Justification of Proposed Technologies

To: Business Directors and Engineering Team

From: Lead Software Engineer

Subject: Justification of Proposed Technologies for FreeCycleInc

I am writing this technical report to justify the use of the proposed technologies in our software prototype at FreeCycleInc. The technologies mentioned below have been carefully chosen to enhance the functionality, performance, and maintainability of our application. The report is divided into sections based on the different components of the software, namely the server framework, server language, client framework, and client language. I recommend using Express.js as Server Framework and Vue as Client Framework.

Express.js

Server Framework Features

For the Server framework, I suggest using Express.js as it is a well-liked and commonly used server network framework for Node.js web development. It provides a variety of features and advantages that turn it into a useful tool for developers. A summary of Express.js's benefits as a server network framework is provided below:

1. **Easy and Flexible Routing**

   It provides a powerful and convenient way to handle HTTP requests and define routes for web applications and APIs. This feature simplifies the process of defining routes, handling different HTTP methods (GET, POST, PUT, DELETE, etc.), and managing parameters in URLs simplifies the process of creating clean and organized code, enabling dynamic URL handling, promoting RESTful API design, and allows for seamless integration of middleware functions. This feature significantly contributes to the efficiency and scalability of development using Express.js.It enables logical URL mapping, making it simple to handle dynamic parts of URLs, such as user IDs, product names, or post slugs, without having to define separate routes for each. This feature helps in providing the ability to chain multiple route handlers allows for easy integration of middleware functions. This simplifies tasks like authentication, input validation, and error handling for specific routes.

   Code snippet :

```
app.get(`/item/:id`, (req, res) => {
```

```
  const { id } = req.params;
  if (items.hasOwnProperty(id)) {
    res.status(200).json(items[id]);
  } else {
    res.status(404).json({ message: 'Item not found' });
  }
});
```

2. **Middleware Support**

Middleware functions, or functions with access to the request and response objects in an application's request-response cycle, are very well supported by Express.js. The many uses for middleware functions are endless and include authentication, input validation, error handling, logging, compression, and many more. It enhances the flexibility, modularity, and maintainability of web applications. It allows developers to add and organize functionality at specific points in the request-response cycle, making it a powerful feature for creating robust and efficient web applications and APIs. It allows developers to break down complex application logic into smaller, reusable components, organize different functionalities into separate middleware functions improving code readability and making the application's structure more coherent.

Code snippet :

```
// Enable CORS for any origin
app.use((req, res, next) =>
{
res.setHeader('Access-Control-Allow-Origin', '*');
res.setHeader('Access-Control-Allow-Methods', 'GET, POST, DELETE, OPTIONS');
next();
});
```

3. **Binding a server to a specific port and IP address**

Ports are essential for network communication as they allow different applications running on the same host to distinguish between different types of network traffic.By default, the Express.js application binds to all available network interfaces, which means it can be accessed through any IP address assigned to the server (including the loopback interface, 127.0.0.1). However, you can choose to bind the server to a specific IP address if you want it to be accessible only via that address.By binding each application to a different port, you can run multiple Express.js applications simultaneously without port conflicts. It makes it easier to identify and fix issues during the development process.It helps to more accurately replicate the server configuration and test for any potential issues related to port or IP bindings.

Code snippet :

```
app.listen(port, () => {
console.log(`Server listening at http://localhost:${port}`);


});
```

Server Language Features

1. **In-build array functions**

   It refers to a set of built-in methods and higher-order functions that are available on arrays. These functions allow developers to perform various operations on arrays in a concise and efficient manner, making it easier to manipulate data and perform complex tasks.Using these array functions, developers can perform complex operations on arrays in a more readable and concise way, leading to cleaner code and improved maintainability. In JavaScript, functions are first-class citizens, which means they can be treated just like any other value. This allows developers to pass functions as parameters to other functions, returning functions from functions, and assigning functions to variables. Functions that can accept other functions as arguments are known as higher-order functions.

   Code snippet :

```
app.get(`/items`, (req, res) => {
const { user_id } = req.query;
if (user_id)
{
const filteredItems = Object.values(items).filter(item => item.user_id ===
user_id);
res.status(200).json(filteredItems);
}
else
{
res.status(200).json(Object.values(items));
}
}
```

2. **Let Keyword**

   In JavaScript, the let keyword was introduced in ECMAScript 6 (ES6) as an alternative to the traditional var keyword for variable declaration.

The let keyword allows for block-scoping, meaning that variables declared with let are limited to the scope of the block in which they are defined. This improves code readability and helps avoid common issues associated with variable hoisting.

When using Express.js, developers can leverage the let keyword to declare variables within specific scopes, such as within route handlers or middleware functions. This allows for better control over variable lifetimes and reduces the likelihood of naming conflicts or unintended global variable declarations.

Code snippet :

```
app.get(`/item/{id}`, (req, res) => {
const { id } = req.params;
if (items.hasOwnProperty(id)) {
res.status(200).json(items[id]);
}
else
{
res.status(404).json({ message: 'Item not found' });
```

3. Mutability in Express.js

Mutability refers to the ability to change the value of a variable or object after it has been created. In JavaScript, objects and arrays are mutable, meaning their properties or elements can be modified or reassigned.

In the context of Express.js, mutability is relevant when dealing with request and response objects. These objects contain various properties and methods that can be modified or accessed during the handling of a request. For example, developers can set headers, add data to the response body, or modify request parameters.

Code snippet :

```
app.post(`/item/`, (req, res) => {
const { user_id, description, keywords, lat, lon } = req.body;
if (!user_id || !description || !keywords || !lat || !lon) {
res.status(400).json({ message: 'Missing fields' });
}
else {
const id = Object.keys(items).length + 1;
const newItem =
{
id,
```

```
user_id,
description,
keywords,
lat,
lon,
date_from: new Date().toISOString().slice(0, 10),
};
items[id] = newItem;
res.status(201).json(newItem)
```

Vue

Client Framework Features

1. **Component-Based Architecture**

   Vue's component-based architecture enables developers to create reusable and self-contained components. Components encapsulate HTML, CSS, and JavaScript logic, making it easier to build complex user interfaces. Vue's component system promotes reusability, modularity, and maintainability, allowing developers to compose applications from smaller, manageable building blocks. It treats objects as events which helps the code reusability and modularity.

   Code snippet :

```
created()
{
this.clearItem();
this.getItems();

},

methods: {
clearItem()
{
this.item = {...this.item, ...{
user_id: '',
keywords: '',
lat: '',
lon: '',
image: '',
description: '',
```

```
}};
},
```

2. Reactive Data Binding

Vue employs a reactive data binding system that automatically keeps the user interface in sync with the underlying data model. When data changes, Vue updates the relevant components efficiently without the need for manual DOM manipulation. This feature simplifies the management of data flow and ensures a responsive and synchronized UI.

Code snippet :

```
<div class="container">
<h3>Create an item:</h3>
<form @submit.prevent="createItem">
<div class="form-group">
       <label for="user_id">UserName:</label>
       <input  v-model="item.user_id"  name="user_id"  id="user_id"  type="text"
       placeholder="Name" class="form-control">
</div>
<div class="form-group">
       <label for="lat">Latitude:</label>
       <input     v-model="item.lat"     name="lat"     id="lat"     type="text"
       placeholder="Latitude" class="form-control">
</div>
```

3. Vue Router

Vue Router is a built-in routing solution for Vue.js applications. It provides a client-side routing mechanism, allowing developers to define and navigate between different views or pages within a single-page application. Vue Router offers features like nested routes, route parameters, and route transitions, making it easier to create dynamic and navigable web applications.

Code snippet :

```
<div class="card-body">
       <h5 class="card-title">User Id: {{ item.user_id }}</h5>
       <p>List Id: {{ item.id }}</p>
              <p class="card-text">Keywords: {{ item.keywords }}</p>
              <p class="card-text">Latitude: {{ item.lat }}</p>
              <p class="card-text">Longitude: {{ item.lon }}</p>
              <p class="card-text">Date: {{ item.date_from }}</p>
              <p class="card-text">Description: {{ item.description }}</p>
```

Client Language Features

1. Single-File Components

Javascript allows developers to define components using single-file components (SFCs). An SFC combines the template, script, and style for a component into a single file, enhancing code organization and readability. This approach enables better separation of concerns and facilitates the reuse and sharing of components across different projects. Using the spread operator we can organize and access the content of an iterable data structure such array or dictionary without having to use `for` or `for each()` to loop through its content

Code snippet :

```
data() {

return {
        Item:
        {
                Id:'',
                user_id: '',
                keywords: '',
                lat: '',
                lon: '',
                image: '',
                date_from:'',
                description: '',
        },

items: [],
};
},
created()
{
this.clearItem();
this.getItems();
```

2. Async Processing

Without waiting for the current or prior function to complete, async processing enables us to call the subsequent function to be run. JavaScript's "'then()'" method is used to manage asynchronous processes like API calls. The Promise API defines it. Javascript offers a predefined project structure, programmable build procedures, and integrated development server functionality. By

automating routine processes like testing, dependency management, and project scaffolding, the command-line streamlines the development process and increases productivity.

Code snippet :

```
getItems()
{
	fetch(`${urlAPI}/items`,
		{
			method: 'GET',
		})
	.then(response => {
	if (response.ok)
	{
	return response.json();
	}
	Else
	{
	throw new Error('Failed to retrieve items.');
	}
})
```

3. **Error Handling with try, catch, and throw**

JavaScript provides a way to handle errors using try, catch, and throw. The basic idea is to wrap a potentially error-prone code inside a try block and catch any errors that may occur with a catch block. If an error is thrown, the execution jumps to the corresponding catch block, allowing developers to handle the error gracefully.

Code snippet :

```
.then(response => {
	if (response.ok)
	{
		return response.json();
	}
	Else
	{
		throw new Error('Failed to retrieve items.');
	}
	})
.then(data => {
```

```
this.items = data;
})
.catch(err => console.error(err));
```

Future Technology Recommendations

Currently, the system has no provision to save the data of requests and works session wise hence using database technology would make the system more efficient. In future the load on the system will increase hence the team will need to integrate technology to enhance a faster data retriever and use serverless technology to reduce the probability of physical system failure and reduce infrastructure dependency and costs.

1. **MongoDB**

   Integrating MongoDB, a NoSQL document-oriented database, can be a beneficial future technology recommendation for the existing system. MongoDB offers scalability and flexibility, allowing the system to manage large collections of unstructured data effectively. With its ability to shard data across multiple instances, MongoDB enables horizontal scalability, accommodating growing data needs. The system can also benefit from MongoDB's fault-tolerant architecture, which ensures data redundancy and operational continuity in case of hardware failures. However, it is important to consider the memory requirements of MongoDB and its limitations, such as the lack of support for joins and limited document size. Proper evaluation of data requirements, performance expectations, and the need for complex querying should be taken into account before adopting MongoDB.

2. **GraphQL**

   Implementing GraphQL can be a recommended future technology for the existing system, addressing the challenges of over-fetching and providing more efficient data retrieval. With GraphQL, the system can reduce network requests and minimize payload by retrieving only the required data, avoiding unnecessary data transfers. The client can specify the shape of the response, allowing for customized data retrieval and reducing over-fetching. Additionally, GraphQL simplifies the management of API endpoints by exposing a single HTTP endpoint (/graphql), which enhances the flexibility and maintainability of the system's API. However, it is important to consider the potential difficulties in implementing a simplified cache compared to REST and the learning curve associated with adopting GraphQL. A thorough evaluation of the system's data retrieval needs, performance requirements, and the development team's familiarity with GraphQL should be conducted before incorporating it into the existing system.

3. **Serverless (AWS)**

   Implementing a serverless architecture using AWS can be a recommended future technology for the existing system. By leveraging serverless computing, the system can benefit from the advantages of offloading server infrastructure management to the cloud service provider. This

allows developers to focus on developing their programs without worrying about server maintenance. The system can scale up or down automatically based on the application's demand, ensuring optimal performance and cost efficiency. However, it is important to consider the potential drawback of increased costs as the business grows and the limitations of being tied to the AWS ecosystem. Therefore, it would be prudent to assess the long-term scalability and cost implications before adopting a serverless approach.