Project Approach and Technology Stack Selection

1. Project Overview

1.1 Project Objectives

The objective of this project is to become familiar with the software development process by developing a middle-fidelity prototype of a car rental application. This is why Agile development (4 sprints) will be the software development methodology to implement the web application.Utilizing GitHub's suite of tools for version control, bug tracking, and task management this project will help us experience the full project lifecycle as well.

The Car Rental Application that is the objective of this project will act as a platform connecting customers seeking rental vehicles with the car rental company offering these services. Users can browse through vehicles that they want to rent and reserve them through the platform. Other features will be implemented to help facilitate the reservation process for the customers as well.

1.2 Scope

The scope of this project spans approximately 10 weeks, with the development process divided into four iterations (four sprints). The initial two weeks of the first sprint will be dedicated to training and setting up the development environment. After that, each sprint will last 3 to 4 weeks until we complete our project.

Main features and functionalities:

- Catalog of rental vehicles with filtering options.
- Option to start a reservation by providing location, pickup/return dates, and select vehicles based on type, category, and price range.
- Customize reservations: Add extra equipment during the reservation process.
- Manage reservations: View, modify, or cancel reservations.
- Find a branch: Locate the nearest rental branch by providing a postal code or airport location.
- Rating and review: Provide feedback and ratings for rented vehicles and the overall rental experience.
- Phone alerts: Customers can get phone alerts for their reservations via SMS.

- Customer Service Representatives can create new reservations if necessary.
- Customer Service Representatives can verify customer reservations and identification
- Customer Service Representatives can proceed with rental agreement review and payment processing.
- Customer Service Representative can confirm returns.
- System Administrator can manage vehicle listings, including creation, updating, and deletion.
- System Administrator can administer user accounts, including creation, updating, and deletion.
- System Administrator can manage reservation data, including creation, updating, and deletion.

1.3 Target Audience

The customers would be individuals looking to rent vehicles, mostly for shorter periods of time. This will mostly include people who are traveling, business professionals who need it for commuting: In short, people who don't own cars and need one. Also, people working using the web application can be considered as a target audience as well. These are people such as customer service representatives (CSRs), employees responsible for assisting customers with the rental process and system administrators, personnel responsible for managing the application and its resources

Customer

Needs to be able to view a catalog of available rental vehicles and to reserve vehicles whilst specifying location, pickup and return dates. They should also be able to add additional equipment at additional prices if wanted. Being able to filter through the cars based on different criterias such as type, category, and price range is also essential. Needs to be able to access reservation details and make changes or cancellations. To know where they should pick up the car they need to be able to locate the nearest branch by providing a postal code or airport. To keep track of their reservation and to be able to receive updates these customers also need SMS phone alerts. And finally, customers leave ratings and feedback to express their opinions and experiences with the service so they should be able to provide feedback and ratings for rented vehicles and the overall rental experience.

Customer Service Representative

- Needs to be able to create new reservations in the system. Also, they need to be able to proceed to the rental agreement review and payment processing after being able to confirm the reservation in the system following the customer's identity verification.
- Check-out Process: Inspect rental vehicles, review rental agreements, process billing, and confirm return completion in the system. Needs to be able to process the final billing, perform the payment settlement and confirm the completion of the return process in the system

System Administrator

Needs to be able to manage the web application so:

- Needs to be able to perform CRUD Operations on Vehicles: Create, read, update, and delete operations on vehicle information.
- Needs to be able to perform CRUD Operations on User Accounts: Manage user accounts, including creation, modification, and deletion.
- Needs to be able to perform CRUD Operations on Reservations: Handle reservation data, allowing for creation, modification, and deletion of reservations.

2. Project Approach

2.1 Development Methodology

Due to the project requirements, the car rental website will be accomplished by using the Agile methodology. Using this development methodology, the team can adapt quickly to any needed changes given by the customer after every iteration. Additionally, there will be regular meetings amongst team members to tackle any possible problems as early as possible which will mitigate risks and ensure that all team members work efficiently.

2.2 Project Timeline

Week 1-2: Sprint 1 - Setup

Milestone: Environment Setup

- Setup development tools and environments.
- Introduction to project management tools (GitHub,).
- Getting used to Agile methodology

Week 3-4: Sprint 2- Core Feature Development

Milestone: Development of Core Features (Start Reservation, Browse Vehicles)

- Design and develop the user interface for browsing vehicles and starting a reservation.

Week 5-7: Sprint 3 - Enhancing Features and Initial Testing

Milestone: Enhancement of Core Features and Initial Prototype Testing

- Add features for viewing, modifying, and canceling reservations.
- Conduct user testing with a focus group;
- gather feedback.
- Develop and integrate additional features based on Sprint 2 feedback (e.g., rating and review system, find a branch).

Week 8-10: Sprint 4 - Finalization and Presentation

Milestone: Final Prototype Development and Project Presentation

- Polish UI/UX based on feedback and testing results.
- Finalize all pending functionalities and ensure integration is smooth across the platform.
- Prepare and conduct a comprehensive project presentation showcasing the prototype, design choices, technologies used, and lessons learned.

Key Deadlines:
End of Week 2: Complete training and setup; finalize project plan and backlog.
End of Week 4: Complete the development of core features
End of Week 7: Complete feature enhancements and conduct focused user testing.
End of Week 10: Complete final prototype development; prepare for and deliver project presentation.

2.3 Collaboration and Communication

As stated, communication amongst team members is important for the development of this project. The project will be accomplished on Github where every team member will submit their accomplished tasks. For communication, the moodle private forum will be used for meeting scheduling and any quick questions where meeting won't be needed. Meetings will occur in person at the university or on Zoom where in-person meetings won't be a possibility. Every meeting will be recorded on a meeting minute for tracking and reviewing.

3. Technology Stack

3.1 Backend Frameworks

3.1.1 - Framework A (**Flask**)

Overview:

Flask is a lightweight web application framework that extends Python. Renowned for its simplicity, Flask provides essential features for building web applications, minimizing complexity. It adheres to the Web Server Gateway Interface (WSGI) standard, acting as a bridge between web applications and servers. As a micro-framework, Flask offers flexibility and a modular design, empowering users to selectively integrate extensions for specific functionalities.

Rationale:

As mentioned in the description, the rationale behind choosing Flask is that first, Flask offers all the essentials related to the development of web applications, all while keeping the complexity level low. Additionally, Flask allows it to be tailored to the developer's need by offering the ability of implementing the appropriate extensions and functionality. Furthermore, Flask is a widely used open-source python framework with a wide community, offering the appropriate help.

As for the performance, Flask is a lightweight framework that uses a single-threaded nature which does not make it the most performant framework compared to the likes of Node.js. However, for a medium size application that handles fewer requests at a given time , makes Flask a great choice performance wise. As for the security, the barebone of Flask does not offer security implementations, however, it easily allows the implementation of plugins and extensions that aid on the security aspect. Lastly, Flask's modular nature and simplicity allow for easier maintenance and allow for tweaking specific elements without compromising the whole application.

Qualitative Assessment

Strengths:

- Flask's simplicity makes it a great strength, as it first allows it to make it easy to learn, and the straightforward design makes it accessible and usable by varying skill levels.

- Additionally, Flask allows for flexibility by allowing the user to implement custom plugins and extensions in order to accurately target the software's needs.

- As Flask is one of the most widely used frameworks for python, It has an extensive community with well documented easing the process of troubleshooting and working with the framework.

- Lastly, Flask's modular nature allows the scalability and easy maintenance of the web application.

Weaknesses:

- One of Flask's strengths can also be considered as its weakness. Its simple nature and lack of features in its base package, can be a weakness for a developer that desires more functions and features included inside the base framework. Additionally, if there are specific functionalities that the developer desires to implement in the application, the use of a third party application will be required.
- Flask is a good choice for smaller and medium scale applications. However for large scale applications, Flask starts to fall behind as the performance is less optimal, and lacks tools required in larger scale applications.

3.1.2 Framework B (**Spring**)

Brief Overview:
The Spring Framework is an open-source backend framework that uses the Java language .It provides comprehensive infrastructure support for developing Java applications. Spring handles the infrastructure so developers can focus on their application.

Rationale
Justification for Choosing Spring:

- *Knowledge*: All team members are already familiar with the Java language because of the courses taken in university (object-oriented programming 1-2)
- *Learning Curve:* Although learning the Spring framework may be complex. It it very popular in the current market and is a useful skill to learn

Qualitative Assessment
Strengths:

- A lot of ressources : extensive documentation, a vast collection of guides, and a supportive community. A lot of youtube tutorials, blogs ,forums
- Security (for car rental transactions)
- Transactional Management
- Support for Testing: With Spring Test module, mock objects and testing frameworks are provided to easily test the application without setting up an entire Spring context.
-

Weaknesses:

- Complexity: Spring has a lot of features which can be confusing

Use Cases:

- Enterprise Applications
- Web Applications

3.1.3 Framework C (**Django**)

Brief Overview:

The Django Framework is an open source, full-stack Python framework that enables rapid development and ensures the security and maintainability of websites.

Rationale

Justification for Choosing Django:

- *Documentation readability and accessibility*: Django's documentation is known for its clarity and availability of ready-to-use code to incorporate into one's own. Since most team members have never worked with frameworks before, a framework that is well documented would increase the rate of development and facilitate the retrieval of information and the implementation thereof.
- *ORM (Object-Relational Mapping)*: The Django ORM simplifies database interactions by allowing developers to work with database models using Python code, reducing the need for writing complex SQL queries which would help with implementing the CRUD operation needed for our web application. Also, since most team members haven't had much exposure to using and implementing databases, having a simplified database interaction would make working on the project easier.
- *Admin Interface*: Django's admin interface provides a powerful tool for managing website content and data models with minimal additional development effort. If Django is implemented, this feature would be extremely useful for managing and developing the CRUD operations on the system admin's side.
- *Compatibility*: Since most team members have some experience with developing with Python, it would make sense for us to use a Python framework.

Qualitative Assessment

Strengths:

- *Security:* Provides already-set-up user user authentication features, that includes: Clickjacking, XSS, CSRF, and SQL injection protection, as well as HTTPS
- *Versatility*: Allows for complex data analysis and the integration of machine learning.
- *Speed and Scalability:* facilitates scalability
- *Community Support*: Facility in finding solutions to common problems and availability of seasoned professionals over the www.
- *Performance Optimization*: Django's built-in caching mechanisms and optimization tools contribute to faster website loading times and improved scalability. This feature is not a priority.

Weaknesses:

- *Complexity*: Full of complicated features and thus can be an overkill for smaller scale projects.
- *Learning Curve*: Due to its extensive feature set and conventions, Django has a steep learning curve for beginners, requiring time and effort to master its concepts and best practices.
- *Overhead*: While Django offers many built-in features, not all of them may be necessary for every project, potentially adding unnecessary complexity and overhead to smaller-scale applications.

Use Cases:

- *E-commerce Platforms*: Django's robust security features, built-in authentication, and scalability make it well-suited for developing secure and scalable e-commerce platforms.
- *Content Management Systems (CMS):* The Django admin interface and ORM capabilities make it ideal for building custom CMS solutions that require flexible content management and easy administration.
- *Data Analytics Dashboards*: Django's support for complex data analysis and integration with machine learning libraries makes it suitable for developing data analytics dashboards and reporting tools.
- *Collaborative Web Applications*: Django's user authentication features and support for real-time collaboration tools make it a suitable choice for building collaborative web applications such as project management platforms or social networks.

3.2 Frontend Frameworks

3.2.1 Framework X (**Svelte**)

Description and rationale
Svelte is a modern JavaScript framework for building user interfaces that prioritizes efficiency and simplicity. Unlike traditional frameworks like React or Vue, Svelte shifts much of the work to compile time, resulting in highly optimized and lightweight code at runtime. This approach makes Svelte an attractive option for developers seeking performance gains without sacrificing development speed or flexibility.

Strengths

- **Efficiency**: Svelte compiler analyzes and optimizes code during build time, resulting in highly efficient and performant applications. This approach eliminates the need for a virtual DOM, leading to faster initial load times and smoother interactions.
- **Simplicity**: With its minimalistic syntax and straightforward approach, Svelte offers a gentle learning curve for beginners and experienced developers alike. The absence of complex abstractions, such as virtual DOM or JSX, streamlines the development process and encourages cleaner, more maintainable code.
- **Bundle Size**: Svelte compiler generates highly optimized JavaScript code, resulting in smaller bundle sizes compared to other frameworks. This advantage is particularly beneficial for applications targeting low-bandwidth environments or aiming to improve load times.
- **Reactivity**: Svelte's reactive programming model simplifies state management by automatically updating the DOM in response to changes in data. This approach reduces the boilerplate code typically associated with manual state management, leading to more concise and expressive applications.
- **Scoped Styles**: Svelte's built-in support for scoped styles allows developers to encapsulate CSS within individual components, preventing style leakage and promoting better maintainability.

Weaknesses

- **Limited Ecosystem**: While Svelte's ecosystem is growing rapidly, it may still lack some of the extensive libraries and tools available in more established frameworks like React or Vue. Developers may need to rely on community-supported solutions or build custom integrations for specific functionality.
- **Learning Curve**: Although Svelte's simplicity is a strength, it may pose a challenge for developers accustomed to the conventions of other frameworks. Learning new concepts such as reactive declarations and Svelte's unique component lifecycle may require an adjustment period.
- **Tooling Overhead**: While Svelte's compiler handles much of the optimization process, setting up a development environment may require additional tooling and configuration. Integrating

features such as hot module replacement or code splitting may involve manual setup compared to frameworks with more integrated tooling solutions.

- **Community Support**: While the Svelte community is vibrant and growing rapidly, it may still lack the depth and breadth of resources available for more established frameworks. Developers may encounter challenges finding answers to specific questions or troubleshooting issues, particularly for niche use cases.

Use Cases

Svelte is well-suited for a variety of applications, particularly those requiring high performance, minimal bundle sizes, and a straightforward development experience. It excels in scenarios such as single-page applications (SPAs), progressive web apps (PWAs), and projects with strict performance requirements. Additionally, Svelte's reactivity and component-based architecture make it an excellent choice for applications requiring dynamic user interfaces or complex data interactions.

3.2.2 Framework Y (**React**)

Description and rationale

React is a user interface library written in JavaScript. Its architecture is component-based, meaning that reusable components are used to build user interfaces. React is our choice because of its efficiency and widely used. Also, React's component-based architecture encourages maintainability and code reuse.

Strengths

- Efficient and fast
- Strong community (a lot of resources and libraries)
- Code reusability

Weaknesses

- Less built-in features compared to frameworks like Angular.

Use Cases:

It would be applied to programs that need dynamic, interactive user interfaces. It could also be used for applications where it's necessary to have reusable user interface elements for its application.

3.2.3 Framework Z (**Angular**)

Description and rationale

Another front-end framework is Angular. This platform is for creating desktop, mobile, and online applications. It offers a methodical approach to application development that prioritizes testability, modularity, and dependency injection. Because Angular is popular and makes use of TypeScript and HTML, we decided to add it to our list.

Strengths

- Built-in features for building complex applications, this makes the process easier.
- TypeScript (static typing, improved developer tooling, enhanced code quality)
- Lot of libraries and tool that makes the task easier

Weaknesses

- Angular has frequent updates

Use Cases:

- Enterprise Applications
- Web Applications

4. Integration and Interoperability

<u>4.1 Backend-Frontend Integration</u>

In order to promote communication between the front end: React, a javascript framework, and back end: Flask, a python framework, of our application, we have decided to use RESTful APIs and the Microservices Software Architecture(MSA). RESTful APIs will offer simplicity, scalability, and the stateless interaction between our stacks. MSA provides scalability, fool proofness, rapid and horizontal development, and the ease of testing, just to mention a few. Scalability allows for the seamless addition of new features into the system. Backed by the independent nature of a service, each microservice can be created and pushed in parallel. What's more, the independence also allows for the ease of testing since each microservice provides one functionality – for which unit tests can be created on the spot. This helps with the alleviation of hard-to-locate bugs in the future. The microservices architecture provides durability in the sense where the operation of the application won't be jeopardised due to the failure of a single microservice. What's more, REST API implements the HTTPS protocol for the transmission of hypermedia documents. HTTPS is a more secure version of HTTP as it encrypts the messages, providing security. In conclusion, the combination of REST APIs and the Microservices

Software Architecture provides a robust system with a knack for scalability, parallel programming, and message encryption.

For the choice of the Database, PostGreSQL has been selected for its powerful features and reliability. It offers strong support for complex queries, ensuring efficient data retrieval and manipulation. Data integrity is guaranteed by PostGre's ACID compliance. ACID compliance is a property that ensures the integrity and security of transactions which is crucial in managing database entries. Additionally, PostgreSQL supports concurrent connections, making it suitable for handling multiple users accessing the application simultaneously. The extensibility of PostgreSQL allows developers to incorporate custom functions and extensions, tailoring the database to the specific needs of a car rental platform. Furthermore, its active community and regular updates contribute to the security and stability of the database, providing a solid foundation for a scalable and high-performance car rental web application.

4.2 External APIs

As for the external APIs, multiple could be implemented. Google maps API will allow you to visually show the location of the closest car rental location. What's more, given the user's current location, a microservice could be developed to provide the shortest route to the branch. A calendar API could be implemented using Google Calendar to sync with the client's personal calendar. Email or SMS notifications microservice will come in handy, for once waitlisted the client must be notified of the availability of the car. Lastly, to process payments an API must be generated. Requirements for safe and secure transactions should be met in order to protect the funds. More APIs could be developed in order to upscale this web application.

5. Security Considerations

Front End

For a software application, different security measures need to be considered for both the front end and the back end. Firstly, for the front end, the measures that can be implemented in the application can consist of establishing secure communications, secure password implementations for accounts, secure session implementations, and appropriate error handling.

In the first instance, establishing secure communications between the front end and the back end can be done by using the appropriate API. More precisely, an API that uses encrypted HTTPS request and response, instead of the typical HTTP. In fact, HTTPS allows a layer of encryption in communications, contrarily to using plain text in HTTP messages. HTTPS allows the protection of data and inhibits third parties from accessing the data. Secondly, for the security of the user accounts, proper password handling can be implemented. Password policies can be implemented to ensure the user creates a strong password respecting certain password policies. Additionally, a client-side input validation can be implemented in order to provide feedback to the user and ensure that the input can be confirmed. Thirdly, managing the session's security by implementing timeouts for session activity. Lastly on the front end, establishing custom error handling, in order to limit the information displayed to the user from the log errors, allows sensitive error information to be displayed to potential threat actors.

Back End

As for the back end, different security measures can be put in place. Firstly, a strong authentication mechanism such as two-factor authentication, as well as a token-based authorization to manage permissions. Second, secure storage can be implemented for storing sensitive information such as passwords. Lastly, tied with the front end, the use of secure APIs that have the appropriate access controls and authentication.

The first security implementation is the addition of a multi-factor authentication procedure. This would allow an extra layer of security to reduce the risk of an authorized party to access an account. Additionally, a token-based approach adds security to the application. In fact, it allows the creation of encrypted tokens to verify users to the website application. The second security consideration is to appropriately implement sensitive information management, such as passwords. First, to store passwords safely, the use of databases is of interest, as it adds a layer of security compared to using plain-text files. However, as for how to manipulate the password before storing it, the options are either encrypting, or hashing the password. For maximum security, hashing the password is more appropriate than encrypting, since hashing is a one-way function, while encryption allows, with the appropriate key, to decrypt the encrypted password. The last security measure for the back end is to make use of secure APIs that allow the appropriate access control and authentication. In fact, this allows it to protect the data, prevent third party access to the system, and establish user identity.

6. Conclusion

To conclude, the car rental website project will utilize Agile methodology to ensure adaptability to customer requirements. The timeline spans 10 weeks, divided into five sprints. The initial sprints focus on setup, core feature development, and enhancement, including user testing for feedback. Later sprints incorporate additional features and integrations based on feedback. Key milestones include the completion of core features by week 4, focused user testing by week 6, and final prototype development by week 10. The project culminates in a comprehensive presentation showcasing the prototype, design decisions, technologies employed, and project insights. The technology stack includes React as the frontend framework, Flask as the backend framework, and PostgreSQL as the database. The integration between React and Python is facilitated by the implementation of the Microservices Software Architecture supported by RESTful APIs. This selection ensures seamless communication between frontend and backend components, facilitating effective development and integration of features within the car rental website project.