

# Lab5 Report

---

Yiwen Xu (48377645)

## Computer Environment

---

Operating system: macOS Ventura Version 13.0.1

Processor: 2.3 GHz Quad-Core Intel Core i7

Memory: 16 GB 3733 MHz LPDDR4X

IDE: Visual Studio Code

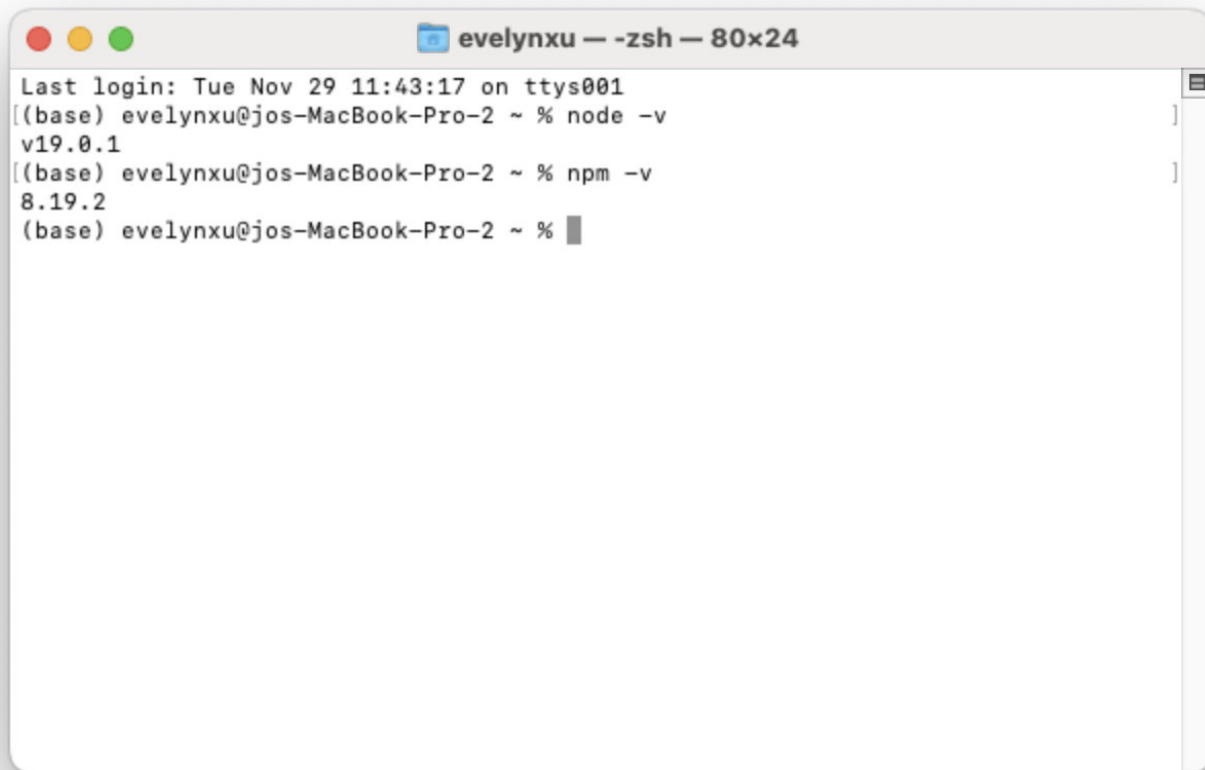
## Setup Electron

---

To begin developing an Electron app, we need to install the Node.js runtime and its bundled npm package manager onto our system. To check that Node.js was installed correctly, we can use the following commands.

```
node -v  
npm -v
```

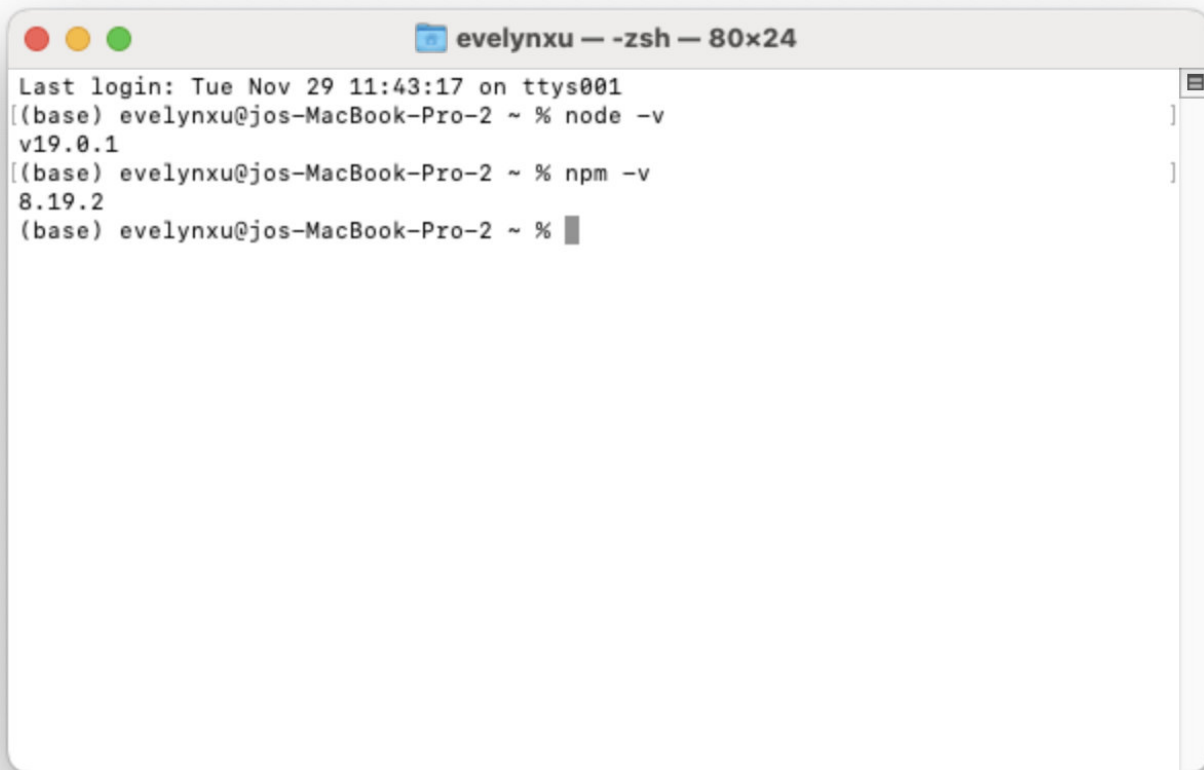
This is the version I used:



```
evelynxu — -zsh — 80x24
Last login: Tue Nov 29 11:43:17 on ttys001
[(base) evelynxu@jos-MacBook-Pro-2 ~ % node -v
v19.0.1
[(base) evelynxu@jos-MacBook-Pro-2 ~ % npm -v
8.19.2
(base) evelynxu@jos-MacBook-Pro-2 ~ %
```

Note that users do not need to install Node.js themselves as a prerequisite to running this Electron app.

In this project, I used Lab4 (Networking) as code base. Therefore, I enter the code base folder and initializing an npm package within it with `npm init`.


A terminal window titled "evelynxu — -zsh — 80x24" with standard macOS window controls (red, yellow, green buttons). The terminal shows the following text: "Last login: Tue Nov 29 11:43:17 on ttys001", "[(base) evelynxu@jos-MacBook-Pro-2 ~ % node -v]", "v19.0.1", "[(base) evelynxu@jos-MacBook-Pro-2 ~ % npm -v]", "8.19.2", and "[(base) evelynxu@jos-MacBook-Pro-2 ~ % ]" with a cursor at the end.

```
Last login: Tue Nov 29 11:43:17 on ttys001
[(base) evelynxu@jos-MacBook-Pro-2 ~ % node -v
v19.0.1
[(base) evelynxu@jos-MacBook-Pro-2 ~ % npm -v
8.19.2
[(base) evelynxu@jos-MacBook-Pro-2 ~ % ]
```

Then install Electron into my app's devDependencies with `npm install electron --save-dev`. Also, for packaging this project, I'll packaging this app with [Electron Forge](#). I install Electron Forge's CLI in my project's devDependencies and import my existing project with a handy conversion script.

```
npm install --save-dev @electron-forge/cli
npx electron-forge import
```

Now we get a `package.json` file, a `forge.config.js` file and a `package-lock.json` file:

A screenshot of a code editor with a dark theme. The editor shows the 'package.json' file for a project named 'chat-websocket-electron'. The file contains a JSON object with the following properties: 'name' (chat-websocket-electron), 'version' (0.0.1), 'private' (true), 'scripts' (start: electron-forge start, package: electron-forge package, make: electron-forge make), 'dependencies' (connect: ^3.7.0, electron-squirrel-startup: ^1.0.0, express: ^3.21.2, ws: ^8.11.0), 'description' (chat-websocket =====), 'main' (main.js), 'devDependencies' (@electron-forge/cli: ^6.0.3, @electron-forge/maker-deb: ^6.0.3, @electron-forge/maker-rpm: ^6.0.3, @electron-forge/maker-squirrel: ^6.0.3, @electron-forge/maker-zip: ^6.0.3, electron: ^21.2.3), 'author' (Yiwen Xu), and 'license' (MIT). The editor has tabs for 'package.json', 'package-lock.json', and 'forge.config.js'. A '4x' magnification icon is visible on the right side of the editor. The breadcrumb path at the top reads 'advanceProgramming > chat-websocket-electron > {} package.json > ...'.

```
1  {
2    "name": "chat-websocket-electron",
3    "version": "0.0.1",
4    "private": true,
5    "scripts": {
6      "start": "electron-forge start",
7      "package": "electron-forge package",
8      "make": "electron-forge make"
9    },
10   "dependencies": {
11     "connect": "^3.7.0",
12     "electron-squirrel-startup": "^1.0.0",
13     "express": "^3.21.2",
14     "ws": "^8.11.0"
15   },
16   "description": "chat-websocket =====",
17   "main": "main.js",
18   "devDependencies": {
19     "@electron-forge/cli": "^6.0.3",
20     "@electron-forge/maker-deb": "^6.0.3",
21     "@electron-forge/maker-rpm": "^6.0.3",
22     "@electron-forge/maker-squirrel": "^6.0.3",
23     "@electron-forge/maker-zip": "^6.0.3",
24     "electron": "^21.2.3"
25   },
26   "author": "Yiwen Xu",
27   "license": "MIT"
28 }
```

## Code Base

### Building Electron App

The `main` script I defined in `package.json` is the entry point of any Electron application. So I created a `main.js` file in root folder of my project. Now we can load my web page `chat.html` into an Electron `BrowserWindow` with the following simple code.

```
const { app, BrowserWindow } = require('electron')

const createWindow = () => {
  const win = new BrowserWindow({
    width: 800,
    height: 600,
```

```

}))

win.loadFile('chat.html')
};

app.whenReady().then(() => {
  createWindow();
});

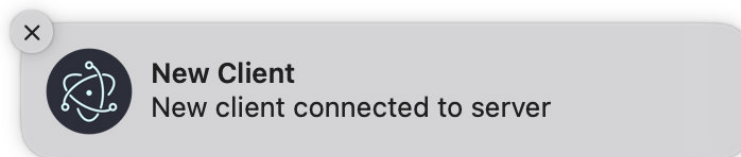
app.on('activate', () => {
  // Can open most 3 clients
  if (BrowserWindow.getAllWindows().length < 3) {
    createWindow();
  }
});

app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') {
    app.quit();
  }
});

```

## Adding New Feature

I added a notification function to my project. It will notified user when a new client is setup.



To implement this for the renderer process, Electron conveniently allows us to send notifications with the HTML5 Notification API, using the currently running operating system's native notification APIs to display it.

```

connection.onopen = function () {
  const NOTIFICATION_TITLE = 'New Client'
  const NOTIFICATION_BODY = 'New client connected to server'
  new Notification(NOTIFICATION_TITLE, { body: NOTIFICATION_BODY })

  connection.send('Start Working');
  work();
};

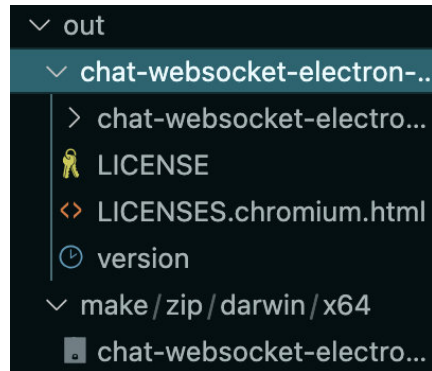
```

## Packaging Application

To create a distributable, use this project's new `make` script (already applied in [Setup Electron](#) part), which runs the `electron-forge make` command.

```
npm run make
```

After the script runs, I get an `out` folder containing both the distributable and a folder containing the packaged application code.



The distributable in the `out/make` folder should be ready to launch.

## Demonstration

---

Before we can start running this project, we need to set up a local server to perform its functions well.

### Server

The server is implemented with JavaScript using node server. It also using Express.js, which is a web application framework for Node.js, enables you to build server applications in Node.js. And the logs of message and progress will be printed during runtime.

```
supervisor app.js  
# or  
node app.js
```



```
chat-websocket-master — node ◀ node /usr/local/bin/supervisor app.js — 8...
[(base) evelynxu@jos-MacBook-Pro-2 chat-websocket-master % supervisor app.js

Running node-supervisor with
  program 'app.js'
  --watch '.'
  --extensions 'node,js'
  --exec 'node'

Starting child process with 'node app.js'
Watching directory '/Users/evelynxu/Documents/workplace_vscode/AdvanceProgrammin
g/chat-websocket-master' for changes.
Press rs for restarting the process.
Express server listening on port 3000
█
```

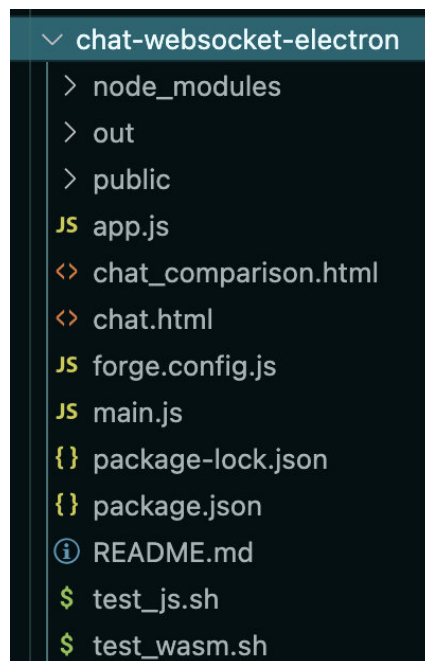
```
chat-websocket-master — node ◀ node /usr/local/bin/supervisor app_comparis...
ReceivedData: 77
3 say: Done
ReceivedData: 78
3 say: Done
ReceivedData: 79
2 say: Done
ReceivedData: 80
3 say: Done
ReceivedData: 81
1 say: Done
ReceivedData: 82
3 say: Done
ReceivedData: 83
2 say: Done
ReceivedData: 84
3 say: Done
ReceivedData: 85
2 say: Done
ReceivedData: 86
1 say: Done
ReceivedData: 87
3 say: Done
ReceivedData: 88
2 say: Done
ReceivedData: 89
3 say: Done
ReceivedData: 90
2 say: Done
ReceivedData: 91
1 say: Done
ReceivedData: 92
3 say: Done
ReceivedData: 93
2 say: Done
ReceivedData: 94
```



```
3 say: Done
ReceivedData: 95
2 say: Done
ReceivedData: 96
1 say: Done
ReceivedData: 97
3 say: Done
ReceivedData: 98
2 say: Done
ReceivedData: 99
3 say: Done
ReceivedData: 100
2 say: Done
Execution Time: 2.8558177943229675
All Execution Time:
3.469878340244293
2.8736786880493166
2.8558177943229675
```

## Client

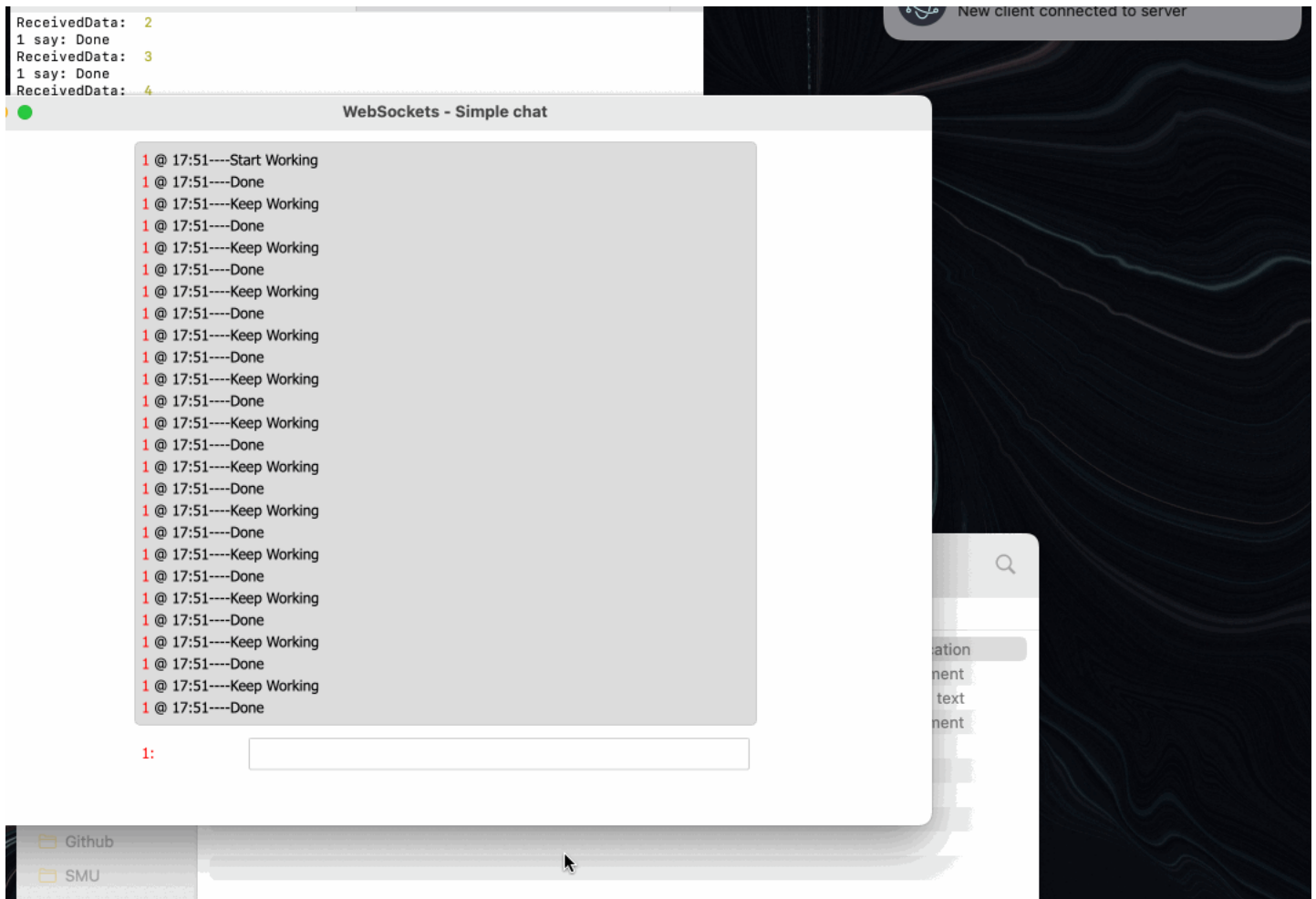
The first to run the project, you need to run `npm install`, which is used to download the dependencies and create the `node_modules` file. After that, all project files should look like this.



To start a new client, we can use this command:

```
npm run start
```

Also, we can open it with the packaged Electron applications by simply double click it.



However, it may not work on a different device because I haven't signed my code as I have not gotten the certification.

## Document API

### app.js

This is a module that contains the implementation to communicate as a server using the WebSocket protocol and to coordinate the work being processed by the client.

### Attributes

Name	Type
app	Express
server	http.Server
wss	WebSocketServer
ws	WebSocket.WebSocket
clients	Array[WebSocket.WebSocket]
colors	Array[String]
userID	Number
receivedData	Number
startTime	Array[Number]
endTime	Number
execTimes	Array[Number]

## Methods

Name	Description
wss.on('connection', function(ws))	Used to handle the logic after the client connects to the server.
ws.on('message', function(msg))	Logical processing after receiving the message.
ws.on('close', function())	Used to handle the logic after the client close the connection with the server.
app.configure()	Configure Express.
app.get('/', (req, res) =>{})	Set up such a single route with Express.

## browser.js

This is a module that contains the implementation to communicate as a client using the WebSocket protocol and processes the work.

## Attributes

## Attributes

Name	Type
myColor	String
myName	String
connection	WebSocket
data	Number

## Methods

Name	Description
WebSocket.onopen	The socket's open event listener, used to handle the logic after the socket is successfully opened.
WebSocket.onmessage	The socket's message event listener, used to handle logic after receiving a message
WebSocket.onclose	The socket's close event listener, used to handle the logic after the socket is closed.
input.keydown	Event listener for the Enter key. Sends a message to the server when the Enter key is pressed.
addMessage	Compose the received content into div blocks for display on the HTML page.
work	Client's work process.

## Main.cpp

This is a module that contains the implementation to communicate as a client using the WebSocket protocol and processes the work, which is a comparison version of `browser.js`.

## Attributes

Name	Type
myName	string
socket	EMSCRIPTEN_WEBSOCKET_T
data	double

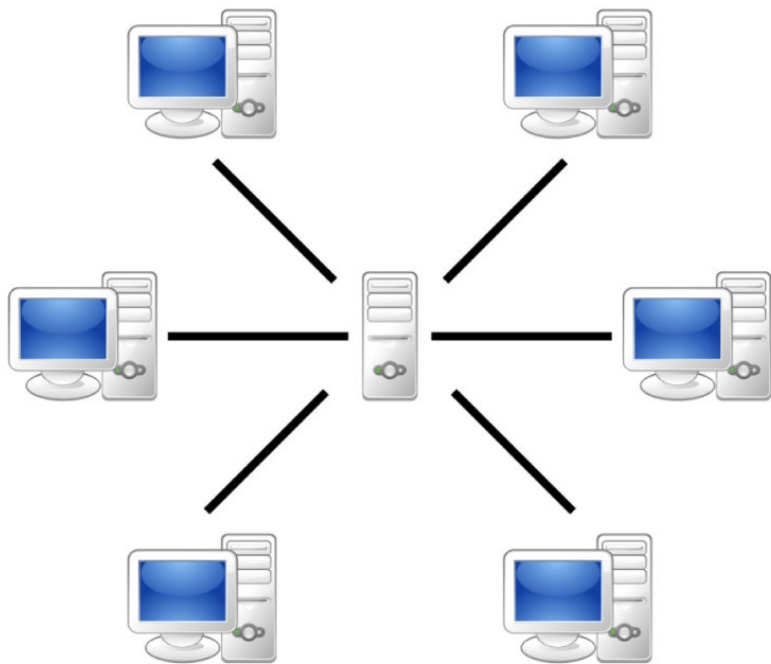
## Methods

METHODS

Name	Input	Output	Description
WebSocketOpen	eventTypr:int, e:EmscriptenWebSocketOpenEvent*, userData:void*	EM_BOOL	
WebSocketClose	eventTypr:int, e:EmscriptenWebSocketCloseEvent*, userData:void*	EM_BOOL	
WebSocketMessage	eventTypr:int, e:EmscriptenWebSocketMessageEvent*, userData:void*	EM_BOOL	
work	Client's work process.	void	

Networking Architecture

To meet the requirements, I modified it to implement a part of producer-consumer model which just like Lab3. In this case, each client is represents a machine and sever is represents a factory which can coordinate the work that being processed by machines. When the client connects to the server, it will start working. After each round of work, it will send messages and data to the server. The server will check if the target number of data has been reached, or let the client continue working.



Server-based

*The networking architecture*

As the sequence diagram shows below, we can see the flow logic of this project and how a server actually coordinates the work process.

