats, and handbrakes are within the algorithm's scope, but there are other product types in the RSS database. The recommendation system can be modified in the future by Ready Set Sim members to include or exclude whatever products they require. All the documentation has been provided to the client for the important aspects and overview of the project.

Regarding the project MVP, all of our clients' requirements have been met, and some additional features have been added along the way, but another direction that they expressed interest in is to incorporate community feedback in the form of surveys. Responses to these surveys could influence product weighting within the algorithm so that products deemed by the community to be best are recommended above others. This would involve a much larger overhaul of the engine, but it could be completed by an experienced team in the future.

XI. Lessons Learned

Firstly, database architecture and management are important when trying to gather useful information. When training our model, we needed certain data(s) from each product. This quickly became an issue as we struggled to find the items we needed within the database. Within the context of school, we are privileged to see some well-organized databases in our learning, and we all learned a lot about why that is important when looking over existing code bases.

The next lesson: You don’t need to re-invent the wheel. We were told that previous developers for this project decided to take their own approach when it came to machine learning. They were writing their own algorithms and optimization code. We decided to utilize a Python library for the underlying model, which saved us a lot of time and headaches. Using an existing system allows us to spend more time focusing on fine-tuning the recommendations rather than building the engine from scratch.

Lastly, we furthered our understanding and appreciation of code reviews when it comes to efficiency and maintaining progress. We had tests in place to make sure that the code is functioning properly. We learned that we don’t need to look at every line and test ourselves when we know that it's working; we just need to take a less granular approach and look at if there are any security risks or any edge cases that we aren’t accounting for. Learning this increased our productivity significantly, as the focus of our code reviews shifted from syntax and bugs to flow and architecture.

XII. Acknowledgments

We would like to express our gratitude to Ready Set Sim for providing us with our problem statement and our weekly meetings that supported our progress over the semester. Their constant availability and focus on our project really allowed us to excel and surpass some of our MVP requirements.

We would also like to acknowledge the work that the whole team has put into the project over the course of the semester to solve this difficult and extremely fun problem, and we thank our advisor, Caleb Bartel.

Finally, we would like to acknowledge the open-source community for the numerous Python libraries we utilized to reach our goals for our recommendation system.

- Django - For backend and database compatibility
- Numpy - For vectorization and other calculations
- Scikit-learn - For models to create our recommendation engine

XIII. Team Profile

Bjorn Karlen
bkarlen@mines.edu

He loves to ski and ride his motorcycle.

Ivan Lopez Rubio
ivan_lopezrubio@mines.edu

He enjoys training in Muay Thai and boxing outside of coding.

Thomas Collyer
thomas_collyer@mines.edu

Outside of coding, he enjoys hiking and playing card games with friends.

Tobin Ford
tobinford@mines.edu

When he is not coding, he enjoys rock climbing.

Appendix A – Key Terms

Term	Definition
Sim Racing	*Simulated racing uses computer software and hardware to mimic the racing experience*
RSS	*Ready Set Sim (Client)*
Simfigurator	*A tool to suggest a customized sim racing setup*

Table 4