COMPSCI 326 Group 1 (Online)

Mini Terminal

(https://github.com/VicayoMua/326_Mini_Terminal)

Milestone 6

Team Member: Vicayo Zhang, Aryan Ghosh, and Stella Dey

4/25/2025

Project Name: Mini Terminal

Problem/Why This Project?

- Large universities, like UMass, usually provide ssh server environments, like EdLab, for students to get familiar with Linux-styled commands. However, most students only use the most basic features on EdLab, so the power assumption and the continuous maintenance cost are not worth it.
- Students, who are not enrolled in a university and have no previous experience in installing and using Linux, usually find it hard to install a Linux distribution on their PCs.

Solution:

- A terminal simulator, which can be accessed on web browsers and run on local devices (phones, tablets, and PCs), can perfectly solve this problem.

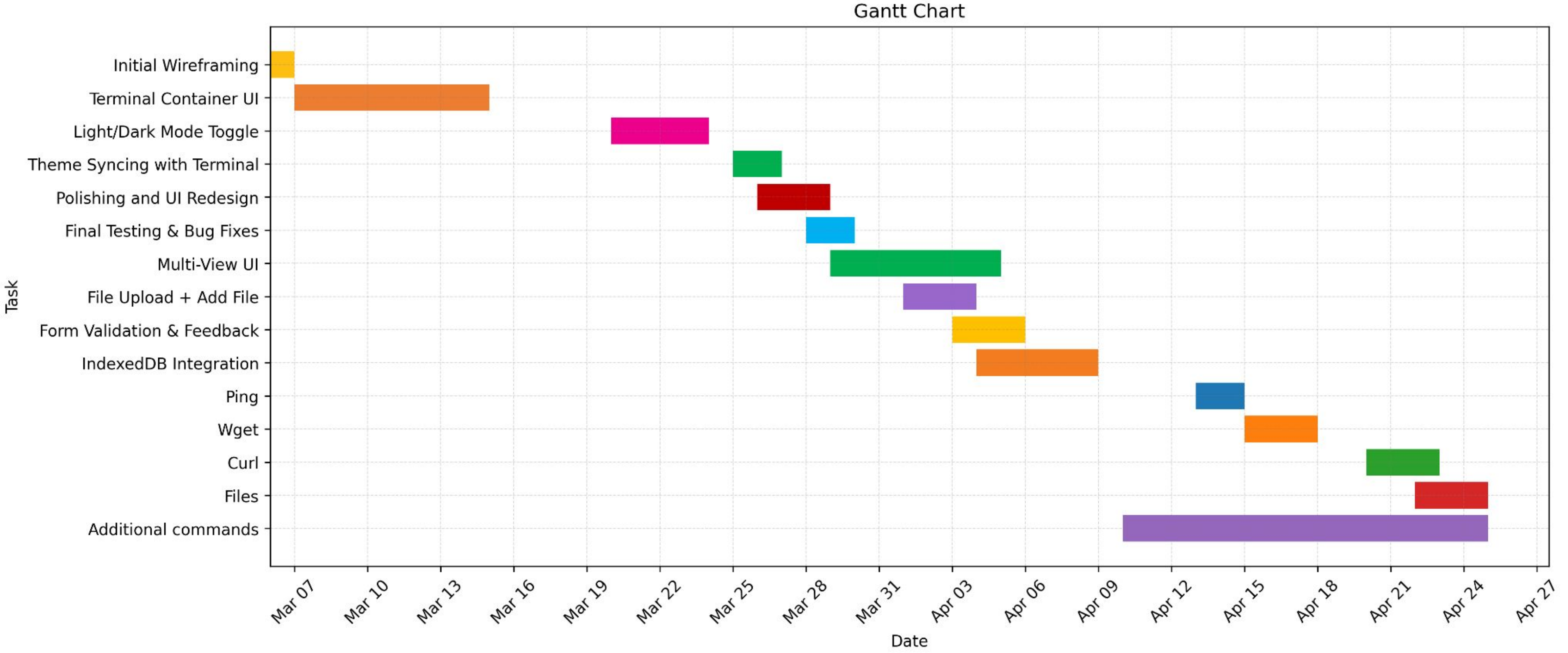
Key Features:

- The terminal simulator will support the most usual Linux commands, like ls, pwd, cd, mkdir, rename, touch, cp, edit. It has file managing abilities.
- Additionally, we can set up additional buttons outside the terminal window to support log-saving, file-system-importing, and file-system-exporting. So, people can easily save and resume their work efficiently.
- Finally, if time allows, the terminal will have supports for Web Assembly.

Team Members

- **Vicayo Zhang – Project Manager & Core Service Developer**
  - **First Issue To Work ON: Basic terminal input and output logics**
  - **Second Issue To Work ON: Log recording logics**
  - **Third Issue To Work ON: file system simulation logics**
  - **Fourth Issue To Work ON: some demo on-terminal applications/commands (hello, help, man, and echo)**
- **Aryan Ghosh – Software Developer & Admin Monitoring**
  - **First Issue To Work ON: Ping command setup**
  - **Second Issue To Work ON: User Session data**
  - **Third Issue To Work ON: Project Documentation**
- **Stella Dey – Debugging & Testing Coordinator**
  - **First Issue To Work ON: Customizable Theme - (Light/Dark Mode) #17**
  - **Second Issue To Work ON: File System Structure Data #18**
  - **Third Issue To Work ON: Curl and files command setup**

Historical Development Timeline

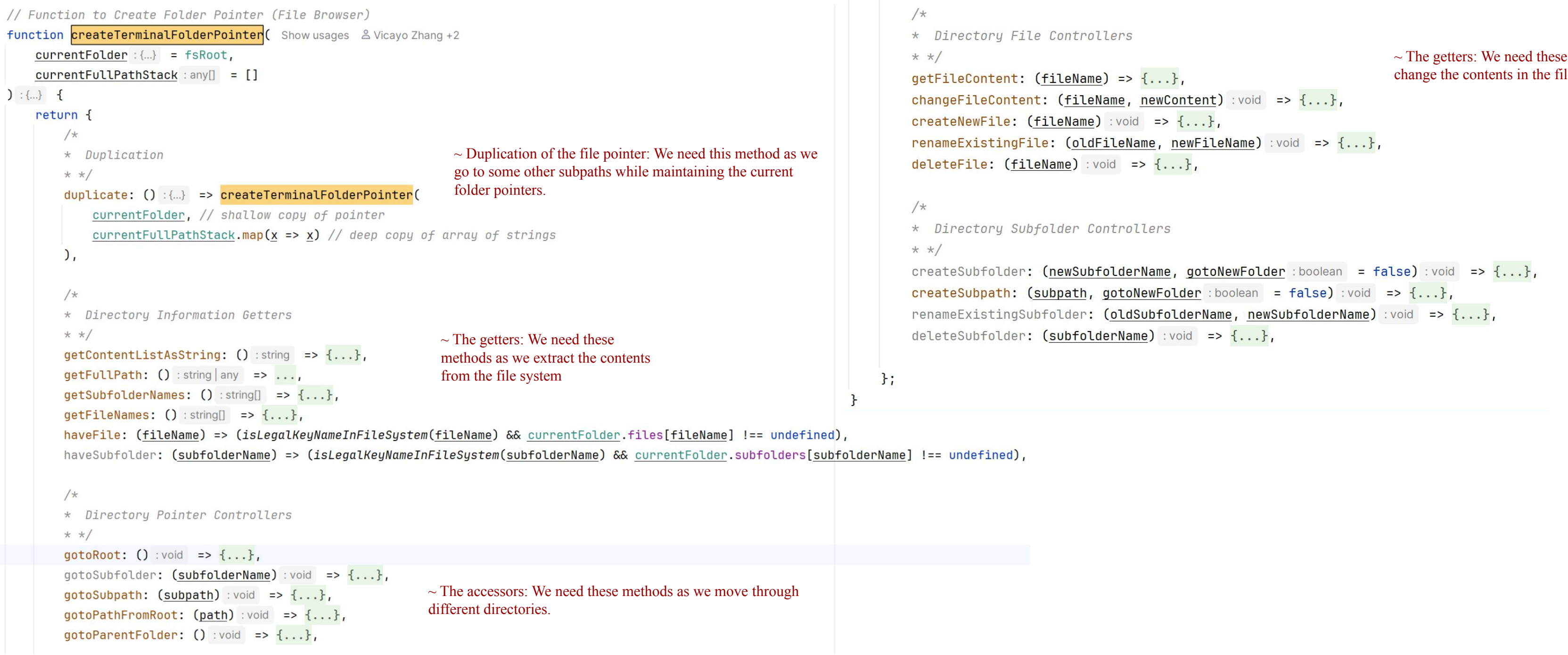


Vicayo Zhang - Assigned Work Summary

In the JS file:

- Implement and update the file system managing APIs (Feature A. Screenshot 1)
- Implement the commands:
  - help (Feature B. Screenshot 2)
  - man (Feature C. Screenshot 3)
  - echo (Feature D. Screenshot 4)
  - ls (Feature E. Screenshot 5)
  - mkdir (Feature F. Screenshot 6)
  - pwd (Feature G. Screenshot 7)
  - wget (Feature H. Screenshot 8)
- Solve all the branch conflicts in the pull requests (Screenshot 9)

Vicayo Zhang - Screenshot 1 (Code & UI Explanation)



```
// Function to Create Folder Pointer (File Browser)
function createTerminalFolderPointer(  Show usages   Vicayo Zhang +2
    currentFolder : {...}  = fsRoot,
    currentFullPathStack : any[]  = []
) : {...}  {
    return {
        /*
        *  Duplication
        * */
        duplicate: () : {...}  => createTerminalFolderPointer(
            currentFolder, // shallow copy of pointer
            currentFullPathStack.map(x => x) // deep copy of array of strings
        ),
```

~ Duplication of the file pointer: We need this method as we go to some other subpaths while maintaining the current folder pointers.

```
        /*
        *  Directory Information Getters
        * */
        getContentListAsString: () : string  => {...},
        getFullPath: () : string | any  => ...,
        getSubfolderNames: () : string[]  => {...},
        getFileNames: () : string[]  => {...},
        haveFile: (fileName) => (isLegalKeyNameInFileSystem(fileName) && currentFolder.files[fileName] !== undefined),
        haveSubfolder: (subfolderName) => (isLegalKeyNameInFileSystem(subfolderName) && currentFolder.subfolders[subfolderName] !== undefined),
```

~ The getters: We need these methods as we extract the contents from the file system

```
        /*
        *  Directory Pointer Controllers
        * */
        gotoRoot: () : void  => {...},
        gotoSubfolder: (subfolderName) : void  => {...},
        gotoSubpath: (subpath) : void  => {...},
        gotoPathFromRoot: (path) : void  => {...},
        gotoParentFolder: () : void  => {...},
```

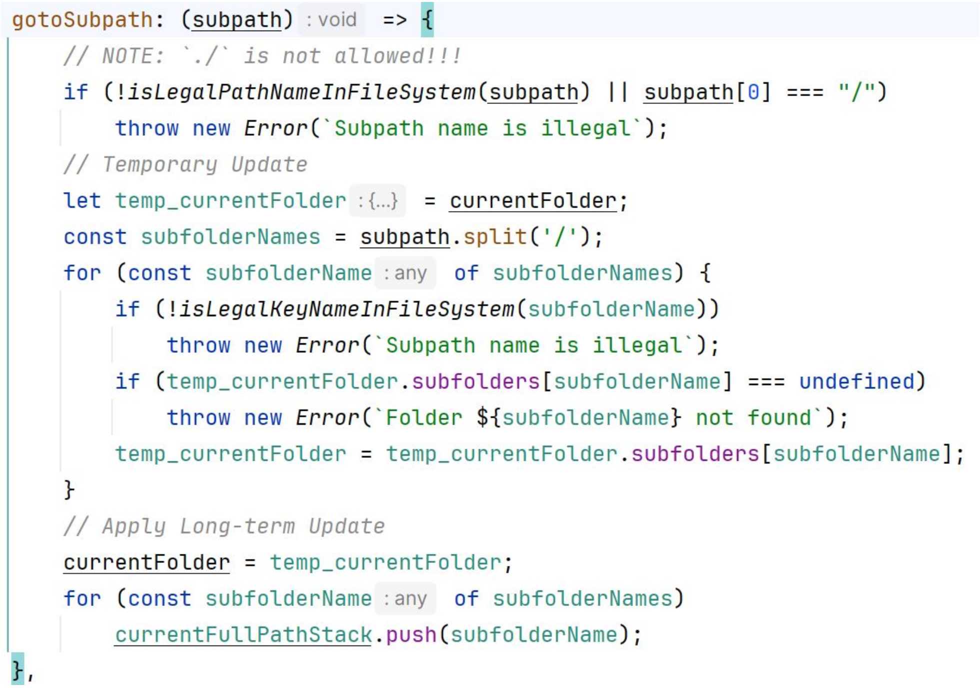
~ The accessors: We need these methods as we move through different directories.

```
        /*
        *  Directory File Controllers
        * */
        getFileContent: (fileName) => {...},
        changeFileContent: (fileName, newContent) : void  => {...},
        createNewFile: (fileName) : void  => {...},
        renameExistingFile: (oldFileName, newFileName) : void  => {...},
        deleteFile: (fileName) : void  => {...},
```

~ The getters: We need these methods as we change the contents in the file system.

```
        /*
        *  Directory Subfolder Controllers
        * */
        createSubfolder: (newSubfolderName, gotoNewFolder : boolean  = false) : void  => {...},
        createSubpath: (subpath, gotoNewFolder : boolean  = false) : void  => {...},
        renameExistingSubfolder: (oldSubfolderName, newSubfolderName) : void  => {...},
        deleteSubfolder: (subfolderName) : void  => {...},

    };
}
```

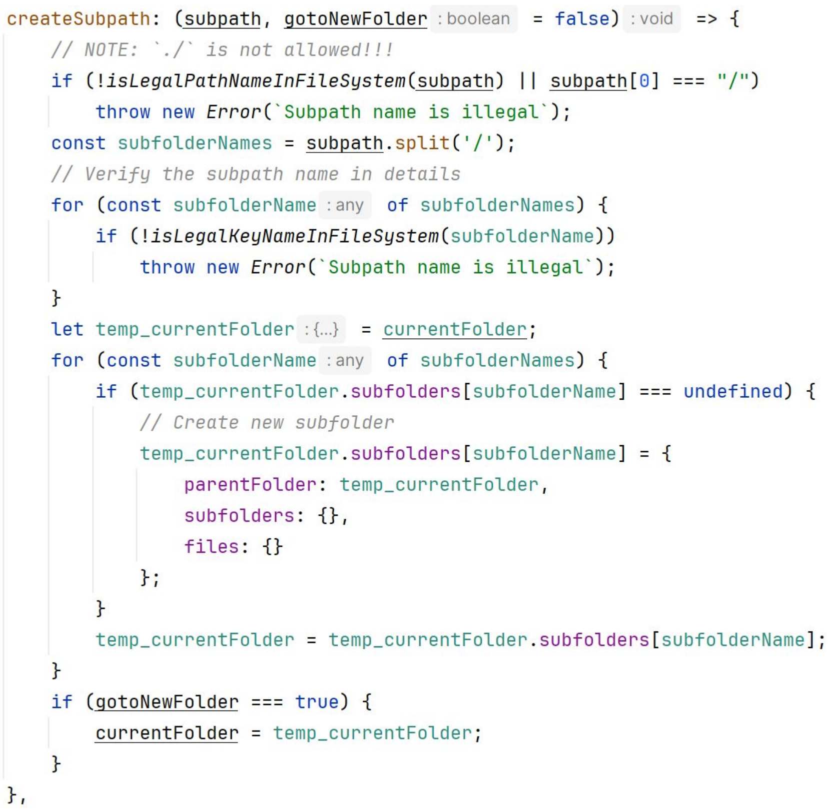
Vicayo Zhang - Screenshot 1 (Code & UI Explanation)



```
gotoSubpath: (subpath) : void  => {
    // NOTE: `./` is not allowed!!!
    if (!isLegalPathNameInFileSystem(subpath) || subpath[0] === "/")
        throw new Error(`Subpath name is illegal`);
    // Temporary Update
    let temp_currentFolder : {...}  = currentFolder;
    const subfolderNames = subpath.split('/');
    for (const subfolderName : any  of subfolderNames) {
        if (!isLegalKeyNameInFileSystem(subfolderName))
            throw new Error(`Subpath name is illegal`);
        if (temp_currentFolder.subfolders[subfolderName] === undefined)
            throw new Error(`Folder ${subfolderName} not found`);
        temp_currentFolder = temp_currentFolder.subfolders[subfolderName];
    }
    // Apply Long-term Update
    currentFolder = temp_currentFolder;
    for (const subfolderName : any  of subfolderNames)
        currentFullPathStack.push(subfolderName);
},
```

To fast goto some subpath of a folder (without much overhead from calling gotoSubfolder multiple times), we need this method, which firstly check whether the destination folder exists or not, and if it exists, then the folder pointer and path stack are changed.

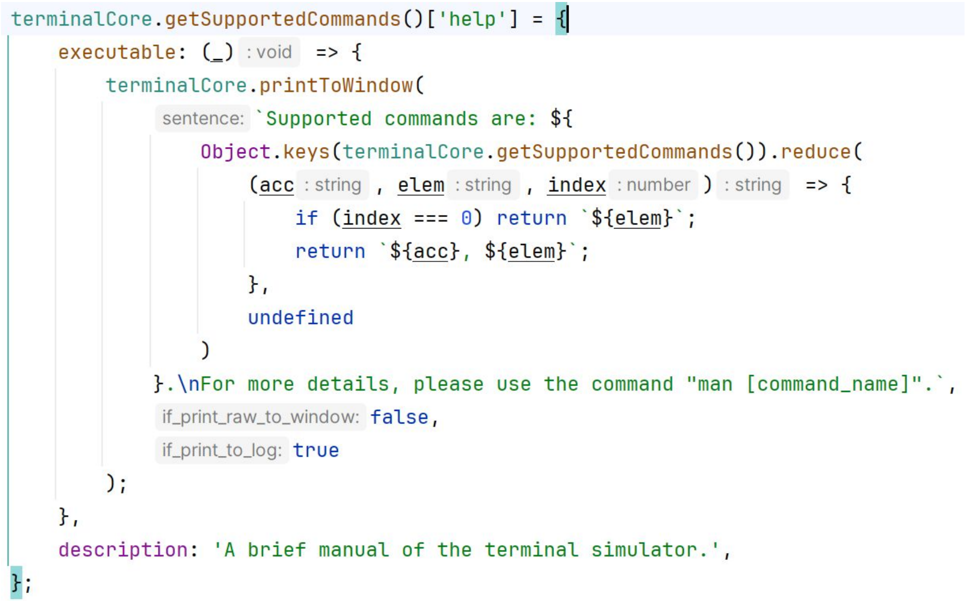
Vicayo Zhang - Screenshot 1 (Code & UI Explanation)



```
createSubpath: (subpath, gotoNewFolder : boolean = false) : void => {
    // NOTE: `./` is not allowed!!!
    if (!isLegalPathNameInFileSystem(subpath) || subpath[0] === "/")
        throw new Error(`Subpath name is illegal`);
    const subfolderNames = subpath.split('/');
    // Verify the subpath name in details
    for (const subfolderName : any of subfolderNames) {
        if (!isLegalKeyNameInFileSystem(subfolderName))
            throw new Error(`Subpath name is illegal`);
    }
    let temp_currentFolder : {...} = currentFolder;
    for (const subfolderName : any of subfolderNames) {
        if (temp_currentFolder.subfolders[subfolderName] === undefined) {
            // Create new subfolder
            temp_currentFolder.subfolders[subfolderName] = {
                parentFolder: temp_currentFolder,
                subfolders: {},
                files: {}
            };
        }
        temp_currentFolder = temp_currentFolder.subfolders[subfolderName];
    }
    if (gotoNewFolder === true) {
        currentFolder = temp_currentFolder;
    }
},
```

To fast create some subpath of a folder (without much overhead from calling createSubfolder multiple times), we need this method, which firstly check whether the destination folder can be created or not, and if it can, then it's created immediately. There is a option to move the current folder pointer to the newly created folder immediately after the creation (for some optimization).

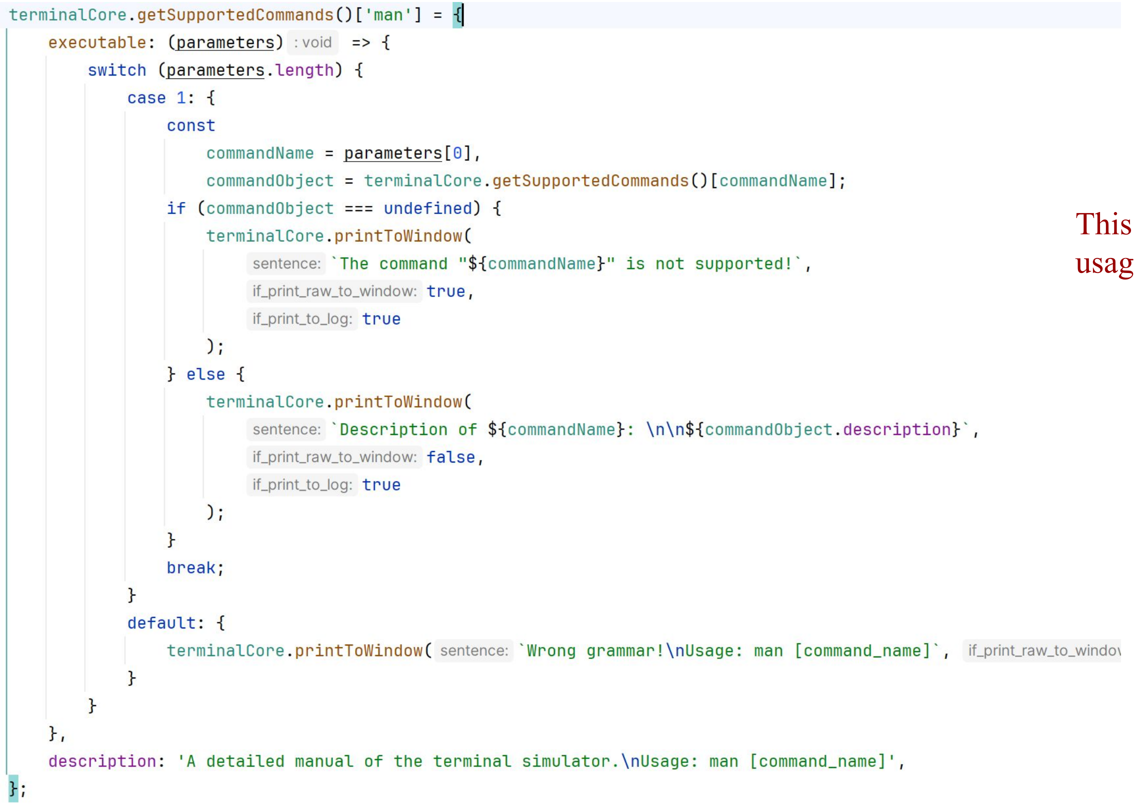
Vicayo Zhang - Screenshot 2 (Code & UI Explanation)



```
terminalCore.getSupportedCommands()['help'] = {
    executable: (_) : void  => {
        terminalCore.printToWindow(
            sentence: `Supported commands are: ${
                Object.keys(terminalCore.getSupportedCommands()).reduce(
                    (acc : string , elem : string , index : number ) : string  => {
                        if (index === 0) return `${elem}`;
                        return `${acc}, ${elem}`;
                    },
                    undefined
                )
            }.\nFor more details, please use the command "man [command_name]".`,
            if_print_raw_to_window: false,
            if_print_to_log: true
        );
    },
    description: 'A brief manual of the terminal simulator.',
};
```

This is the help command, which prints some basic usage hints for the user.

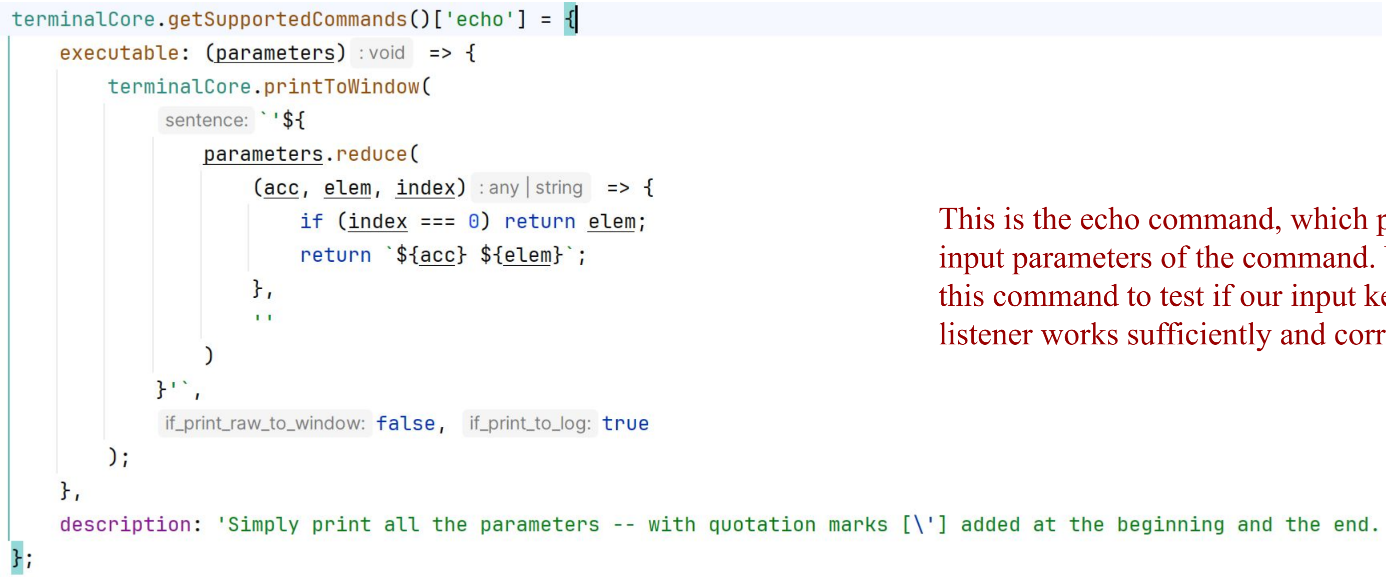
Vicayo Zhang - Screenshot 3 (Code & UI Explanation)



```
terminalCore.getSupportedCommands()['man'] = {
    executable: (parameters) : void  => {
        switch (parameters.length) {
            case 1: {
                const
                    commandName = parameters[0],
                    commandObject = terminalCore.getSupportedCommands()[commandName];
                if (commandObject === undefined) {
                    terminalCore.printToWindow(
                        sentence: `The command "${commandName}" is not supported!`,
                        if_print_raw_to_window: true,
                        if_print_to_log: true
                    );
                } else {
                    terminalCore.printToWindow(
                        sentence: `Description of ${commandName}: \n\n${commandObject.description}`,
                        if_print_raw_to_window: false,
                        if_print_to_log: true
                    );
                }
                break;
            }
            default: {
                terminalCore.printToWindow( sentence: `Wrong grammar!\nUsage: man [command_name]`,  if_print_raw_to_windov
            }
        }
    },
    description: 'A detailed manual of the terminal simulator.\nUsage: man [command_name]',
};
```

This is the man command, which prints details usage hints for the user.

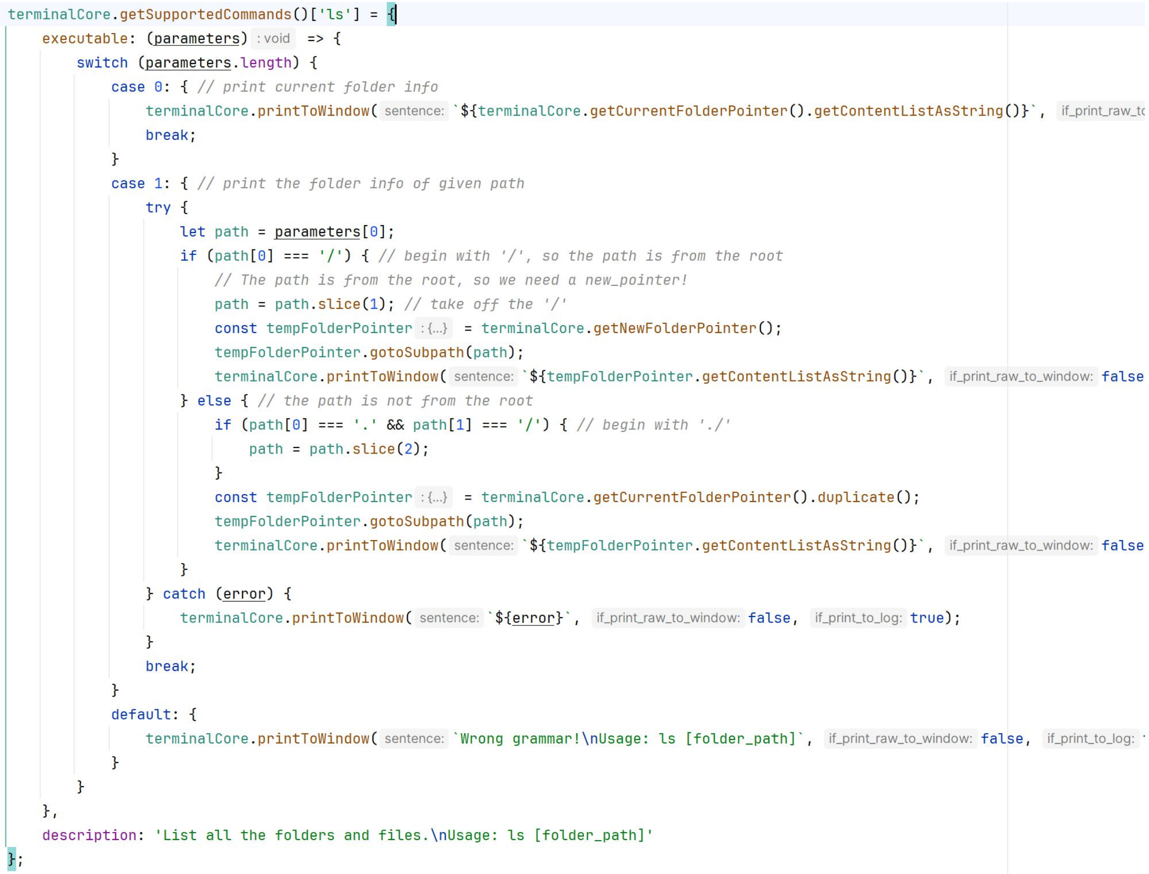
Vicayo Zhang - Screenshot 4 (Code & UI Explanation)



```
terminalCore.getSupportedCommands()['echo'] = {
    executable: (parameters) : void  => {
        terminalCore.printToWindow(
            sentence: `'${
                parameters.reduce(
                    (acc, elem, index) : any | string  => {
                        if (index === 0) return elem;
                        return `${acc} ${elem}`;
                    },
                    ''
                )
            }'`,
            if_print_raw_to_window: false,  if_print_to_log: true
        );
    },
    description: 'Simply print all the parameters -- with quotation marks [\'] added at the beginning and the end.'
};
```

This is the echo command, which prints the input parameters of the command. We can use this command to test if our input keyboard listener works sufficiently and correctly.

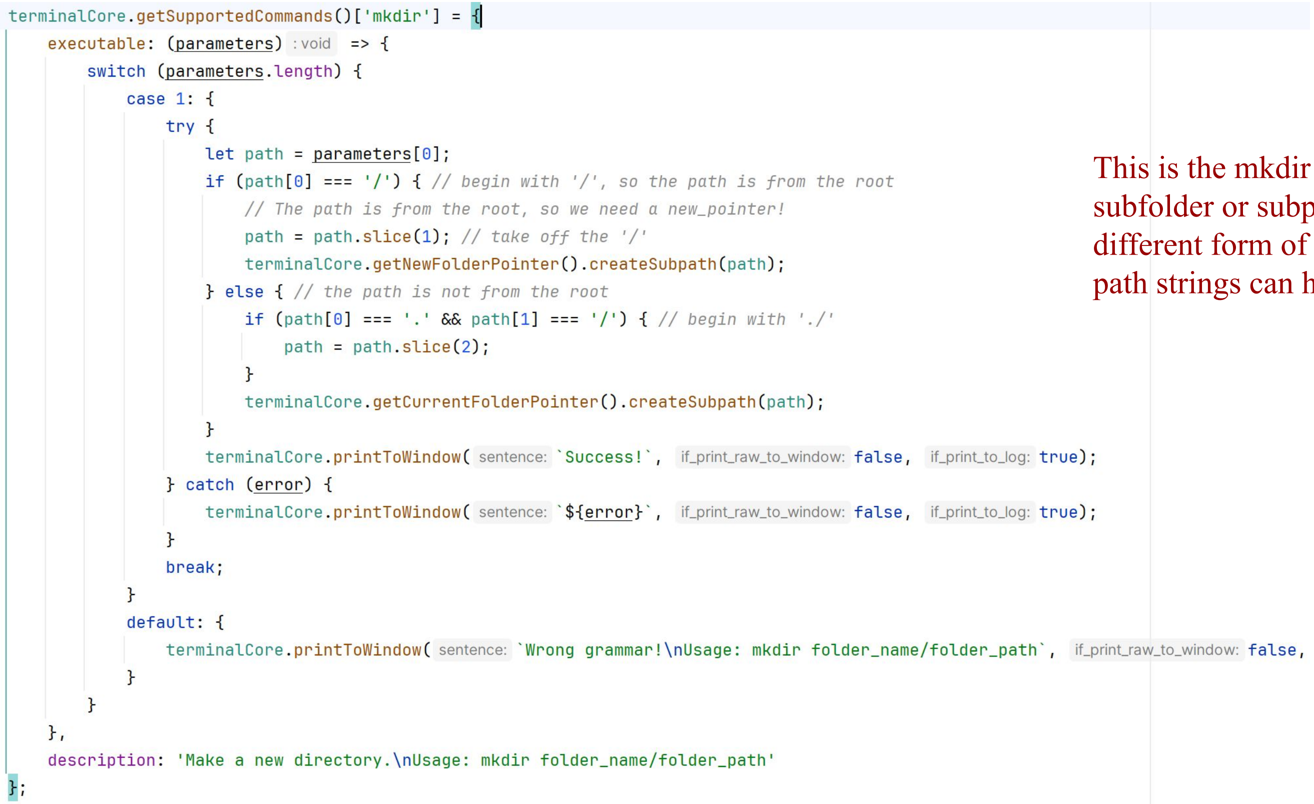
Vicayo Zhang - Screenshot 5 (Code & UI Explanation)



```
terminalCore.getSupportedCommands()['ls'] = {
    executable: (parameters) :void  => {
        switch (parameters.length) {
            case 0: { // print current folder info
                terminalCore.printToWindow( sentence: `${terminalCore.getCurrentFolderPointer().getContentListAsString()}`,  if_print_raw_tc
                break;
            }
            case 1: { // print the folder info of given path
                try {
                    let path = parameters[0];
                    if (path[0] === '/') { // begin with '/', so the path is from the root
                        // The path is from the root, so we need a new_pointer!
                        path = path.slice(1); // take off the '/'
                        const tempFolderPointer :{...}  = terminalCore.getNewFolderPointer();
                        tempFolderPointer.gotoSubpath(path);
                        terminalCore.printToWindow( sentence: `${tempFolderPointer.getContentListAsString()}`,  if_print_raw_to_window: false
                    } else { // the path is not from the root
                        if (path[0] === '.' && path[1] === '/') { // begin with './'
                            path = path.slice(2);
                        }
                        const tempFolderPointer :{...}  = terminalCore.getCurrentFolderPointer().duplicate();
                        tempFolderPointer.gotoSubpath(path);
                        terminalCore.printToWindow( sentence: `${tempFolderPointer.getContentListAsString()}`,  if_print_raw_to_window: false
                    }
                } catch (error) {
                    terminalCore.printToWindow( sentence: `${error}`,  if_print_raw_to_window: false,  if_print_to_log: true);
                }
                break;
            }
            default: {
                terminalCore.printToWindow( sentence: `Wrong grammar!\nUsage: ls [folder_path]`,  if_print_raw_to_window: false,  if_print_to_log:
            }
        }
    },
    description: 'List all the folders and files.\nUsage: ls [folder_path]'
};
```

This is the ls command, which prints all the names of the folders and files in the current/given directory. Here, dealing with different form of path string is critical: some path strings can have leading “/” or “./”.

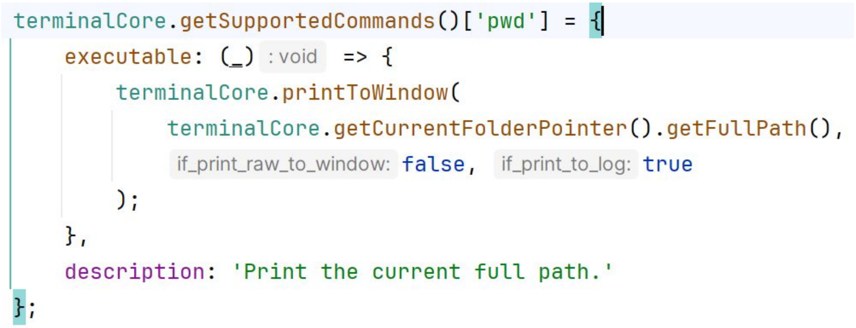
Vicayo Zhang - Screenshot 6 (Code & UI Explanation)



```
terminalCore.getSupportedCommands()['mkdir'] = {
    executable: (parameters) : void  => {
        switch (parameters.length) {
            case 1: {
                try {
                    let path = parameters[0];
                    if (path[0] === '/') { // begin with '/', so the path is from the root
                        // The path is from the root, so we need a new_pointer!
                        path = path.slice(1); // take off the '/'
                        terminalCore.getNewFolderPointer().createSubpath(path);
                    } else { // the path is not from the root
                        if (path[0] === '.' && path[1] === '/') { // begin with './'
                            path = path.slice(2);
                        }
                        terminalCore.getCurrentFolderPointer().createSubpath(path);
                    }
                    terminalCore.printToWindow( sentence: `Success!`,  if_print_raw_to_window: false,  if_print_to_log: true);
                } catch (error) {
                    terminalCore.printToWindow( sentence: `${error}`,  if_print_raw_to_window: false,  if_print_to_log: true);
                }
                break;
            }
            default: {
                terminalCore.printToWindow( sentence: `Wrong grammar!\nUsage: mkdir folder_name/folder_path`,  if_print_raw_to_window: false,
            }
        }
    },
    description: 'Make a new directory.\nUsage: mkdir folder_name/folder_path'
};
```

This is the mkdir command, which creates a subfolder or subpath. Here, dealing with different form of path string is critical: some path strings can have leading "/" or "./".

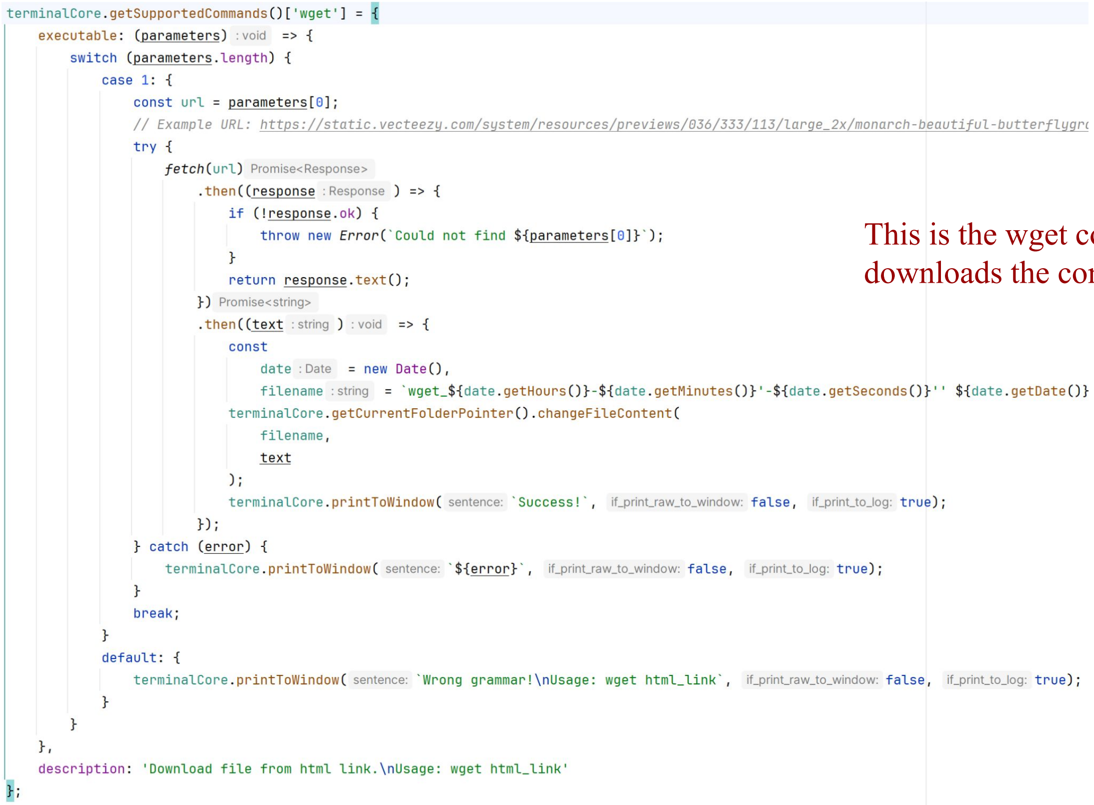
Vicayo Zhang - Screenshot 7 (Code & UI Explanation)



```
terminalCore.getSupportedCommands()['pwd'] = {
    executable: (_) : void  => {
        terminalCore.printToWindow(
            terminalCore.getCurrentFolderPointer().getFullPath(),
            if_print_raw_to_window: false,  if_print_to_log: true
        );
    },
    description: 'Print the current full path.'
};
```

This is the pwd command, which prints the full path of the current folder.

Vicayo Zhang - Screenshot 8 (Code & UI Explanation)

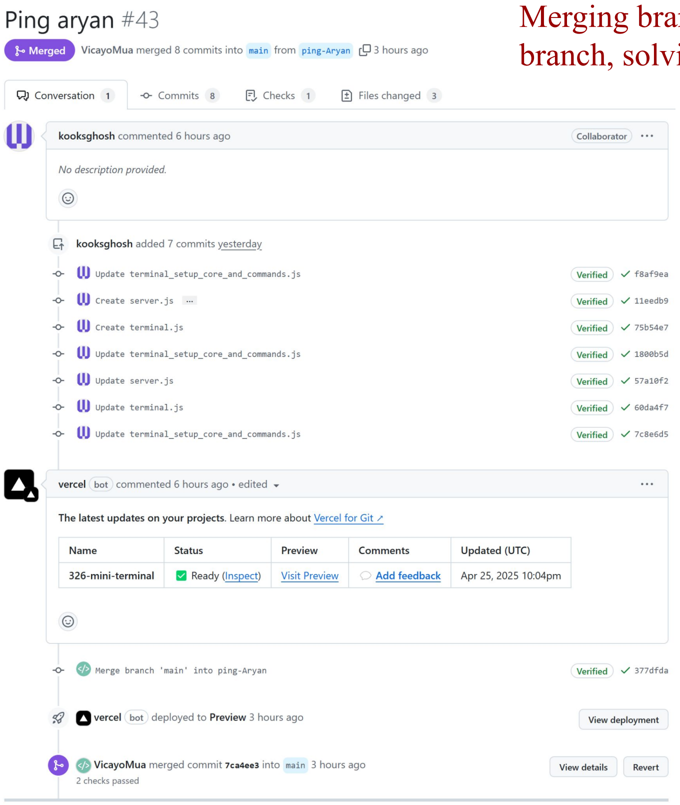


```
terminalCore.getSupportedCommands()['wget'] = {
    executable: (parameters) : void  => {
        switch (parameters.length) {
            case 1: {
                const url = parameters[0];
                // Example URL: https://static.vecteezy.com/system/resources/previews/036/333/113/large_2x/monarch-beautiful-butterflygr
                try {
                    fetch(url) Promise<Response>
                        .then((response : Response ) => {
                            if (!response.ok) {
                                throw new Error(`Could not find ${parameters[0]}`);
                            }
                            return response.text();
                        }) Promise<string>
                        .then((text : string ) : void  => {
                            const
                                date : Date  = new Date(),
                                filename : string  = `wget_${date.getHours()}-${date.getMinutes()}'-${date.getSeconds()}'' ${date.getDate()}
                            terminalCore.getCurrentFolderPointer().changeFileContent(
                                filename,
                                text
                            );
                            terminalCore.printToWindow( sentence: `Success!`,  if_print_raw_to_window: false,  if_print_to_log: true);
                        });
                } catch (error) {
                    terminalCore.printToWindow( sentence: `${error}`,  if_print_raw_to_window: false,  if_print_to_log: true);
                }
                break;
            }
            default: {
                terminalCore.printToWindow( sentence: `Wrong grammar!\nUsage: wget html_link`,  if_print_raw_to_window: false,  if_print_to_log: true);
            }
        }
    },
    description: 'Download file from html link.\nUsage: wget html_link'
};
```

This is the wget command. Given a URL, it downloads the content that the URL is linked to.

Vicayo Zhang - Screenshot 9 (Code & UI Explanation)

Merging branches to the main branch, solving the conflicts.



Ping aryan #43

Merged · VicayoMua merged 8 commits into `main` from `ping-Aryan` 3 hours ago

kooksghosh commented 6 hours ago · Collaborator

No description provided.

kooksghosh added 7 commits yesterday

`Update terminal_setup_core_and_commands.js`	Verified	✓ `f8af9ea`
`Create server.js` …	Verified	✓ `11eedb9`
`Create terminal.js`	Verified	✓ `75b54e7`
`Update terminal_setup_core_and_commands.js`	Verified	✓ `1800b5d`
`Update server.js`	Verified	✓ `57a10f2`
`Update terminal.js`	Verified	✓ `60da4f7`
`Update terminal_setup_core_and_commands.js`	Verified	✓ `7c8e6d5`

vercel bot commented 6 hours ago • edited

The latest updates on your projects. Learn more about Vercel for Git ↗

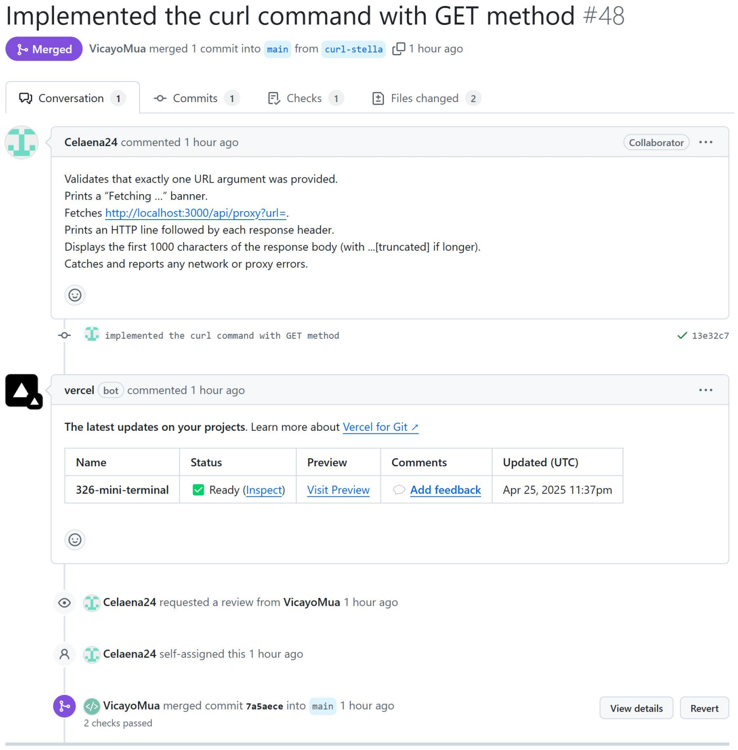
Name	Status	Preview	Comments	Updated (UTC)
326-mini-terminal	✅ Ready (Inspect)	Visit Preview	💬 **Add feedback**	Apr 25, 2025 10:04pm

`Merge branch 'main' into ping-Aryan` · Verified ✓ `377dfda`

vercel bot deployed to **Preview** 3 hours ago

VicayoMua merged commit **`7ca4ee3`** into `main` 3 hours ago

2 checks passed



Implemented the curl command with GET method #48

Merged · VicayoMua merged 1 commit into `main` from `curl-stella` 1 hour ago

Celaena24 commented 1 hour ago · Collaborator

Validates that exactly one URL argument was provided.
Prints a "Fetching …" banner.
Fetches http://localhost:3000/api/proxy?url=.
Prints an HTTP line followed by each response header.
Displays the first 1000 characters of the response body (with …[truncated] if longer).
Catches and reports any network or proxy errors.

`implemented the curl command with GET method` ✓ `13e32c7`

vercel bot commented 1 hour ago

The latest updates on your projects. Learn more about Vercel for Git ↗

Name	Status	Preview	Comments	Updated (UTC)
326-mini-terminal	✅ Ready (Inspect)	Visit Preview	💬 **Add feedback**	Apr 25, 2025 11:37pm

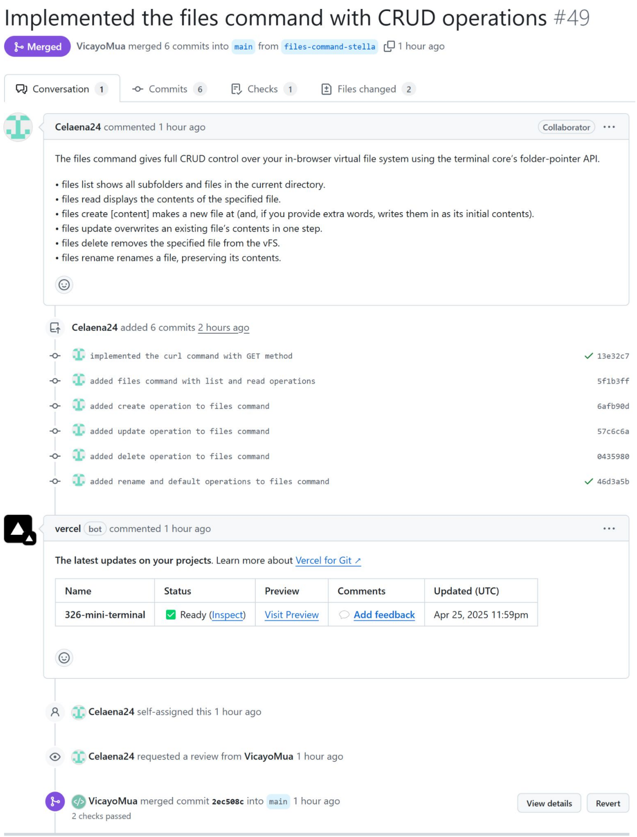
Celaena24 requested a review from **VicayoMua** 1 hour ago

Celaena24 self-assigned this 1 hour ago

VicayoMua merged commit **`7a5aece`** into `main` 1 hour ago

2 checks passed

Vicayo Zhang - Screenshot 9 (Code & UI Explanation)



Implemented the files command with CRUD operations #49

Merged VicayoMua merged 6 commits into `main` from `files-command-stella` 1 hour ago

Celaena24 commented 1 hour ago

The files command gives full CRUD control over your in-browser virtual file system using the terminal core's folder-pointer API.

- files list shows all subfolders and files in the current directory.
- files read displays the contents of the specified file.
- files create [content] makes a new file at (and, if you provide extra words, writes them in as its initial contents).
- files update overwrites an existing file's contents in one step.
- files delete removes the specified file from the vFS.
- files rename renames a file, preserving its contents.

Celaena24 added 6 commits 2 hours ago

```
implemented the curl command with GET method                 ✓ 13e32c7
added files command with list and read operations              5f1b3ff
added create operation to files command                        6afb90d
added update operation to files command                        57c6c6a
added delete operation to files command                        0435980
added rename and default operations to files command         ✓ 46d3a5b
```

vercel bot commented 1 hour ago

The latest updates on your projects. Learn more about Vercel for Git ↗

Name	Status	Preview	Comments	Updated (UTC)
326-mini-terminal	✅ Ready (Inspect)	Visit Preview	Add feedback	Apr 25, 2025 11:59pm

Celaena24 self-assigned this 1 hour ago

Celaena24 requested a review from **VicayoMua** 1 hour ago

VicayoMua merged commit **2ec508c** into `main` 1 hour ago
2 checks passed

Merging branches to the main branch, solving the conflicts.

Vicayo Zhang - Challenges and Insights

It’s challenging to implement the core services, especially the file system handler, since we need to combine the real-life application cases and the efficiency of the APIs.

After everything (in the core services) is setup, we can efficiently manage the files in the terminal, upload local files to the terminal file system.

Later, we may add the support of using WebAssembly to program new commands.

Vicayo Zhang - Future Improvements

In the future, we can add more practical and interesting commands, such as touch, cp, rename, and cd.

Actually, I don’t think the file system handler is perfect because it doesn’t support complex path strings, such as “/abc/../././abc/../..” Thus, we can absolutely update our core services.

Aryan Ghosh - Assigned Work Summary

Tasks Completed

PR Closed: https://github.com/VicayoMua/326_Mini_Terminal/pull/43

- Designed and implemented full-stack integration for ping command
- Debugged backend execution errors, added timeout handling
- Validated exec() responses and piped them into terminal UI
- Collaborated on front-end command dispatching and output formatting
- Verified backend routes and API endpoint with browser and CLI tools

server.js

- Created and configured Express server
  Set up /api/run endpoint integration
- Enabled CORS support for cross-origin requests

routes/terminal.js

- Implemented secure backend execution of ping command using exec()
- Added logging for debugging and safety checks
- Limited command access to only ping for security

terminal_setup_core_and_commands.js

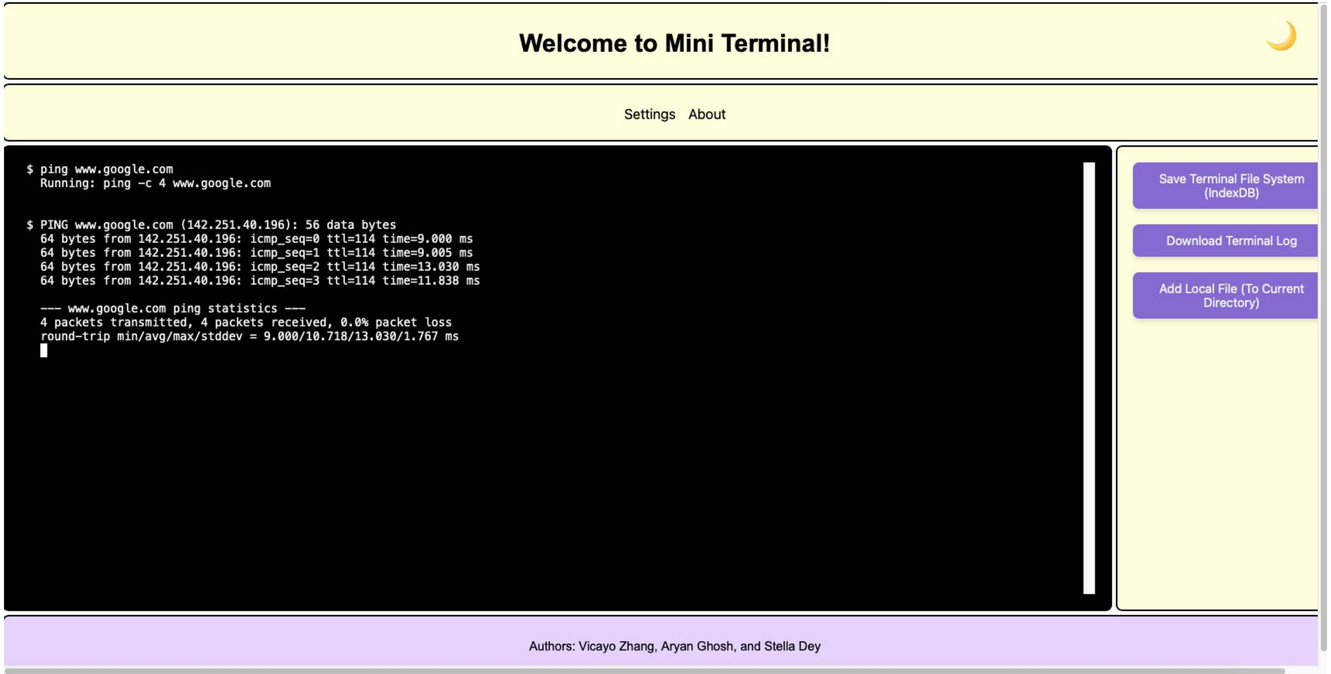
- Registered new ping command in the terminal interface
- Implemented fetch() logic to call backend endpoint
- Enhanced terminal UX by parsing and displaying command output

Aryan Ghosh - Feature Demonstration

Feature Implemented: ping Command

- Added full-stack support for the ping command in the Mini Terminal
- Allows user to execute ping [hostname] in the terminal UI
- Backend securely executes ping -c 4 and returns real-time results
- Output is displayed directly in the terminal interface with formatting

Branch Name: `ping-Aryan`

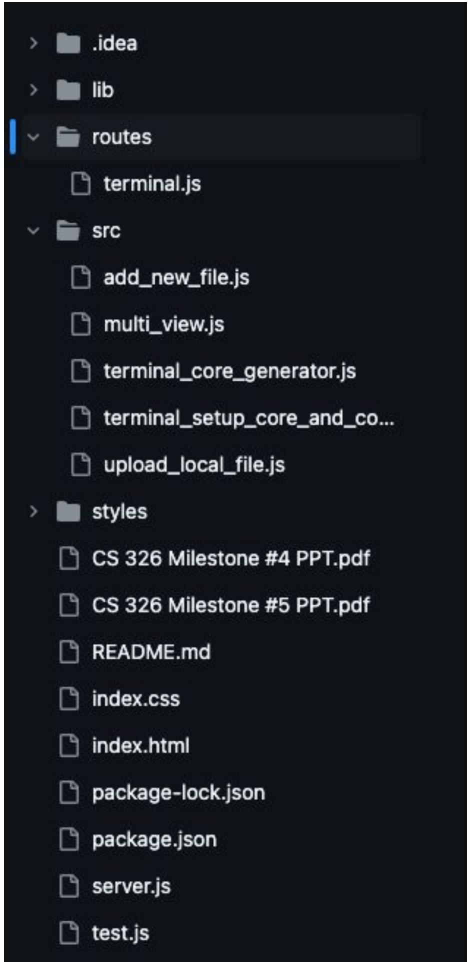


Welcome to Mini Terminal!
Settings
About
Save Terminal File System (IndexDB)
Download Terminal Log
Add Local File (To Current Directory)

```
$ ping www.google.com
  Running: ping -c 4 www.google.com


$ PING www.google.com (142.251.40.196): 56 data bytes
  64 bytes from 142.251.40.196: icmp_seq=0 ttl=114 time=9.000 ms
  64 bytes from 142.251.40.196: icmp_seq=1 ttl=114 time=9.005 ms
  64 bytes from 142.251.40.196: icmp_seq=2 ttl=114 time=13.030 ms
  64 bytes from 142.251.40.196: icmp_seq=3 ttl=114 time=11.838 ms

  --- www.google.com ping statistics ---
  4 packets transmitted, 4 packets received, 0.0% packet loss
  round-trip min/avg/max/stddev = 9.000/10.718/13.030/1.767 ms
```

Authors: Vicayo Zhang, Aryan Ghosh, and Stella Dey

Aryan Ghosh - Code Structure & Organization



```
> .idea
> lib
v routes
    terminal.js
v src
    add_new_file.js
    multi_view.js
    terminal_core_generator.js
    terminal_setup_core_and_co...
    upload_local_file.js
> styles
  CS 326 Milestone #4 PPT.pdf
  CS 326 Milestone #5 PPT.pdf
  README.md
  index.css
  index.html
  package-lock.json
  package.json
  server.js
  test.js
```

Front-End & Back-End Separation

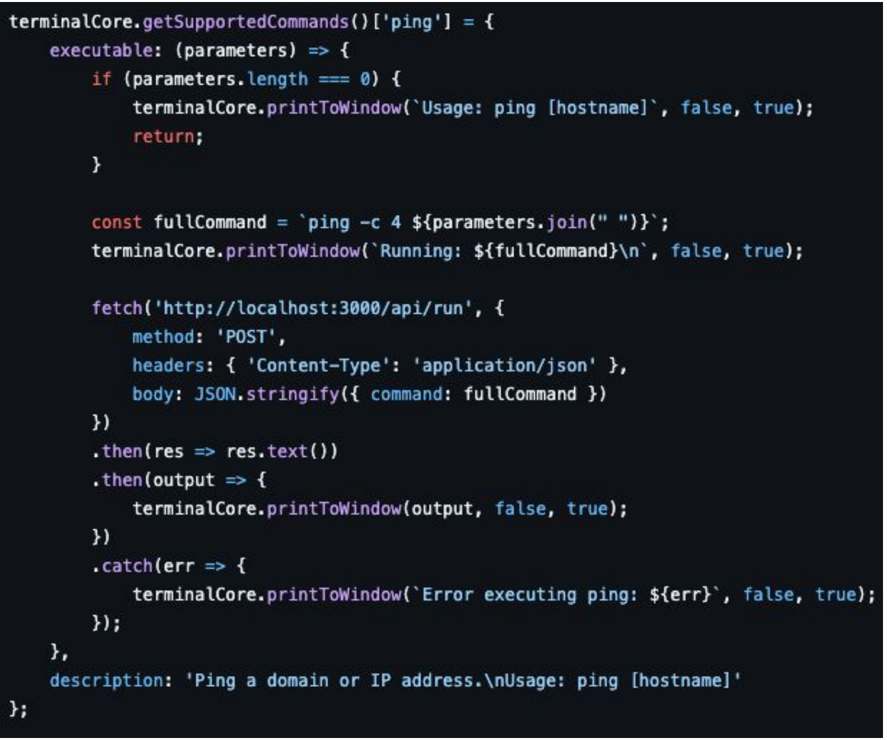
- **Front-End (User Interface & Logic)**:
  - Located inside the src/ folder.
  - Manages terminal generation, user input, command parsing, and display output.
  - Communicates with the backend through HTTP fetch() requests.
- **Back-End (Server-Side APIs)**:
  - Located at project root (server.js) and routes/ directory.
    Handles execution of safe system commands (ping) and returns output.
  - Ensures API security (only allows safe commands).
- Clear API communication between src/JavaScript and routes/terminal.js endpoints.

Critical Components Labeling

- **server.js**
  - Sets up the Express server and API route handling.
  - Located in the project root directory (/).

- **routes/terminal.js**
  - Defines the /api/run POST endpoint to execute the ping command securely.
  - Located inside the /routes/ folder.
- **src/terminal_core_generator.js**
  - Handles the creation of the terminal window interface.
  - Manages the in-browser file system logic (folders, files, commands).
- **src/terminal_setup_core_and_commands.js**
  - Registers terminal commands like ping, ls, mkdir, and others.
  - Links terminal input to command execution and fetch calls to the backend.
- **index.html**
  - Loads the front-end application (terminal UI) into the browser.
  - Links to the necessary JavaScript and CSS files.
- **styles/ folder**
  - Contains CSS files used for styling the terminal window and UI components.

Aryan Ghosh - Front End Implementation



```
terminalCore.getSupportedCommands()['ping'] = {
    executable: (parameters) => {
        if (parameters.length === 0) {
            terminalCore.printToWindow(`Usage: ping [hostname]`, false, true);
            return;
        }

        const fullCommand = `ping -c 4 ${parameters.join(" ")}`;
        terminalCore.printToWindow(`Running: ${fullCommand}\n`, false, true);

        fetch('http://localhost:3000/api/run', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ command: fullCommand })
        })
        .then(res => res.text())
        .then(output => {
            terminalCore.printToWindow(output, false, true);
        })
        .catch(err => {
            terminalCore.printToWindow(`Error executing ping: ${err}`, false, true);
        });
    },
    description: 'Ping a domain or IP address.\nUsage: ping [hostname]'
};
```

UI SCREENSHOT ON FEATURE DEMONSTRATION SLIDE

Front-End Structure Overview

- src/terminal_core_generator.js
  - Generates the terminal window, handles input, output, and filesystem.
- src/terminal_setup_core_and_commands.js
  - Registers commands like ping, sets up fetch-based API calls.
- styles/
  - Styles terminal window appearance (e.g., fonts, colors, layout).
- index.html
  - Loads the terminal and connects front-end scripts.

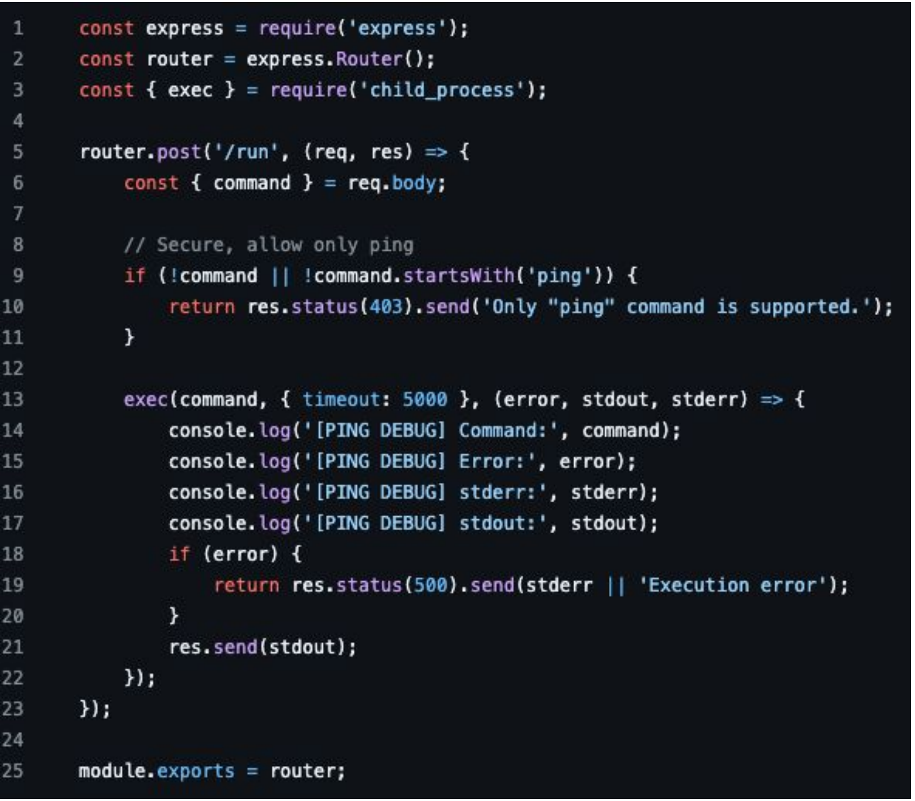
Integration with Back-End

- The fetch() call sends user-typed commands (like ping) to /api/run on the backend.
- Backend (routes/terminal.js) processes the command and returns execution output.
- Front-end then prints the output back inside the terminal window.

Challenges Faced & Solutions

- **CORS issues when running from file://**
  → Resolved by enabling CORS in the Express backend using the cors middleware.
- **fetch() request failing silently**
  → Fixed by using the full backend URL (http://localhost:3000/api/run) instead of a relative path.
- **Ping command running indefinitely on macOS**
  → Solved by adding -c 4 to limit ping to 4 packets and ensure proper backend timeout behavior.
- **Backend exec() throwing execution errors**
  → Added error logging and better frontend .catch() handling to display messages in the terminal.
- **Output formatting in terminal UI**
  → Used printToWindow() and newline formatting to keep ping results readable and consistent.

Aryan Ghosh - Back-End Implementation



```
const express = require('express');
const router = express.Router();
const { exec } = require('child_process');

router.post('/run', (req, res) => {
    const { command } = req.body;

    // Secure, allow only ping
    if (!command || !command.startsWith('ping')) {
        return res.status(403).send('Only "ping" command is supported.');
    }

    exec(command, { timeout: 5000 }, (error, stdout, stderr) => {
        console.log('[PING DEBUG] Command:', command);
        console.log('[PING DEBUG] Error:', error);
        console.log('[PING DEBUG] stderr:', stderr);
        console.log('[PING DEBUG] stdout:', stdout);
        if (error) {
            return res.status(500).send(stderr || 'Execution error');
        }
        res.send(stdout);
    });
});

module.exports = router;
```

Back-End Structure Overview

- **server.js**
  - Sets up the Express app and binds /api routes.
  - Loads middlewares (CORS, JSON parsing).
- **routes/terminal.js**
  - Defines /api/run route.
  - Handles POST requests to execute and return system commands.
- **models/**
  - (Not needed yet for ping; would be used if expanding to file storage.)

Integration with Front-End

- Front-end sends a POST request to /api/run with the ping command.
- Backend securely executes the ping using child_process.exec().
- Backend captures and sends the command output as HTTP response.
- Front-end receives and prints the response inside the terminal UI.

Challenges Faced & Solutions

- **Handling infinite ping output (macOS/Linux differences)**
  → Added -c 4 to limit ping to 4 packets.
- **Backend server not accessible from file:// front-end**
  → Installed and used CORS middleware.
- **Execution timeout issues with long-running commands**
  → Set a timeout limit inside exec() options (5000 ms).
- **Graceful handling of backend errors**
  → Implemented .catch on front-end and status code responses on backend.

Aryan Ghosh - Challenges and Insights

Reflections on Obstacles Faced

- Encountered CORS policy restrictions when connecting front-end to backend from file:// — resolved with CORS middleware.
- Faced issues with backend command execution timing out — learned to control system command behavior using flags like -c 4 for ping.
- Debugging asynchronous fetch failures and backend errors improved my understanding of frontend-backend communication flow.
- Ensured backend security by filtering allowed commands, avoiding arbitrary execution risks.

Key Takeaways from Collaborative Teamwork

- Learned the importance of clear file structure and separation of concerns (front-end vs back-end).
- GitHub Issues and Pull Requests provided clarity on task division and accountability within the team.
- Regular communication and code reviews helped spot integration issues early.
- Following consistent coding standards (naming, commenting) made merging different parts smoother.
- Teamwork taught me to be flexible: adjusting frontend APIs after backend feedback and vice versa.

Aryan Ghosh - Future Improvements

Proposed Improvements

- **Real-Time Streaming of Command Output**
  - **Description:**
    Currently, the entire ping result is displayed only after the command completes execution. In future versions, implement real-time line-by-line streaming using child_process.spawn() instead of exec(). This will make the terminal feel more responsive and closer to a real-world terminal experience.
  - **GitHub Issue:** https://github.com/VicayoMua/326_Mini_Terminal/issues/44?issue=VicayoMua%7C326_Mini_Terminal%7C45
- **Frontend Command History Navigation**
  - **Description:**
    Allow users to use the Up and Down arrow keys to navigate through previously entered commands (command history). This feature will improve usability by mimicking traditional terminal behavior and help users re-execute previous commands easily.
  - **GitHub Issue:** https://github.com/VicayoMua/326_Mini_Terminal/issues/44?issue=VicayoMua%7C326_Mini_Terminal%7C46

Stella Dey - Assigned Work Summary

Assigned Issues:

- #39: Rename and delete file functionality
https://github.com/VicayoMua/326_Mini_Terminal/issues/39
- #50 Add “files” command for virtual file system CRUD
https://github.com/VicayoMua/326_Mini_Terminal/issues/50
- #51: Add curl command to fetch http resources
https://github.com/VicayoMua/326_Mini_Terminal/issues/51

Tasks Completed:

- Created a files command to perform all virtual-FS CRUD operations—list, read, create, update, delete, and rename—directly from the terminal.
- Leveraged the terminal core’s folder-pointer API to manage file and folder actions entirely in-browser, with IndexedDB persistence.
- Added a curl command that routes requests through the back-end proxy (/api/proxy), validates a single URL argument, and prints the HTTP status line, response headers, and a 1 000-character body snippet (with graceful error reporting).
- Updated and color-matched the project’s Gantt timeline to include Ping, Wget, Curl, and Files features, ensuring dates align exactly with the original chart.

Closed PRs:

- https://github.com/VicayoMua/326_Mini_Terminal/pull/49
- https://github.com/VicayoMua/326_Mini_Terminal/pull/48

Commits Authored:

- 13e32c7: Implemented the CURL command with GET method
- 5f1b3ff: Added files command with list and read operations
- 6afb90d: Added CREATE operation to files command
- 57c6c6a:Added UPDATE operation to files command
- 0435980: Added DELETE operation to files command
- 46d3a5b: Added RENAME and default operations to files command

Feature Demonstration

Feature Description:

I implemented two new terminal commands for enhanced functionality:

1. Files command - full virtual-FS CRUD in the browser

- files list
- files read <path>
- files create <path> [content]
- files update <path> <content>
- files delete <path>
- files rename <old> <new>

All operations use the folder-pointer API with IndexedDB persistence.

2. Curl command – HTTP GET via our backend proxy:

- Validates a single URL argument
- Fetches through /api/proxy?url=… to avoid CORS
- Prints HTTP <status> <statusText>, all response headers, and a 1 000-char body snippet
- Reports network or proxy errors

Feature Type:

1. Advanced Integration
2. Basic Integration

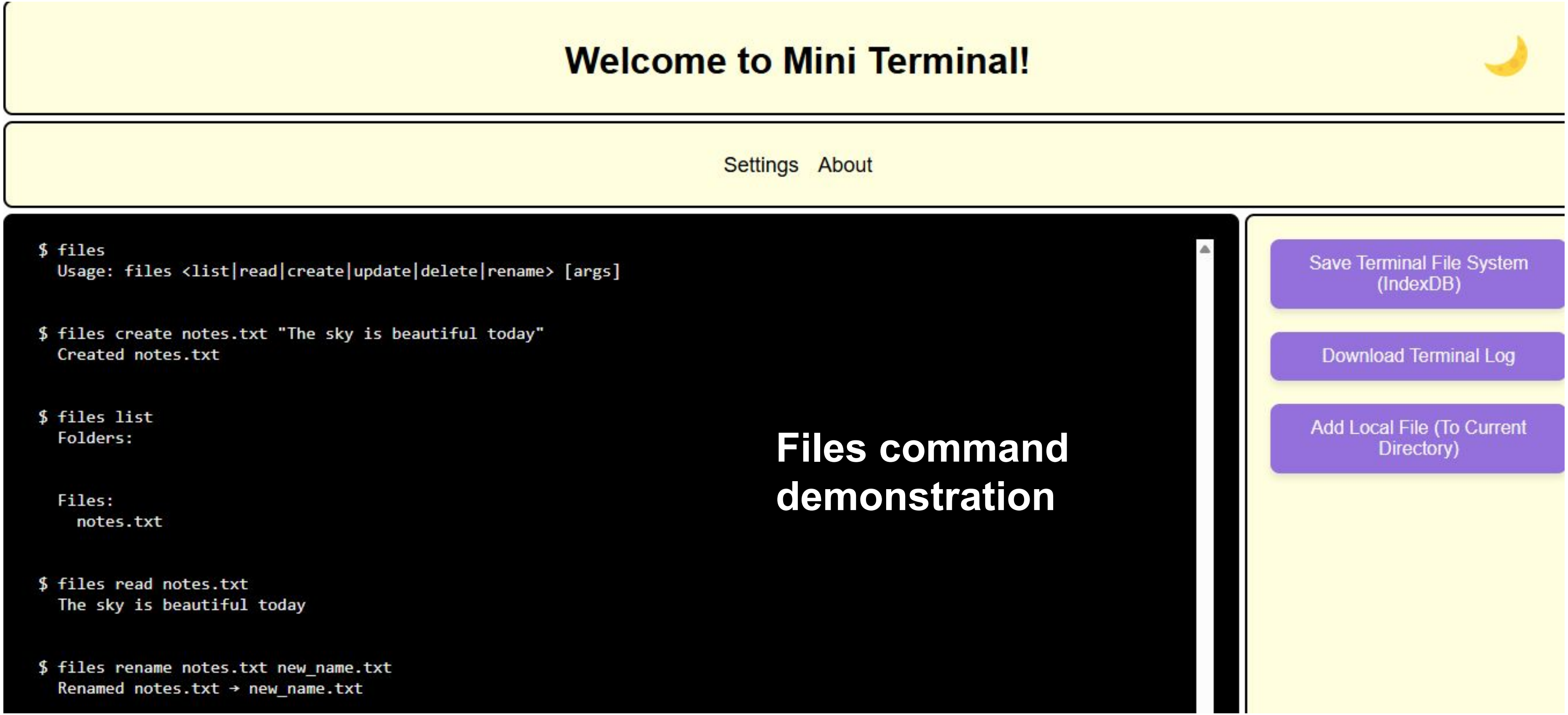
Completion Progress:

Both commands are 100% complete and functional, with full error handling, help/man entries, and persistence where applicable.

Git Branches:

https://github.com/VicayoMua/326_Mini_Terminal/tree/files-command-stella
https://github.com/VicayoMua/326_Mini_Terminal/tree/curl-stella

Stella Dey - Screenshot

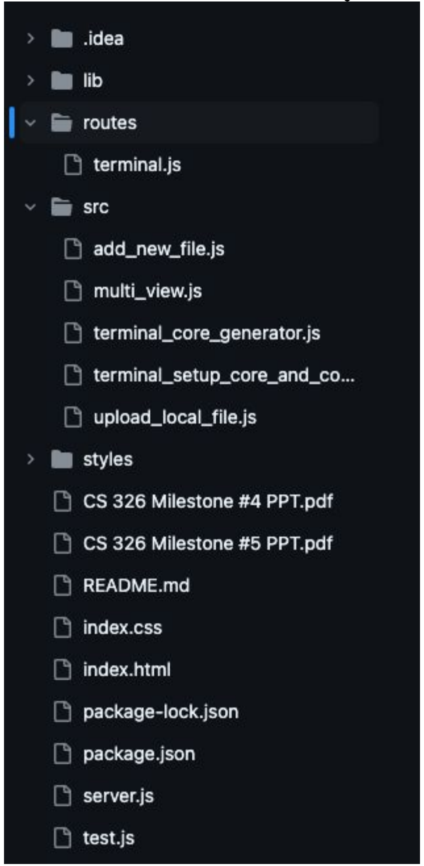


Welcome to Mini Terminal!
Settings
About
Save Terminal File System (IndexDB)
Download Terminal Log
Add Local File (To Current Directory)

```
$ files
  Usage: files <list|read|create|update|delete|rename> [args]


$ files create notes.txt "The sky is beautiful today"
  Created notes.txt


$ files list
  Folders:


  Files:
    notes.txt


$ files read notes.txt
  The sky is beautiful today


$ files rename notes.txt new_name.txt
  Renamed notes.txt → new_name.txt
```

Files command demonstration

Stella Dey - Code Structure and Organization



```
> .idea
> lib
v routes
    terminal.js
v src
    add_new_file.js
    multi_view.js
    terminal_core_generator.js
    terminal_setup_core_and_co...
    upload_local_file.js
> styles
  CS 326 Milestone #4 PPT.pdf
  CS 326 Milestone #5 PPT.pdf
  README.md
  index.css
  index.html
  package-lock.json
  package.json
  server.js
  test.js
```

Front-End (Browser)

- Lives under public/ and its subfolders.
- All terminal UI logic, virtual-FS (files), and HTTP commands (curl) are implemented in JS there.
- Persists data client-side via IndexedDB

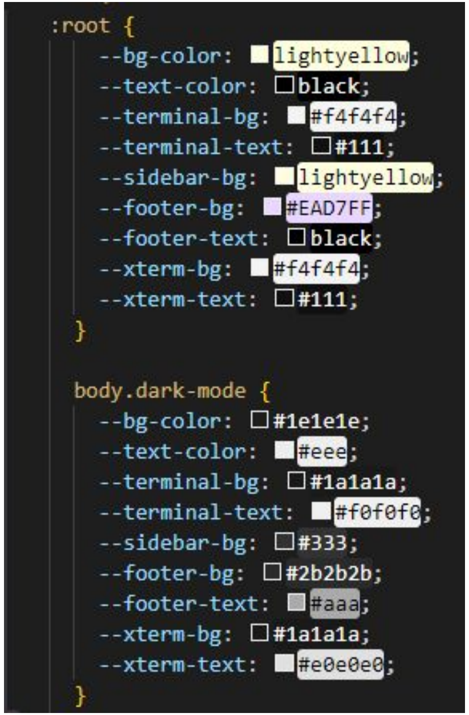
Back-End (Server)

- Entry point: server.js, which sets up CORS, JSON parsing, and mounts /api routes.
- All server logic lives in routes/terminal.js (e.g. /api/run for ping, /api/proxy for curl).
- No front-end files are imported on the server—strict client/server boundary.

3. Critical Component Labeling

- **public/src/terminal_core_generator.js**
  Core terminal initialization and folder-pointer API.
- **public/src/terminal_setup_core_and_commands.js**
  Registers built-in commands (help, files, curl, ping, etc.) and binds UI buttons.
- **routes/terminal.js**
  Implements secure ping execution and HTTP proxy endpoint.
- **server.js**
  Bootstraps the Express server and mounts the API routes.

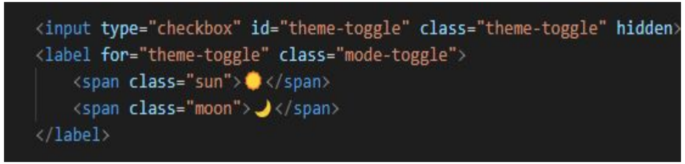
Stella Dey - Frontend Implementation



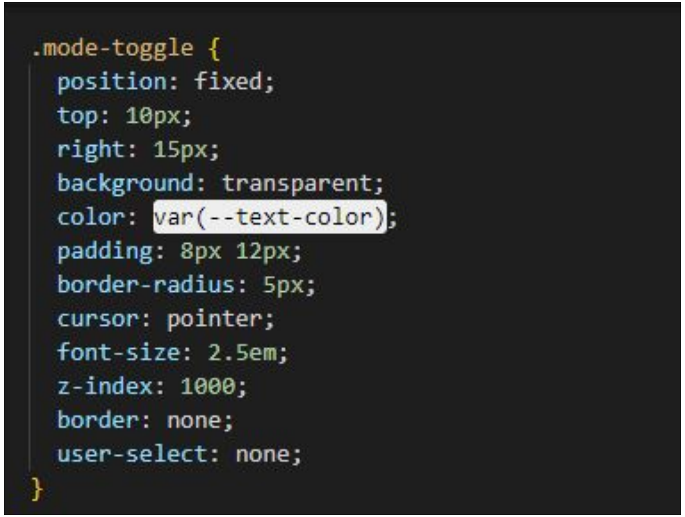
```
:root {
    --bg-color: lightyellow;
    --text-color: black;
    --terminal-bg: #f4f4f4;
    --terminal-text: #111;
    --sidebar-bg: lightyellow;
    --footer-bg: #EAD7FF;
    --footer-text: black;
    --xterm-bg: #f4f4f4;
    --xterm-text: #111;
  }

  body.dark-mode {
    --bg-color: #1e1e1e;
    --text-color: #eee;
    --terminal-bg: #1a1a1a;
    --terminal-text: #f0f0f0;
    --sidebar-bg: #333;
    --footer-bg: #2b2b2b;
    --footer-text: #aaa;
    --xterm-bg: #1a1a1a;
    --xterm-text: #e0e0e0;
  }
```

The theme toggle integrates into the overall UI by dynamically adding or removing a dark-mode class on the <body> element. All visual components—including the header, terminal container, sidebar, and footer—reference CSS variables (like --bg-color and --text-color) for styling. When the dark-mode class is active, these variables are redefined, allowing the entire interface to switch themes consistently with minimal code duplication.

This approach ensures centralized styling control and keeps the UI modular and easy to maintain.

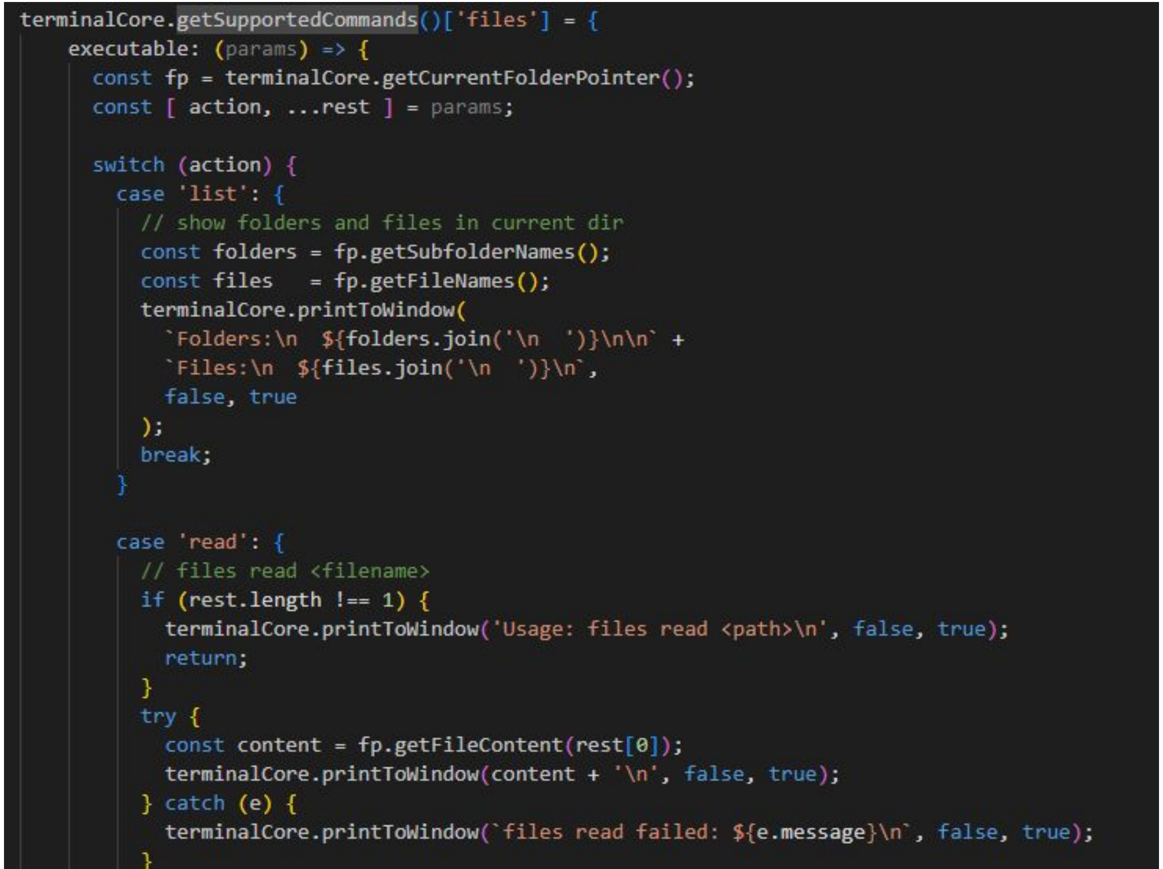


```
<input type="checkbox" id="theme-toggle" class="theme-toggle" hidden>
<label for="theme-toggle" class="mode-toggle">
    <span class="sun">🌞</span>
    <span class="moon">🌙</span>
</label>
```



```
.mode-toggle {
  position: fixed;
  top: 10px;
  right: 15px;
  background: transparent;
  color: var(--text-color);
  padding: 8px 12px;
  border-radius: 5px;
  cursor: pointer;
  font-size: 2.5em;
  z-index: 1000;
  border: none;
  user-select: none;
}
```

Stella Dey - Backend Implementation



```
terminalCore.getSupportedCommands()['files'] = {
    executable: (params) => {
      const fp = terminalCore.getCurrentFolderPointer();
      const [ action, ...rest ] = params;

      switch (action) {
        case 'list': {
          // show folders and files in current dir
          const folders = fp.getSubfolderNames();
          const files   = fp.getFileNames();
          terminalCore.printToWindow(
            `Folders:\n  ${folders.join('\n  ')}\n\n` +
            `Files:\n  ${files.join('\n  ')}\n`,
            false, true
          );
          break;
        }

        case 'read': {
          // files read <filename>
          if (rest.length !== 1) {
            terminalCore.printToWindow('Usage: files read <path>\n', false, true);
            return;
          }
          try {
            const content = fp.getFileContent(rest[0]);
            terminalCore.printToWindow(content + '\n', false, true);
          } catch (e) {
            terminalCore.printToWindow(`files read failed: ${e.message}\n`, false, true);
          }
```

Structure & Integration

- **Routes**
  - All back-end endpoints live under routes/terminal.js and are mounted at /api in server.js.
- **Controllers**
  - Each route handler:
    1. **Validates** input (e.g. missing url parameter).
    2. **Calls** an upstream service (node-fetch).
    3. **Streams** status, headers, and body back to the front-end.
- **Middleware**
  - cors() and express.json() configured in server.js to support JSON APIs and cross-origin fetches.

Stella Dey - Challenges and Insights

Challenges & insights

The toughest bit was cleaning up the new curl proxy and the files command. Debugging URL validation, CORS hiccups, and edge-case rename logic ate hours, but stepping through little test scripts and adding verbose logs finally nailed it. Biggest lesson: kill one bug at a time and log everything early.

Collaborative Takeaways

Quick PR reviews and a bit of pair-debugging with the crew saved the day. Short feedback loops meant cleaner code and less stress, and I learned that clear commit messages plus tiny pull requests make teamwork way smoother.

Stella Dey - Future Improvements

Future Improvements & Next Steps:

○ Add file size and timestamp metadata to the files command output. Would make it feel more like a real filesystem and help users manage files better.

○ Add tests for edge cases like appending to an existing file or using invalid URLs in curl.