# CS7319 Final Project - Group 7

Here are the list of softwares needed to run the project.

* NodeJS - Minimum Version v18
* MySQL - Community Server 8.4.3 LTS
* RabbitMQ - Minimum Version v4
* Visual Studio Code - Minimum Version 1.89 or another IDE to support NodeJS and ReactJS Development
* Git Tools - Latest Version

Below are the steps for setting up the development environment

* Install NodeJS (for MacOS using bash or for Windows using fnm)
* Install MySQL from MySQL Downloads page
* Install RabbitMQ (for MacOS using Homebrew or for Windows using Chocolatey)
* Install Git Tools
* Install Visual Studio Code
* Clone the Git Repository
* Prepare local database
* Running the application

## Install NodeJS for MacOS using bash:

* Install nvm (Node Version Manager)

curl -o- <https://raw.githubusercontent.com/nvm>sh/nvm/v0.40.0/install.sh | bash

* Download and install Node.js (you may need to restart the terminal)

nvm install 20

* Verifies the right Node.js version is in the environment

node -v # should print `v20.18.0`

* Verifies the right npm version is in the environment

npm -v # should print `10.8.2`

## Install NodeJS for Windows using fnm:

* Installs fnm (Fast Node Manager)

winget install Schniz.fnm

* Configure fnm environment

fnm env --use-on-cd | Out-String | Invoke-Expression

* Download and install Node.js

fnm use --install-if-missing 20

* Verifies the right Node.js version is in the environment

node -v # should print `v20.18.0`

* Verifies the right npm version is in the environment

npm -v # should print `10.8.2`

## Install MySQL

If MySQL is already installed ignore below steps

* Downlod the installation package from <https://dev.mysql.com/downloads/mysql/>

Recommended version: MySQL Community Server 8.4.3 LTS

Note: you do not need t login to download the installation package. Looks for the option to download without login.

* During installation, when prompted for root user password use “mysmu@123” or based on your preference.
* Downalod MySQL Workbench installation package from <https://dev.mysql.com/downloads/workbench/>
* Connect to the local MySQL instance from MySQL Workbench and run the sql script from DB folder

## Install RabbitMQ

If RabbitMQ is already installed, please ignore these steps.

Based on the operating system, download and install RabbitMQ <https://www.rabbitmq.com/docs/platforms>

#### For macOS, the recommended option is to use Homebrew

brew update

brew install rabbitmq

To find out locations for your installation, use:

brew info rabbitmq

Run the RabbitMQ Server Node as a background service

brew services start rabbitmq

Highly recommended: enable all feature flags on the running node

For Apple Silicon Macs

/opt/homebrew/sbin/rabbitmqctl enable\_feature\_flag all

For Intel Macs

/usr/local/opt/rabbitmq/sbin/rabbitmqctl enable\_feature\_flag all

For Stopping the service

brew services stop rabbitmq

#### For Windows, the recommended option is to use Chocolatey

To install RabbitMQ using Chocolatey, run the following command from the command line or from PowerShell:

choco install rabbitmq

More information can be foound here. <https://www.rabbitmq.com/docs/install-windows#chocolatey>

## Install Git Tools

Install Git Tools from below URL. <https://git-scm.com/downloads>

## Install Visual Studio Code

Install latest version of Visual Studio Code from below URL <https://code.visualstudio.com/download>

## Clone the Git Repository

Create new folder

md CS7319-Final-Project-Group07-Smith-Sreepada-Alavala

Change to new folder

cd CS7319-Final-Project-Group07-Smith-Sreepada-Alavala

Clone Git Repository

git clone <https://github.com/CS7319-Architecture-Group7/CS7319-Final->Project-Group07-Smith-Sreepada-Alavala.git

## Before running the application

### Set up the Local Environment:

* Run the DB Scripts from ./DB/PollManagement\_SQL\_Script\_DDL.sql in MySQL Workbench to create PollManagement database schema with all necessary SQL Objects.
* You should have received the .env files for frontend and backend applications through email. If not, please send us an email.
* Copy the FrontEnd “.env” file for frontend application in “./selected/frontend” and “./unselected/frontend/” folders
* Copy the BackEnd “.env” file for backend application in “./selected/backend” and “./unselected/backend/” folders

## Running the “selected” application.

* Open two terminals in MacOS or two Command Prompts in Windows OS.
* Navigate to “./selected/backend/” folder in frist terminal or command prompt and run below commands

npm install  
npm run dev

If there are no errors, backend application must start and listen to port 5001 on localhost

* Navigate to “./selected/frontend/” folder in second terminal or command prompt and run below commands

npm install  
npm start

If there are no errors, frontend application must start and listen to port 3001 on localhost as well as open the default browser

Open additional browsers for testing the poll updates. We recommend to have 3 browsers in below layout to observe the Poll Result updates in a better way.

|  |  |
| --- | --- |
| Browser 1 | Browser 3 |
| Browser 2 |

Use Browser 3 for all the changes like, creating, updating, and delting polls. Once enough polls are created, use Browser 1 and 2 to observe the Poll results when polls are voted from Browser 3.

### Running the “unselected” application

Repeat the same steps above in “./unselected/” folder.

## Difference between the architecture designs for both candidate architecture styles

The Client-Server architecture is characterized by a centralized server that handles requests from multiple clients. This architecture is straightforward to implement and manage, making it ideal for applications with simple, predictable interactions. It allows for centralized control over data and security, ensuring that all clients interact with a single source of truth. However, it can become a bottleneck as the number of clients increases, leading to potential performance issues and scalability challenges. Additionally, real-time updates are harder to implement efficiently, often requiring clients to poll the server frequently.

The Publisher-Subscriber (Pub-Sub) architecture is designed to handle real-time updates and scalability more effectively. In this model, publishers send messages to a message broker, which then distributes these messages to all subscribed clients. This decouples the producers and consumers of data, allowing for more flexible and scalable systems. The Pub-Sub architecture excels in scenarios where real-time data dissemination is crucial, as it enables instant updates to all subscribers without the need for constant polling. However, it introduces additional complexity in managing the message broker and ensuring message delivery, which can require more sophisticated infrastructure and monitoring.

## The rationale for final selection

Proposed Selection: **Client-Server Architecture**

We initially chose the Client-Server architecture for its simplicity and centralized control, which made it easier to implement, maintain, and manage security in the application. Additionally, the centralized nature of this architecture aligned well with the initial requirements for a straightforward, small-scale system with no need for real-time updates

Final Selection: **Publisher-Subscriber Architecture**

As the project evolved, the need for real-time updates in displaying poll results became critical, which the Client-Server architecture struggled to handle efficiently. The Pub-Sub architecture’s event-driven model provided instant updates to subscribers without the need for constant polling, reducing unnecessary traffic. Additionally, the scalability offered by Pub-Sub, with its ability to handle large numbers of users and events through a message broker, proved more suitable for the growing user base. The loosely coupled nature of Pub-Sub allowed for more flexibility and easier management of components as the system complexity increased. Finally, the improved user experience from real-time interaction made Pub-Sub the optimal choice for this application.

## Other useful information about the architectural design decisions

* The choice of architecture was based on the need for scalability and real-time updates.
* Client-Server architecture was initially chosen for its simplicity and ease of implementation.
* The centralized control in Client-Server architecture made it easier to manage security.
* As the project requirements evolved, the need for real-time updates became critical.
* Publisher-Subscriber architecture was selected for its event-driven model.
* Pub-Sub architecture allows for instant updates to subscribers without constant polling.
* The scalability of Pub-Sub architecture can handle a large number of users and events.
* The decoupling of producers and consumers in Pub-Sub architecture provides flexibility.
* Pub-Sub architecture reduces unnecessary traffic by avoiding constant polling.
* The message broker in Pub-Sub architecture ensures efficient message distribution.
* The loosely coupled nature of Pub-Sub architecture simplifies component management.
* Real-time interaction in Pub-Sub architecture improves user experience.
* The initial simplicity of Client-Server architecture was suitable for small-scale systems.
* As the user base grew, the need for a more scalable solution becames a necessity.
* Pub-Sub architecture supports the growing complexity of the system.
* The event-driven model of Pub-Sub architecture aligns with the need for real-time updates.
* The centralized server in Client-Server architecture can become a bottleneck with a growing user base just for checking poll updates.
* Pub-Sub architecture provides better performance under high load conditions.
* The flexibility of Pub-Sub architecture allows for easier integration of new features.
* The final selection of Pub-Sub architecture was driven by the need for improved scalability and real-time updates.