Vehicle Matchmaker Feature Review

An in-depth, formal code review of User Story 15, written by the Car2go2 team.

## Purpose

The changes considered in this review are those described in pull request #290. If you have access to the repository, you can view the PR by clicking the link below.  
<https://github.com/mahmoudmus/car2go2-soen341projectW2024/pull/290>

The purpose of these changes are to develop a new, interactive way for users to find vehicles that suit their interests. The idea is to provide the user with a series of vehicles that they can either *like* (by pressing a heart icon) or *dislike* (by pressing an *x* icon).

Based on the vehicles liked, a vehicle *profile* is built which describes what the user’s ideal car would look like. This profile is then compared against vehicles in Gas’s database and the user is presented with the closest match.

## Becoming Familiar

To better understand and appreciate the code written for this feature, we suggest reviewing pull request 278 & 279 to better understand aspects of the frontend approach, and also pull request 281 to understand the backend. Links to these PRs can be found below.

<https://github.com/mahmoudmus/car2go2-soen341projectW2024/pull/278>  
<https://github.com/mahmoudmus/car2go2-soen341projectW2024/pull/279>  
<https://github.com/mahmoudmus/car2go2-soen341projectW2024/pull/281>

Further, we suggest reviewing the meeting minutes for meetings that took place on March 28th and on April 6th. These meetings are the birthplace of the majority of the ideas implemented though the changes considered in this review. Links to these meeting minutes are found below.

<https://github.com/mahmoudmus/car2go2-soen341projectW2024/blob/a6f201e9d29487697254e222e200103ec4c93d4c/Sprint%204/Meeting%20Minutes/Car2go2_Sprint4_m16_Mar_28.docx>

<https://github.com/mahmoudmus/car2go2-soen341projectW2024/blob/a6f201e9d29487697254e222e200103ec4c93d4c/Sprint%204/Meeting%20Minutes/Car2go2_Sprint4_m17_Apr_6.docx>

## Environment Setup

To test these changes locally, you must follow the installation instructions as detailed in the repository’s readme (see the link below).  
<https://github.com/mahmoudmus/car2go2-soen341projectW2024?tab=readme-ov-file#installation-instructions>

Afterwards, you must perform the following actions:

* If PR 290 has already been merged, checkout the *dev* branch and ensure your local version of this branch is up to date (git fetch & pull).
* If PR 290 hasn’t yet been merged, checkout the 245-us-15000-vehicle-dating-ui branch and ensure your local version of this branch is up to date (git fetch & pull).
* Run *npm install* in the terminal (at the *app* directory, not the repository’s root) to ensure your dependencies are up to date.
* Run *npm run devstart* to start the local development server.
* In your browser, navigate to <http://localhost:3000/> and login as any user type (customer, csr, or admin).
* Click on *Learn more* in the vehicle matchmaker CTA on the home page, or on the *Matchmaker* link in the navigation tab.

You should now be able to test the matchmaker functionality for yourself.

## Code Review Approach

This code review was performed by breaking the feature down into manageable chunks, with minimal overlap - more or less on a file by file basis.

For each of these domains, we searched for areas of improvement in terms of coding standards, readability & maintainability, functionality & requirements, error handling, performance considerations, security, testing, comments & documentation, version control practices, and dependencies, if appropriate.

## Frontend

The frontend aspect of this feature concerns the new homepage CTA, the new navigation bar link, the matchmaker rules page, and the vehicle dating cards. Changes made to the following files fall are considered to be frontend code: datingCard.component.js, swipeGame.component.js, reservationFinder.component.js, views/dating/card.ejs, views/dating/start.ejs, views/index.ejs, views/partials/navigation.ejs, and views/reservation/return.ejs.

## Backend

The backend aspect of this feature concerns primarily the matchmaking algorithm. Changes made to datingController.js are thus considered to be backend changes.

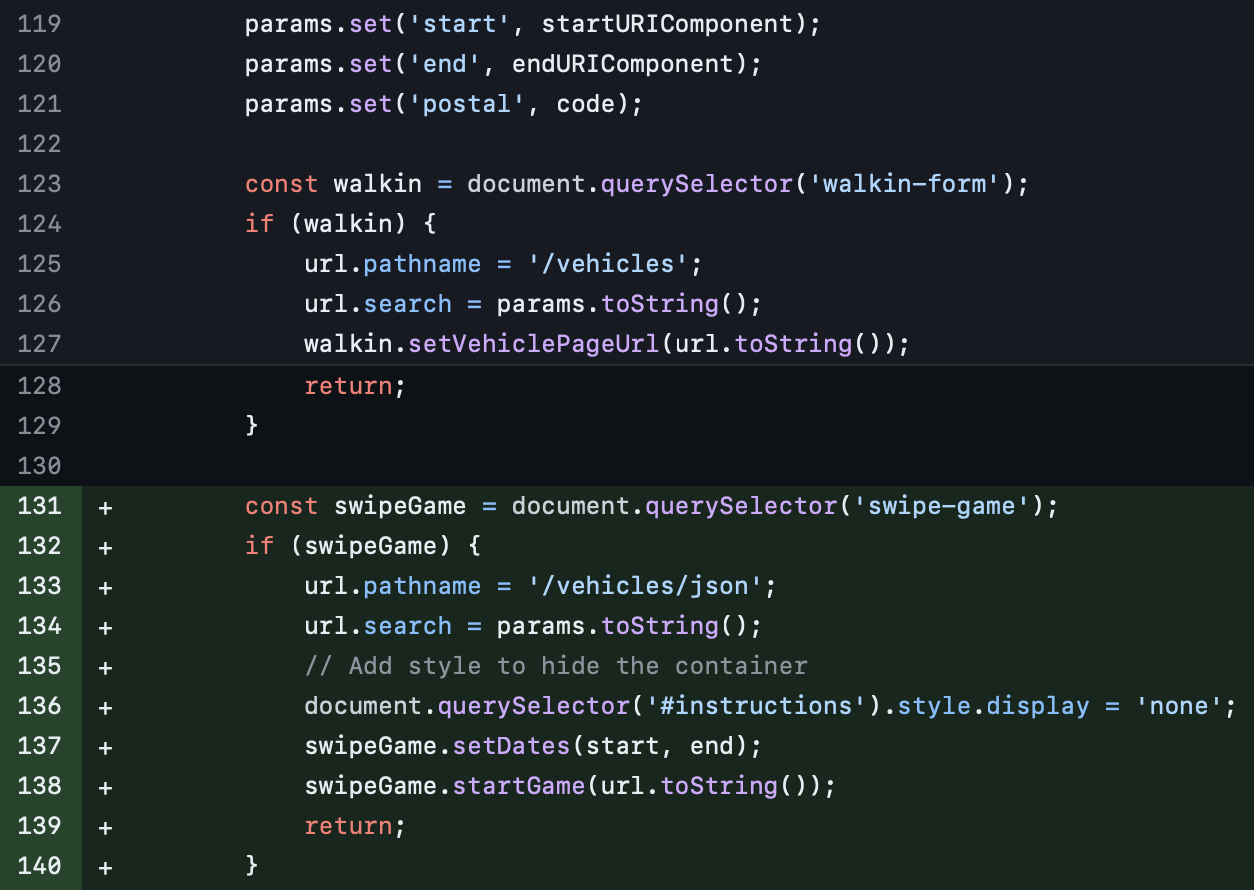
### Coding Standards

Frontend changes appear to mostly follow the guidelines and coding standards established for this project, such as applying naming conventions & properly indenting/formatting code. The weakest example of adherence to our coding standards is the separation of concerns and modularity of the code. Consider for example the following snippet from *datingCard.component.js*:

This is from a function that is intended to apply styling to the datingCard component only if the *match* variable is true. However, it is also defining the styling in question, and thus is essentially assuming the responsibility of both client-side display logic & client-side styling. The styling responsibility would preferably be delegated to a separate, external style sheet for maintainability & structural purposes.

### Readability and Maintainability

Generally, the frontend code for this feature is written such that it is logically structured. Consider for example this snippet from *reservationFinder.component.js*:



Notice how, after setting the query parameters, the decision of where to send the formatted URL is made based on the presence of other elements in the document object model (DOM). This clearly communicates how the reservationFinder component behaves differently on different pages, and sets designated code blocks for these different execution environments.

However, despite allowing us to reuse the same component in several places, the approach taken here is actually a weak, indirect form of dependency injection. The snippet above is not necessarily maintainable, as changes in the DOM surrounding this component may break its behaviour.

A more consistent, modular, and maintainable solution would be to have surrounding components specify a callback function for the reservationFinder component to run, instead of relying on the reservationFinder to decide its behavior based on the existence of DOM nodes.

Further, although we’ve established that the logic is clear and thus readable, the exact contents of certain variables could be clearer. The variables *code* and *walkin*, for example, do not efficiently communicate their purpose nor their contents unless we visit the lines wherein they are declared & defined.

### Functionality and Requirements

The code does meet the functional requirements specified in the user story, as it presents users with a car that best matches the ideal vehicle profile they’ve built while playing the matchmaker game.

Still, there is some opportunity for the exact inner workings of the logic used to suggest a vehicle to be made more explicit to the user. One solution to this problem would be to provide users with a visual representation of their profile being “built” as they *like* and *dislike* vehicles. Whether there is time to implement such a visual before sprint 4 is due is unlikely, due to the relatively large scale of such an addition.

### Error Handling

Frontend error handling is mostly well done, with calls made to the global toast popup to signify invalid inputs and server errors. One example is shown below.



However, as in the snippet above, error messages could be made more specific. Increased specificity would improve developer experience (when debugging a server error) and user experience (when justifying an invalid input).

### Performance Considerations

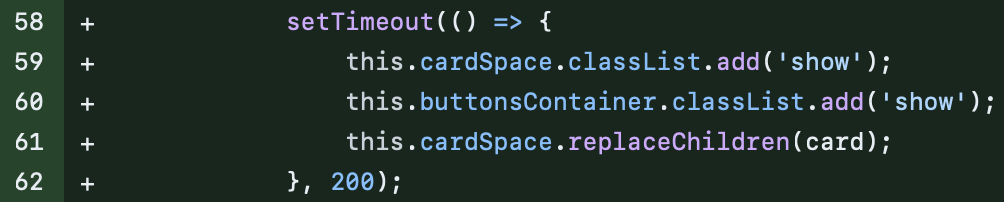
Generally, the code is efficient and performance isn’t a serious concern in this review. However, there still exist opportunities for slight improvement. For example, frontend inefficiency exists in the operation of the matchmaker page (whose logic resides mostly in the swipeGame web component definition). Specifically, Consider the snippet below.



This snippet runs when a winning card has been found. Notice how we remove the visibility from the buttons container, cardSpace, and loadingOverlay even though these will no longer be needed on the page. Thus they continue to exist in the DOM, using resources & memory, despite no longer being visible and no longer being necessary.

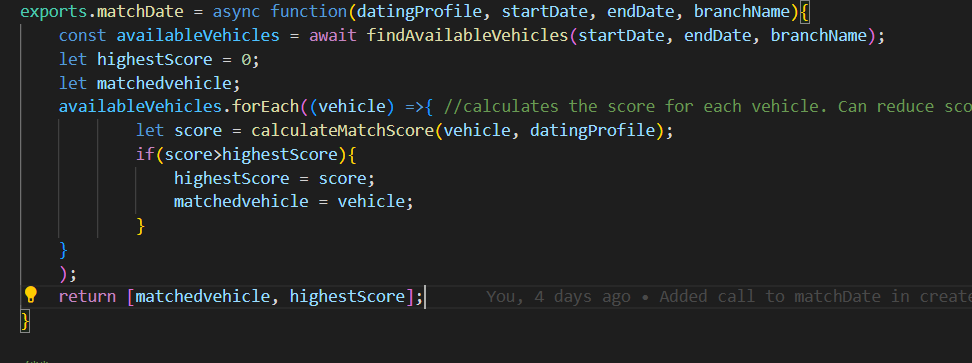
Performance of the page (especially in terms of memory) could be improved by removing (deleting) these nodes from DOM, as opposed to just making them invisible.

Similar performance considerations exist in how the game proceeds, where each new card is an entirely newly rendered component. Notice how we replace the child node each time a card is *liked* or *disliked*:



Since distinct dating card components share the exact same structure, we should instead be altering the attributes of a single dating card instead of instantiating & rendering several throughout the course of the game.

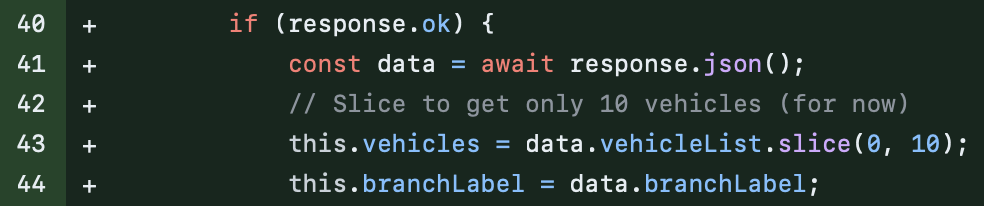
Here, the function runs on O(n2) time complexity which is not ideal. There are a few ways to ensure performance: reduce size of available vehicles by using mongodb indexing for cars that we know will be the best match (strict preferences), or using a sorting algorithm with better time complexity.



### Security

There are no noticeable frontend security considerations, especially since there is no handling of sensitive or even protected data in this user story. Users mustn’t even be signed in to try the new vehicle matchmaker feature.

That being said, basic security practices dictate that we should not send more vehicle data to the client-side than what is necessary. Thus, when we send the initial vehicles that will either be *liked* or *disliked*, we should only send as many vehicle JSON objects as is necessary (that is, a maximum of 10). Instead, more than 10 may be sent to the client side where only the first 10 are considered (and the rest discarded):



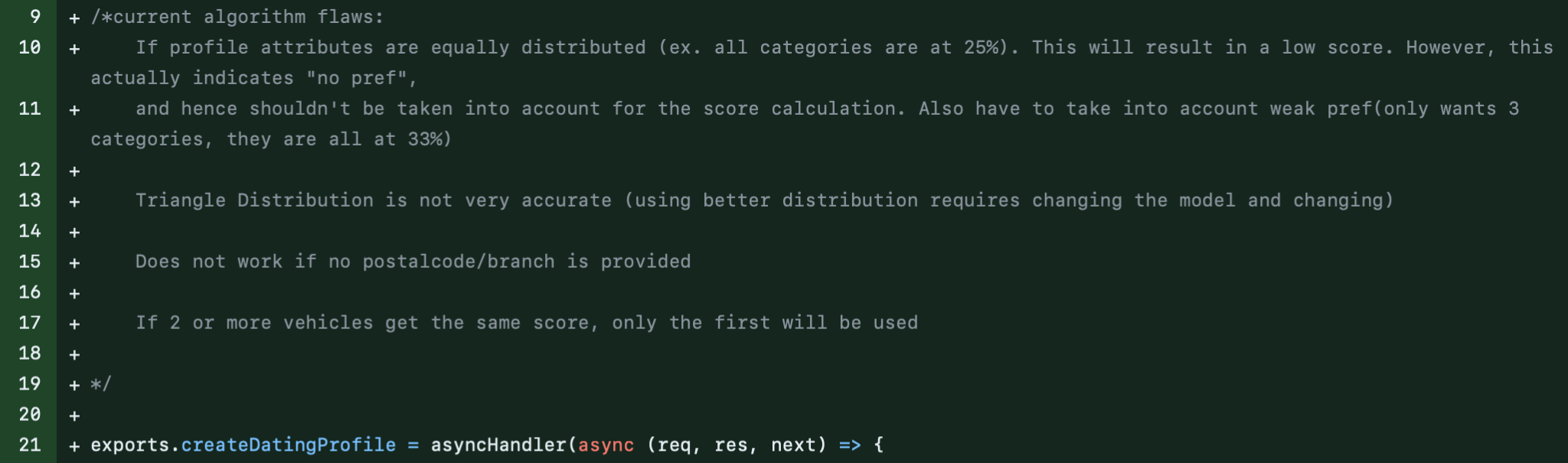
It should be noted that only sending what is necessary would also have the benefit of slight improvements in performance.

### Testing

Existing tests still pass, and new tests have been written for the dating controller methods that respond to exposed endpoints.

### Comments and Documentation

Helpful comments have been left where necessary. Documentation on the algorithmic workings of the backend functionality for this feature were placed in code as comments. However, it would be better to place these specific notes in a dedicated wiki page or markdown page (or any place that is more formal). This is especially true for the comments in the dating controller file that list areas of improvement for the matchmaking logic:

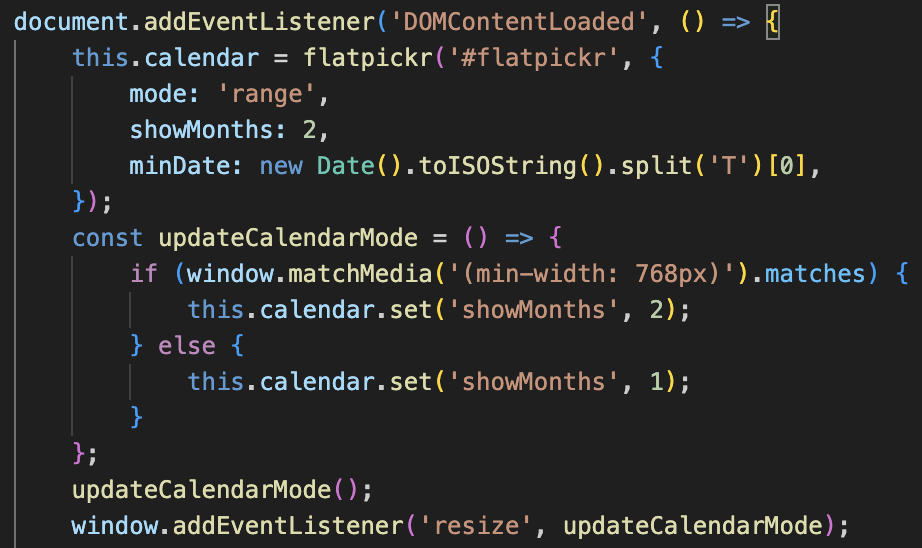


In effect, these points are more difficult to track and modify when writing them directly in source code. One alternative would be to create issues for each, or to add to the spreadsheet that we use to track the backlog.

### Code Duplication

There are no egregious examples of code duplication in this pull request. However, since changes have been made to the reservationFinder component, this PR could’ve taken the opportunity to resolve the following code duplication.

reservationFinder.component.js:  


bookingForm.component.js:  


reservationForm.component.js:  


Since our project’s use of the *flatpickr* library is consistent across pages, the above duplication instances could be refactored into a separate component that “wraps” the *flatpickr* library. In all fairness, this was not so much in the scope of the new feature.

### Version Control Best Practices

For the most part, each commit represents a logical and atomic change. There are still a few exceptions to this rule that have been observed and should be avoided in the future, such as the following commits (d6309db9680829d8df9119547befdf234d32ead7 & c1fe3ad3e36b90e4fd16ffbcee119df4199fbf7c):



It should be noted however that just about all commit messages were meaningful, which is a success on the part of all team members that participated in this user story.

### Dependencies

No new dependencies were introduced for this user story, minus a front-end library that is only included on the client-side via a CDN:



The library in question is *js-confetti*, which itself has no dependencies and presents no security vulnerabilities.

### Conclusion

In conclusion, the changes considered in pull request #290 to implement the vehicle matchmaker feature introduced an innovative approach to vehicle selection by letting users interact with the app after a series of likes and dislikes. Our backend functionality will then find the “perfect” vehicle corresponding to their needs. The backend and frontend components of this feature have been reviewed for coding standards, readability, maintainability, functionality, error handling, performance considerations, security, testing, comments, documentation, version control practices, and dependencies.

This review highlights that while the feature meets the functional requirements and adheres to coding standards, there are still areas for improvement such as code modularity, maintainability, and specificity of error messages. Performance considerations would suggest potential improvements in DOM manipulation and data transmission. Security aspects are adequately addressed, and the addition of new dependencies is minimal. The review emphasizes the importance of continuous improvement in code quality and efficiency, such that new features like the vehicle matchmaker can exceed expectations.