The Node.js and NPM/Yarn section suited the specifics of our project:

---

# Node.js and Package Managers Setup

Requiring the use of NVM (Node Version Manager) in our project is a practical approach, our project consists of multiple submodules that may require different versions of Node.js.

---

## Node.js

Many of our submodules are Node.js applications. Given the diversity of these submodules, different parts of the project may require different versions of Node.js. To ensure compatibility and proper functioning, it's essential to manage these versions effectively.

### Installing Node.js:

1. **Visit the Official Node.js Website**:
    - To get started, download and install Node.js from the official Node.js website.
    - We recommend using the Long-Term Support (LTS) version for its stability and broad support.
2. **Specific Node.js Versions**:
    - Some submodules may require specific Node.js versions. This information is typically specified in the submodule's documentation or `package.json` file.

### Requirement: Using NVM (Node Version Manager)

Given the need for multiple Node.js versions, **we require all developers to use NVM (Node Version Manager)**. NVM simplifies the process of managing and switching between different Node.js versions.

1. **Why NVM**:
    - NVM allows you to install multiple versions of Node.js and switch between them depending on the submodule you're working on.
    - It ensures that all developers are using the exact version of Node.js required for each part of the project, avoiding version-related inconsistencies and bugs.
2. **Installing NVM**:
    - Follow the installation and usage instructions on the NVM GitHub repository to set up NVM on your machine.
    - After installing NVM, you can install and use different Node.js versions as needed.
3. **Using NVM**:
    - To install a specific version of Node.js:

```
nvm install <version>
```

- To switch to a specific version:

```
nvm use <version>
```

- Ensure that you switch to the correct Node.js version before working on a submodule.

### Conclusion

Using NVM in our project setup is essential for managing the various Node.js versions required by different submodules. This approach ensures a consistent and stable development environment for everyone on the team. If you have any questions or need assistance with setting up NVM, please don't hesitate to reach out to the team for support.

---

This documentation clearly states the requirement of using NVM, outline its benefits, and provide guidance on installation and usage. This approach will help standardize the development environment across our team and ensure smooth handling of multiple Node.js versions.

---

### NPM and Yarn

NPM (Node Package Manager) and Yarn are tools for managing project dependencies. NPM is automatically installed with Node.js, while Yarn is an alternative that offers some performance and usability improvements.

1. **NPM**:
   - Comes bundled with Node.js.
   - Used to install dependencies specified in a `package.json` file.
   - Verify NPM installation by running `npm --version` in your terminal.
2. **Yarn**:
   - Some of our submodules might use Yarn due to its faster installation times and improved lockfile format.
   - To install Yarn, follow the instructions on the official Yarn website.
   - Verify Yarn installation by running `yarn --version` in your terminal.

---

### Managing Dependencies with Yarn

Our project utilizes Yarn for efficient dependency management across all submodules. Yarn ensures that we have consistent dependency installations, improves performance, and offers a reliable lockfile format. Here's

how to manage dependencies in our project:

1. **Installing Dependencies**:
   - After cloning a submodule or pulling the latest changes, it's essential to install its dependencies to ensure that you have all the necessary packages.
   - Navigate to the root directory of the submodule:

     ```
     cd path/to/submodule
     ```

   - Run the following command to install dependencies:

     ```
     yarn install
     ```

   - This command reads the `package.json` file in the submodule, installs the required packages, and generates a `yarn.lock` file that locks the installed versions.
2. **Consistent Dependency Versions**:
   - The `yarn.lock` file ensures that every team member gets the same versions of dependencies, leading to fewer "works on my machine" issues.
   - Commit the `yarn.lock` file to your Git repository to maintain consistency across all environments.
3. **Adding New Dependencies**:
   - To add a new package to a submodule:

     ```
     yarn add package-name
     ```

   - This will update both the `package.json` and `yarn.lock` files. Make sure to commit these changes.
4. **Updating Dependencies**:
   - To update a specific package:

     ```
     yarn upgrade package-name
     ```

   - To update all packages to their latest versions based on the version ranges specified in `package.json`:

```
yarn upgrade
```

5. **Using Yarn in Docker Compose**:
    - Our `docker-compose.yml` file is configured to build Docker images for each submodule. During the build process, `yarn install` is automatically run, ensuring that all required dependencies are included in the Docker image.

---

## Conclusion

**Dependency Management with Yarn:**

- Yarn plays a crucial role in our development process, particularly in managing dependencies across our multi-module project. It ensures consistency, reliability, and efficiency, making it an indispensable tool for our setup.
- For any queries or issues related to dependency management with Yarn, such as package installations, feel free to reach out to the team. We're here to help ensure a smooth development experience.

**Node.js and Package Managers:**

- The correct setup of Node.js, along with the use of Yarn as our package manager, is essential for the development and operation of our Node.js applications. This setup ensures compatibility and functionality across all submodules.
- We recommend using the specified Node.js versions and leveraging NVM for managing multiple versions efficiently. This approach helps in maintaining a consistent development environment and avoids compatibility issues.
- Should you encounter any difficulties or have questions about setting up Node.js, NVM, Yarn, or managing Node.js applications, our team is always available to assist and provide guidance.

---

This comprehensive conclusion brings together the key aspects of using Yarn for dependency management and the importance of the correct Node.js setup. It provides a clear message on the importance of these tools in our project's development workflow and encourages team collaboration and support.