SvelteState's Project Approach and Technology Document

By:

Nathan Grenier, David Carciente, Nirav Patel, Annabel Zecchel, Jeremy Crete, and Brian Tkatch

40250986, 40247907, 40248940, 40245507, 40246576, 40191139

A report submitted in fulfillment of the requirements of SOEN 341

Concordia University

February 2, 2024

Contents

1. Project Overview

1.1 Project Objectives

The main goal of our car rental website is to provide the user with a seamless/ user-friendly platform for renting cars. Leveraging Agile methodology and utilizing GitHub for seamless collaboration, we aim to create an efficient service that can aid both individual and corporate clients. The key goals of the project are the following:

1. User-Friendly Interface: Creating a clean and easy interface to navigate allows the user to browse, select and rent cars effortlessly.

2. Data Security: Implementing a robust security system to protect personal data such as credit card information is crucial to maintain trust.

3. Availability across Devices: Implements and endure that the car rental website is accessible and adaptable to different screen sizes.

4. Integration with maps: Since we will be creating a web-based car rental application we will need to integrate with maps. This integration is crucial since it will be easier for the user to find the rental location and drop off points.

1.2 Scope

The scope of our car rental website is composed of various features and functionalities aimed in creating a comprehensive and efficient service. Here are some of the key features:

1. Booking System: A secure and user-friendly booking system allowing our customers to reserve vehicles for a specific date and time.

2. Vehicle Information: Detailed information about each vehicle, including manufacture, model, year, images, and availability status.

3. User Authentication: Include a robust password requirement and email verification for signup and login. The users will be able to manage their accounts easily, with secure options for password resets.

1.3 Target Audience

The target audience for this car rental service web application is for individuals and businesses in need of temporary transportation solutions. The car rental service is useful in many ways. For example, tourists may want to rent a car to explore new cities, friends may want to rent a car for a road trip, and even locals may need to rent a car for a special occasion, or during their own vehicle's maintenance periods. Furthermore, the service appeals to those who prefer the flexibility of renting a car over the commitments associated with ownership, such as young adults, or people living in urban areas with limited parking.

2. Project Approach

2.1 Development Methodology

The agile methodology has been selected as the development methodology for this project. Firstly, the iterative development structure is ideal for the development of core features such as starting a reservation, viewing/modifying/canceling a reservation, and browsing vehicles for rent. These features are based on user interaction, and Agile allows for their continuous improvement based on testing and user feedback. The iterative nature also easily integrates testing into the cycles.

Secondly, Agile offers the flexibility to prioritize certain functionalities, like the CRUD operations, in the backlog. Addressing the most critical aspects first based on their value and impact on the overall project is important in this project.

Thirdly, the Agile methodology has a quick response to changes. Being a dynamic project, we need a methodology that can adapt quickly to changes. For example, there is the possibility that a feature or requirement is changed. Agile uses iterative cycles, which allows the team to implement changes quickly, and not have to wait a long time for the next cycle.

Fourthly, the Agile methodology supports early and incremental deliveries of a product. This allows the team to produce a product quickly for the customer to view. This facilitates the swift implementation of user feedback and requirement changes.

Lastly, the Agile methodology encourages smaller teams, which facilitates continuous and transparent communication between team members. This ultimately contributes to a dynamic and collaborative work environment. Another benefit of being in close communication with other team members is that the risk of mistakes has been lowered. With more team members, there is an increase of perspective and expertise for each task or story.

2.2 Project Timeline

The objective of this project is to complete it in about 10 weeks' time. Since the current development methodology will fall under Agile, the project will contain four different sprints, which align to the requirements provided by the client. Each completion of a sprint will mark a milestone. When completing a sprint, the requirements divided for that sprint indeed satisfy a portion of the client's requirements. After these four sprints are completed, the project should be expected to be delivered. The first sprint will be completed on February 12th, the second sprint will be due March 11th, the third sprint will be due March 25th, and the final sprint is due April 10th, 2024.

2.3 Collaboration and Communication

During the development of the project, communication and disciplined collaboration is imperative for a fluid and successful developmental process. In this regard, there will be five main communication channels to establish distinct cohesion.

1. <u>Informal Communication</u>: Via Discord Chat / Voice Calls

Discord is a group-oriented messaging platform, which provides the ability to create various *channels*, which provide nested groups, which are categorized depending on the need of the major group. In this context, the discord *Main* chat will be used to have general friendly discussions and discuss anything informal about to the project. The user's part of this group are the developers of the project, and not the client nor any mediator. If necessary, a voice channel exists to discuss matters in a vocal sense. Other channels are defined, such as the project components, like the *Front-End*, *Back-End* and *Deployment*. Just like the Main channel, these channels are made to organize and distinguish the informal conversations about the project, but within their respective category. It is important to note that all discussions evoked on this platform are unofficial and used for general brainstorming and prototyping.

2. Semi-Formal Communication – Agile Scrum

A component of the Agile methodology is to have stand-ups, or daily meetings about the overall progression of the project. After ideas are agreed upon on the Discord chat, then a serious discussion is evoked in person, during the scrum meeting. The scrum meeting will baseline all the informal discussions and decide on official directions regarding the project. The outcome is discussed in (3), below. In addition, the scrum meetings will also discuss any progression of the project, after assigned a task to one of the team members.

3. Formal Communication **A** – GitHub Wiki and GitHub Issues

After the semi-formal discussion about the project with a global consensus is discussed during a scrum, then it is time to officially declare it on a platform, which will be available to mediators and clients. The platform used is GitHub, which contains a *Wiki* and *Issues*. The Wiki will hold any formal documentation about the project, such as meeting minutes, technologies used in the project, resources to learn about the technologies used in the project, the overall workflow, and more if needed. This is useful when the client or mediator is interested in communicating the documentation of the project.

The Issues feature on GitHub is an atomic element, which provides a dynamic use case for the project's development. In the context of this project, an issue could either be a Bug Report, a User Story, or a general Task. In all cases, their functionality will contain the ability to set a title, and a body description depending on their type. This is useful when the client or mediator is interested in understanding the overall development of the project.

Overall, both Wiki and Issues allows to keep track of the progression of the project, while communicating them to the client and mediator, which further allows a clear communication.

4. Formal communication **B** - Mediator

During the development of the project, all formal decisions posted on the Wiki or Issues, or informal ideas will be presented and communicated to the mediator. This is to assert that the project development is following the vision of the client.

5. <u>Formal communication **C**</u> – Client

In cases where the mediator is unable to provide information regarding a decision, then the communication can be relayed directly to the client. This is to assert that there is no miscommunication between the group and the client.

In addition to these communication channels, the main collaboration tool used will be Figma and Git. Figma is used to prototype and create wireframes for the development of the project. Figma can collaborate with multiple users in parallel, which allows for a more efficient development of the wireframes. In terms of collaboration of code, the use of Git is implemented, to further allow version tracking of the overall development of the project and allow for users to collaborate to the project as needed.

3. Technology Stack

3.1 Backend Frameworks

3.1.1 No Framework

When developing web applications, opting not to use any front-end framework means abstaining from relying on pre-built libraries or organized structures like React, Svelte, or Vue.js. Instead, our developers would manually handle HTML, CSS, and JavaScript without the aid of a specific framework to structure their code.

Justification for choosing no framework.

Choosing no front-end framework is primarily driven by the desire for complete control over the codebase. The flexibility of writing custom code and tailoring solutions to specific project requirements without adhering to a predefined framework's conventions could be useful in specific situations.

- **Community Support:** Lack of a dedicated framework means relying less on a specific community. However, developers might need to find solutions to problems independently.
- **Scalability:** Without a pre-structured framework, scalability primarily depends on our developer's ability to create efficient and scalable code. The absence of a clear framework could lead to challenges later as the project grows larger.

- **Ease of Integration:** There will most likely be more flexibility in terms of integration with various libraries and tools. However, the absence of a structured framework could result in more effort spent on manually configuring and managing these integrations.

Qualitative Assessment

Strengths

- **Flexibility:** Provides the freedom to structure the code based on specific project needs without adhering to a specific framework's constraints.
- **Performance:** With no layer of abstraction, the final product might be more streamlined which may also lead to better performance.

Weaknesses

- **Reinventing the Wheel:** During development, features that are readably available might end up being recreated leading to increased development time and potential for errors.
- **Maintenance:** Without a framework, code maintenance and updates will most likely be more challenging, especially as the project scales.

Use Cases

- **Small Projects:** Suitable for smaller projects where the overhead of a framework is not necessary or justified and the focus is on simplicity and speed of development.
- **Highly Custom Projects:** Projects with unique requirements that don't align with existing frameworks would require the freedom to design a custom solution.

3.1.2 Remix

Remix is a meta framework designed for building web applications along side React. It extends the capabilities of React and introduces additional features to simplify the development process. Remix emphasizes server-rendered React applications (SSR), offering a blend of client and server-side rendering to enhance performance and user experience.

Justification for choosing Remix.

Remix could be chosen for its ability to improve React development by providing a structured approach to server-rendered applications. The framework integrates extremely well with React components and offers features that enhance the developer experience, such as route-based code splitting and tightly integrated data fetching.

- **Community Support:** Remix benefits from the larger React community. Because of this, any developer can find support and resources within the broader React ecosystem.
- **Scalability:** Remix's server side rendering focused architecture would lead to improved scalability by optimizing the initial load time and enhancing the overall performance of web applications.
- **Ease of Integration:** Remix should integrate seamlessly with any existing react applications.

Qualitative Assessment

Strengths

- **Improved Performance:** Remix's focus on server rendering enhances initial page load times, which would contributing to a smoother user experience.
- **Developer Experience:** The framework introduces features like route-based code splitting and built-in data fetching. This would streamline the development process by providing a more enjoyable experience for developers due to its rich features.

Weaknesses

- **Learning Curve:** As Remix introduces new concepts and approaches, developers who are not familiar with meta frameworks will face a learning curve.
- **Community Maturity:** Remix’s community is relatively new compared to more established frameworks. This could lead to fewer third-party libraries and resources tailored specifically for Remix.

Use Cases

- **Server-Rendered Applications:** Remix is well-suited for projects where server-rendered pages can significantly enhance performance.
- **React Ecosystem Integration:** Remix would easily integrate in a react based project, being able to take advantage of the react ecosystem in the process.

3.1.3 SvelteKit

SvelteKit is a meta-framework built on top of the Svelte framework. SvelteKit tries to simplify the development process by providing high level abstractions to developers of Svelte applications. It includes features like routing, server-side rendering, and a plugin system to enhance the development experience.

Justification for choosing SvelteKit.

SvelteKit could be chosen for its seamless integration with the Svelte framework. This would enable developers to gain the benefits of the Svelte frontend framework while also providing additional tools for building scalable and feature-rich web applications.

- **Community Support:** SvelteKit can leverages the growing Svelte community. However, as a relatively newer meta-framework its community might be smaller compared to more established frameworks.
- **Scalability:** SvelteKit inherits the scalability advantages from Svelte which focuses on compiling components during build time.
- **Ease of Integration:** As a meta-framework built on Svelte, integration with Svelte is seamless.

Qualitative Assessment

Strengths

- **Performance:** SvelteKit is very performance due to most of the work being done during build time, resulting in optimized and minimal runtime code.
- **Developer Experience:** SvelteKit simplifies the development process by providing built-in features like routing and server-side rendering, reducing the need for additional configurations and third-party libraries.
- **Great Documentation:** Not only does SvelteKit have highly refined documentation, but it also provides developers with interactive learning tools.

Weaknesses

- **Community Maturity:** Being a relatively new meta-framework, SvelteKit's community might be less mature compared to more established frameworks. This could lead to fewer resources and community-driven plugins available for developers.
- **Learning Curve:** Developers that are new to SvelteKit may encounter a slight learning curve. However, this curve is much less harsh than other alternatives like Remix.

Use Cases

- **Svelte Ecosystem Integration:** SvelteKit is ideal for projects that already leverage Svelte and its reactive approach and component-based architecture.
- **Projects Prioritizing Performance:** Applications where performance is a critical factor can benefit from SvelteKit's build-time optimizations and efficient runtime code.

3.2 Frontend Frameworks

3.2.1 Angular

Angular is a front-end framework used to build dynamic web applications. It utilizes a component-based architecture and is written in Typescript. Angular provides a comprehensive set of tools and features for building robust and scalable single-page applications (SPAs).

Justification for choosing Angular.

Angular would be chosen for its rich ecosystem and extensive features, making it a smart choice for large applications requiring a structured and opinionated framework.

- **User Interface Capabilities:** Angular's component-based architecture allows for the creation of reusable UI components, promoting a modular and maintainable user interface.
- **Responsiveness:** Angular supports the development of responsive web applications through features like Reactive Forms and component libraries like Angular Material.
- **Cross-Browser Compatibility:** Angular is designed to ensure cross-browser compatibility, enabling applications to run consistently on various web browsers.

Qualitative Assessment

Strengths

- **Structured Architecture:** Angular enforces a modular architecture, making it easier to organize and scale large codebases.
- **Powerful Two-Way Data Binding:** Angular's two-way data binding simplifies the synchronization of the model and the view, greatly reducing boilerplate code.

Weaknesses

- **Verbose Syntax:** The syntax in Angular is more verbose compared to other frameworks.
- **Bundle Size:** Angular applications might have larger bundle sizes which could impacting initial load times.

Use Cases

- **Enterprise-Level Applications:** Angular is well-suited for large-scale enterprise applications with complex requirements and extensive feature sets.

3.2.2 React

React is a front-end JavaScript library used to build the frontend. It follows a component-based architecture, allowing developers to create reusable and modular UI components. React is widely used for building single-page applications (SPAs) and is known for its virtual DOM, which enhances performance by minimizing unnecessary updates.

Justification for choosing React.

React could be chosen for its simplicity, flexibility, and a large community. Its component-based structure promotes code reusability and ease of maintenance. React's virtual DOM contributes to efficient rendering, making it suitable for creating dynamic and responsive user interfaces.

- **User Interface Capabilities:** React's component-based approach allows developers to create interactive and reusable UI components.
- **Responsiveness:** React's virtual DOM and efficient diffing algorithm contribute to improved application performance and responsiveness, especially in scenarios where frequent updates are required.
- **Cross-Browser Compatibility:** React applications are designed to be cross-browser compatible.

Qualitative Assessment

Strengths

- **Virtual DOM:** The virtual DOM improves performance by minimizing actual DOM manipulations, resulting in faster rendering and a smoother user experience.
- **Large Community and Ecosystem:** React has an extensive community providing abundant resources like third-party libraries and continuous support for developers.

Weaknesses

- **Configuration Over Convention:** React gives developers more freedom, but this lack of conventions may lead to inconsistencies in project structures and coding styles.
- **Boilerplate Code:** React applications might require additional boilerplate code, especially for features like state management, which can increase the overall codebase.

Use Cases

- **Dynamic and Interactive UIs:** React is well-suited for projects that require dynamic and interactive user interfaces.

- **Large-Scale Applications:** React is a viable choice for building large-scale applications where code maintainability and scalability are crucial.

3.2.2 Svelte

Svelte is a front-end JavaScript framework that differs from traditional frameworks like React or Angular. Instead of running in the browser, Svelte shifts much of the work to compile time, generating highly optimized and efficient JavaScript code. It emphasizes a declarative approach to building components and aims to reduce the amount of boilerplate code required for web development.

Justification for choosing Svelte.

Svelte could be chosen for its simplicity and the innovative approach of moving much of the framework's functionality to compile time. This results in smaller bundle sizes, faster load times, and a more straightforward development experience with less code to write and maintain.

- **User Interface Capabilities:** Svelte provides a clean and concise syntax for building interactive user interfaces. Its reactive approach simplifies state management within components *which helps* to *streamline the* development process.
- **Responsiveness:** Svelte's focus on generating optimized code at compile time translates to improved responsiveness and faster execution during runtime. This results in a better user experience.
- **Cross-Browser Compatibility:** Svelte applications are designed to be compatible with various web browsers.

Qualitative Assessment

Strengths

- **Smaller Bundle Sizes:** Svelte's compilation approach results in smaller bundle sizes, leading to faster initial page loads and improved performance.
- **Declarative Syntax:** Svelte's declarative syntax simplifies the creation of components, reducing boilerplate code and making the codebase more readable.
- **Built-in Transition Effects:** Svelte includes built-in transition effects, making it easy to add animations to elements without the need for external libraries.

Weaknesses

- **Smaller Ecosystem**: The ecosystem around Svelte is not as extensive as some larger frameworks.
- **Limited Control Over Runtime Behavior:** Because much of the work is done at compile time, developers will have less control over runtime behavior.

Use Cases

- **Prototyping and Rapid Development:** Svelte's simplicity makes it suitable for prototyping and projects where quick development cycles are essential.
- **Web Applications Requiring Smooth Animations:** The built-in transition effects make Svelte an excellent choice for applications requiring smooth and elegant animations.

4. Integration and Interoperability

4.1 Backend-Frontend Integration

Shared Syntax and Concepts: Since SvelteKit is built on top of Svelte, they share the same underlying syntax and concepts. It is possible to seamlessly transition to using SvelteKit without encountering a significant learning curve.

Unified Development Workflow: Both Svelte and SvelteKit promote a similar development workflow centered around component-based architecture and reactive programming. This unified approach ensures consistency in code organization and development practices across the frontend components of the project.

Common Tooling and Ecosystem: Svelte and SvelteKit leverage the same tooling and ecosystem, allowing developers to use familiar tools for tasks such as bundling, testing, and debugging. This shared ecosystem streamlines the development process and enhances developer productivity.

Integrated Routing and Server-Side Rendering: SvelteKit introduces features such as routing and server-side rendering (SSR) that seamlessly integrate with Svelte components. These features are designed to work in harmony with Svelte's reactive nature, enabling developers to create dynamic and SEO-friendly web applications with ease.

Optimized Build Process: SvelteKit optimizes the build process by leveraging Svelte's compiler to generate highly optimized JavaScript code at compile time. This approach minimizes runtime overhead and results in smaller bundle sizes, leading to faster load times and improved performance.

4.2 Third-Party Services

- **Prisma:** A tool used for building SQL queries and managing database migrations, aiding in efficient database management.
- **Skeleton UI:** Provides pre-made components and style tokens for the site.
- **Tailwind CSS:** A utility-first CSS framework for rapidly building custom designs directly in markup.
- **Serverless Application Deployment (Vercel):** Deployment platform for our application. Serves our site and hosts it is serverless functions.
- **Serverless Database Hosting (Neon):** Manages multiple PostgreSQL databases for each deployment.

5. Security Considerations

5.1 User Authentication and Permission Control

User accounts for the website are held in a Postgres database. Users are authenticated with an email address and password. Email address uniqueness is enforced by a database constraint. Password data held in the database is stored as a bcrypt hash. Bcrypt is a hashing construct designed for protecting passwords in a database. It has been used in production for more than 20 years and is used by default to protect passwords in OpenBSD. It is an adjustable-difficulty function and is a salted hash to protect against rainbow table attacks in the event that password hashes held in the database became exposed.

Password security depends on maintaining the confidentiality of the password. A minimum length constraint and a requirement to use both numbers and digits in the password will be enforced at creation time. Additionally, users will be encouraged not to use a password that is used elsewhere.

Once authenticated, a session cookie will be set on the user's browser. The app will use stateless sessions that expire in 30 minutes. Accordingly, the session cookie will be a JSON Web Token carrying the user ID, role, and name and email (to prevent excessive database queries), and a cryptographic signature. This signature will be verified on future requests and if valid, the token will be refreshed for 30 minutes from the new request. Accordingly, after 30 minutes of inactivity, the user will no longer have an active session. JSON Web Tokens provide a mature framework for this type of session token and help avoid pitfalls that may be associated with bespoke cryptographic implementations.

The data read and authenticated from the session cookie will be used for authorization. Accordingly, each feature must include checks of a user's role to ensure their level of access is appropriate.

5.2 Transport Security

The app is hosted in the Vercel hosting environment on the vercel.app domain, which is pre-loaded in major browsers for Strict Transport Security. Accordingly, clients can only connect to our app over an encrypted HTTPS connection. Only currently-secure cipher suites are enabled by Vercel.

Vercel.app is listed on the Mozilla Public Suffix List, which constrains cookies issued by our app to our Vercel subdomain. This ensures that cookies remain confidential when browsing to apps hosted on other Vercel subdomains.

Vercel uses VPNs and proprietary technologies to protect communications within its Edge Network, including between the server hosting the application and the server hosting the database.

5.3 Server Security

Our app is hosted in the Vercel “serverless” environment. This means Vercel takes on many responsibilities including server maintenance and patch management. Vercel automatically deploys the app from our GitHub repository, reducing the need to make manual configurations or access server shell prompts. Vercel holds a SOC 2 Type 2 attestation with respect to its change management and security controls. Vercel also provides a customizable firewall and DDoS mitigation.

6. Conclusion

In conclusion, the chosen project approach and technology stack for the development of our car rental website represent a carefully considered combination aimed at achieving our project objectives effectively. Our approach embraces Agile methodology, allowing for iterative development cycles, quick adaptation to changes, and prompt delivery of product increments. This methodology fosters collaboration, transparency, and continuous improvement within our development team.

In terms of technology stack, our decision to utilize SvelteKit, not only as the frontend but also as the backend framework, signifies a strategic move towards enhancing performance, streamlining development, and ensuring seamless integration between frontend and backend components. Leveraging SvelteKit's high-level abstractions and built-in features such as routing, server-side rendering, and plugin system, alongside Svelte's simplicity and efficiency, empowers us to develop scalable and feature-rich web applications efficiently.

Additionally, our selection of tools such as Prisma, Skeleton UI, and Tailwind CSS enhances the development process by providing efficient database management, pre-made UI components, and rapid styling capabilities, respectively.

Furthermore, our emphasis on security considerations underscores our commitment to protecting user data and ensuring a safe browsing experience. By implementing robust user authentication, transport security measures, and leveraging the secure hosting environment provided by Vercel, we aim to safeguard user information and maintain the integrity of our application.

Overall, our project approach and technology stack have been carefully chosen to prioritize user experience, development efficiency, and security. We believe that these strategic decisions will contribute to the successful delivery of a seamless and user-friendly car rental platform that meets the needs of both individual and corporate clients.

Division of Sections	
Name	**Section Numbers**
Nathan Grenier	**3.1, 3.2, 4.2**
Brian Tkatch	**5.1, 5.2, 5.3**
Nirav Patel	**6, 4.1**
David Carciente	**2.2, 2.3**
Annabel Zecchel	**2.1,1.3**
Jeremy Crete	**1.1,1.2**