

CardanoVSC

Title: Plutus and Haskell Extension Integrated
within Visual Studio Code IDE.

Milestone 1: Scope and Design
Documentation.

Project Catalyst Fund 12.

Udai Solanki
Gaurav Raj
Aditya Solanki

December 2024

Contents

1	Introduction	3
2	Objective	3
3	Scope	3
3.1	Syntax highlighting and code completion	3
3.2	Cardano Blockchain Explorer API Integration	4
3.3	Smart Contract Deployment/Connecting Node	5
3.4	Wallet Management	5
3.5	Debugging Tools	5
4	Technical Specifications for CardanoVSC Extension.	5
4.1	Setup Environment	5
4.2	Develop Syntax Highlighting and Code Completion	6
4.3	Implement Cardano Blockchain Explorer API Integration	8
4.4	Smart Contract Deployment from VS Code IDE	13
4.5	Wallet Management	15
4.6	Debugging Tools for Smart Contracts and Wallet Transactions 19	
4.7	Code Examples and Documentation	21
5	UI/UX Mockups	21
6	Architecture diagram	25
7	Conclusion	25

1 Introduction

CardanoVSC is a Plutus and Haskell Extension integrated within Visual Studio(VS) Code Integrated Development Environment(IDE) designed to enhance the development experience for Plutus and Haskell programmers by offering:

- Syntax Highlighting.
- Code Completion.
- Cardano Blockchain Explorer Integration.
- Smart Contract Deployment/Cardano node connection for Preprod, Preview and Mainnet.
- Wallet Management.
- Debugger tool.

2 Objective

- Syntax Highlighting and Code Completion for Plutus and Haskell: Improve code readability and provide coding assistance.
- Cardano Blockchain Explorer API Integration: Enable interaction with the Cardano blockchain directly from VS Code.
- Smart Contract Deployment/Connecting Node: Facilitate the deployment of smart contracts and connection to a Cardano node. Allow user to select between Preprod, Preview and mainnet network.
- Wallet Management: Provide tools for creating and managing wallets.
- Debugging Tools for Plutus and Haskell: Enhance the debugging capabilities for Plutus smart contracts.

3 Scope

3.1 Syntax highlighting and code completion

The extension will be designed to provide comprehensive syntax highlighting for Plutus and Haskell programming language. By delivering precise and visually appealing syntax highlighting, the extension aims to enhance productivity and make coding more enjoyable. This includes all function, operators and variables being highlighted with different colors, as is the norm for syntax highlighting in Visual Studio Code for other languages.

We will also be providing code completion suggestions. If a user wants they can use the suggestion to auto complete their code. Auto completion will also

add closing brackets to any opening brackets for Plutus and Haskell. Our objective here is to provide clear and distinct visual cues for different elements of Plutus and Haskell code and to assist developers by suggesting possible completions for partially typed Plutus and Haskell code in Visual Studio Code. The extension will provide visually distinct and color-coded highlighting for the following elements:

Syntax Highlighting for Default Dark Modern Theme Colors:

- Keywords
- Strings
- Comments
- Variables
- Functions
- Numbers
- Operators
- Constants

Code Completion:

Provides context-aware suggestions to assist users in writing accurate and efficient code. The extension will offer intelligent, context-aware suggestions, including:

- Function names and parameters.
- Suggest Variable names.
- Suggest Data types.
- Suggest Code Snippets for common patterns.
- Auto complete brackets.

This makes coding on Visual Studio Code a more desirable experience for Plutus and Haskell developers in Cardano.

3.2 Cardano Blockchain Explorer API Integration

- Query blockchain data from one of the blockchain explorers to fetch data such as transactions, addresses, and block details. We will use CardanoScan for this due to its popularity.
- Present the fetched data in an intuitive format within the terminal, ensuring easy accessibility for developers.

3.3 Smart Contract Deployment/Connecting Node

- Allow users to deploy Plutus smart contracts directly from VS Code.
- Enable connecting to a Cardano node from inside of Visual Studio Code. We will use Blockfrost API to connect to Cardano node and deploy smart contracts on the Cardano blockchain. For this we will require a blockfrost API key.
- Deployment of smart contract is dependent on the node connection.

3.4 Wallet Management

- Provide users with the ability to create and manage Cardano wallets within the extension.
- Include features for: Wallet creation, Restore via mnemonic phrases, Viewing balances and sending and receiving transactions.

3.5 Debugging Tools

- Create debugging tools that allow developers to step through smart contract execution and transaction debugging.
- Step-through Debugging: Allows developers to execute smart contracts.

Transaction Debugging: Include features to analyze transaction inputs, outputs, and execution results.

Error Insights: Display detailed error messages.

4 Technical Specifications for CardanoVSC Extension.

4.1 Setup Environment

- First, use VS Code Extension Generator to scaffold a JavaScript/TypeScript project ready for development.

Run the following

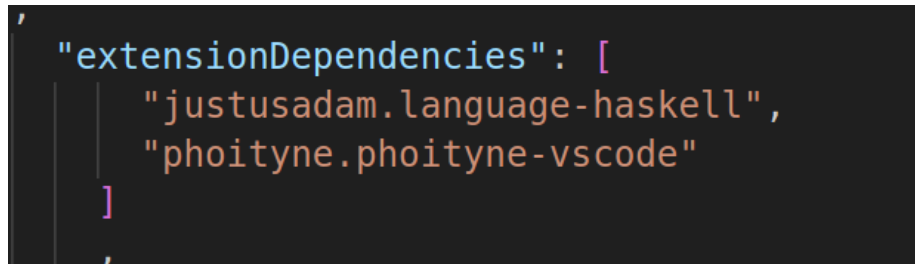
```
npx --package yo --package generator-code -- yo
code
```

to initialize building the extension. This will create the necessary project structure for building the extension.

- Review the Project Structure. Ensure the required files are set up correctly for development.

4.2 Develop Syntax Highlighting and Code Completion

- For Syntax Highlighting we add dependencies to the existing package.json file to include the Haskell language dependency provided to us by Justusadam.



```
'  
  "extensionDependencies": [  
    "justusadam.language-haskell",  
    "phoityne.phoityne-vscode"  
  ]  
,
```

Figure 1: External extension dependencies are defined in Dependencies section of package.json file inside of extensionDependencies.

- In Figure 1 we define "justusadam.language-haskell" as an extension dependency. This means we will be installing justusadam.language-haskell when we choose to start our extension. It will enable us to use Haskell syntax Highlighting.
- **Syntax Highlighting for Default Dark Modern Theme Colors:**
 - Keywords: Light Blue (#569CD6).
 - Strings: Light Green (#CE9178).
 - Comments: Grey (#6A9955).
 - Variables: White (#D4D4D4).
 - Functions: Yellow (#D4D4D4).
 - Numbers: Light Blue (#B5CEA8).
 - Operators: White (#D4D4D4).
 - Constants: Teal (#4EC9B0).
- **Syntax Highlighting for Default Light+ Theme:**
 - Keywords: Blue (#0000FF).
 - Strings: Green (#008000).
 - Comments: Grey (#008000).
 - Variables: Blue (#001080).
 - Functions: Brown (#795E26).
 - Numbers: Green (#098658).

- Operators: Black (#000000).
- Constants: Teal (#267F99).

```
import System.IO (hFlush, stdout)

main :: IO ()
main = do
    putStrLn "Enter your name:"
    hFlush stdout -- Ensure output is flushed
    name <- getLine
    putStrLn ("Hello, " ++ name ++ "!")
```

Figure 2: Syntax Highlighting (e.g., keywords, operators, comments) UI example.

VS Code syntax highlighter has predefined colors for syntax highlighting. However, it does not yet recognize Plutus-specific functions and keywords. By defining these Plutus-specific elements, the syntax highlighter will automatically apply the appropriate colors according to the theme. Since some colors may not be clearly visible in all themes, it is essential for theme creators to specify these colors accordingly. The default dark and light mode syntax highlighting color is given above, we will be using those colors for Plutus and Haskell syntax highlighting.

Code Completion

- Language Server Protocol (LSP): Utilize the Visual studio code's Haskell Language Server(HLS) to provide code completion features for Haskell and Plutus.

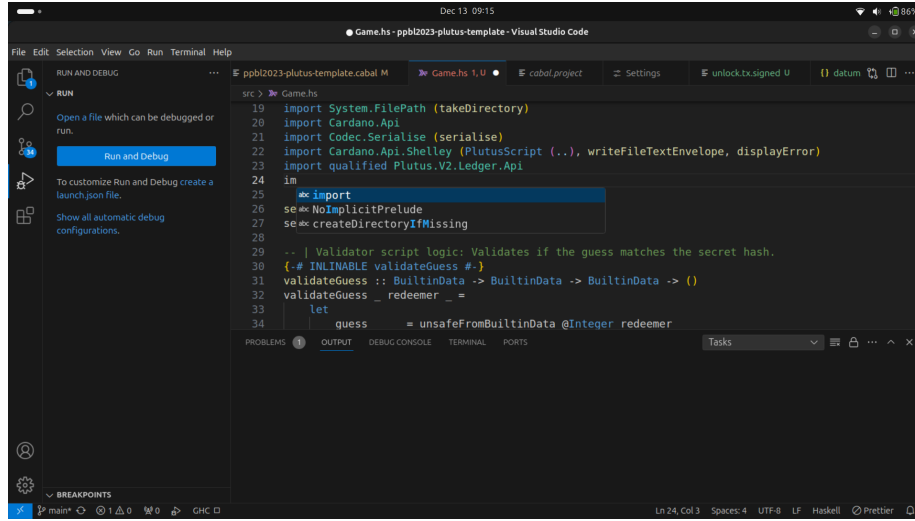


Figure 3: Code completion UI example.

- Custom Snippets: To create snippets for common Plutus constructs and patterns we use Completion.API() present in VS code to define and suggest Plutus and Haskell code snippets.
- Create boilerplate templates for Plutus Smart contract development that can be accessed using preset command for getting started.

4.3 Implement Cardano Blockchain Explorer API Integration

- We use the following list of API's to connect to a Cardano Blockchain Explorer and get the required information. It requires a free API key from CardanoScan to connect to it. Each user should provide their own Free API key in the URL.

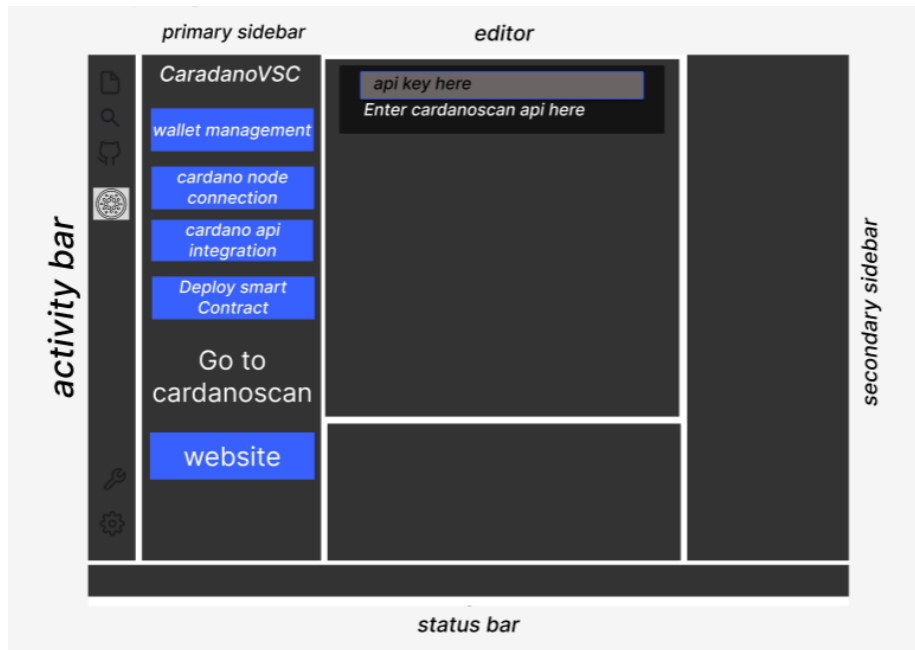


Figure 4: Cardanoscan API UI.

Here is the list of API's:

Get Block Details:

Query Parameter: Block hash(string), blockHeight(integer), absoluteSlot(integer), epoch(integer), slot(integer).

```
1 curl \
2 -X GET https://api.cardanoscan.io/{User_api_key}/v1/block
```

Listing 1: Bash Example: Get Block Details

Get latest block details:

```
1 curl \
2 -X GET https://api.cardanoscan.io/{User_api_key}/v1/block/latest
```

Listing 2: Bash Example: Get latest block details

Get address balance:

Query parameters: Address(integer).

```
1 curl \
```

```

2 -X GET https://api.cardanoscan.io/{
  User_api_key}/v1/address/balance?address=string

```

Listing 3: Bash example: Get address balance.

Get pool details:

Query parameters: PoolID(string).

```

1 curl \
2 -X GET https://api.cardanoscan.io/{User_api_key}/
  v1/pool?poolId=string

```

Listing 4: Bash Example: Get pool id.

Get pool stats:

Query parameters: poolId

```

1 curl \
2 -X GET https://api.cardanoscan.io/{User_api_key}/
  v1/pool/stats?poolId=string

```

Listing 5: Bash example: Get Pool Stats

Get pools list:

Query parameters: pageNo (integer), search(string), retiredPools (boolean), sortBy(string), order(string), limit (integer).

```

1 curl \
2 -X GET https://api.cardanoscan.io/{User_api_key}/
  v1/pool/list?pageNo=1

```

Listing 6: Bash example: Get Pool List.

Get pools which are set to expire:

Query parameters: pageNo(integer), limit(integer).

```

1 curl \
2 -X GET https://api.cardanoscan.io/{User_api_key}/
  v1/pool/list/expiring?pageNo=42
3

```

Listing 7: Bash example: Get pools which are set to expire.

Get expired pools:

Query parameters: pageNo(integer), limit(integer).

```

1 curl \
2 -X GET https://api.cardanoscan.io/{User_api_key}/
  v1/pool/list/expired?pageNo=1

```

Listing 8: Bash example: Get expired pools.

Get asset details

Query parameters: `assetId(string)`, `fingerprint(string)`.

```
1 curl \
2 -X GET https://api.cardanoscan.io/{User_api_key}/
  v1/asset
```

Listing 9: Bash example: Get asset details.

Get assets by policyId

Query parameters: `policyId(string)`, `pageNo(integer)`, `limit integer`.

```
1 curl \
2 -X GET https://api.cardanoscan.io/{User_api_key}/
  v1/asset/list/byPolicyId?policyId=string&pageNo
  =42
```

Listing 10: Bash example: Get assets by policyId.

Get assets by address:

Query parameters: `address(string)`, `pageNo(integer)`, `limit (integer)`.

```
1 curl \
2 -X GET https://api.cardanoscan.io/{User_api_key}/
  v1/asset/list/byAddress?address=string&pageNo=2
```

Listing 11: Bash example: Get assets by address.

Get transaction details:

Query parameters: `hash(string)`.

```
1 curl \
2 -X GET https://api.cardanoscan.io/{User_api_key}/
  v1/transaction?hash=string
```

Listing 12: Bash example: Get transaction details.

Get transaction list by address:

Query parameters: `address(string)`, `pageNo(integer)`, `limit (integer)`, `order(string)`.

```
1 curl \
2 -X GET https://api.cardanoscan.io/{User_api_key}/
  v1/transaction/list?address=string&pageNo=42
```

Listing 13: Bash example: Get transaction details.

Get stake key details:

Query parameters: `rewardAddress(string)`.

```
1 curl \
2 -X GET https://api.cardanoscan.io/{User_api_key}/
  v1/rewardAccount?rewardAddress=string
```

Listing 14: Bash example: Get stake key details.

Get addresses associated with a stake key:

Query parameters: rewardAddress(string), pageNo(integer), limit(integer).

```
1 curl \  
2 -X GET https://api.cardanoscan.io/{User_api_key}/  
v1/rewardAccount/addresses?rewardAddress=string  
&pageNo=42
```

Listing 15: Bash example: Get addresses associated with a stake key.

Get network details:

```
1 curl \  
2 -X GET https://api.cardanoscan.io/{User_api_key}/  
v1/network/state
```

Listing 16: Bash example: Get network Details.

Get network protocol details:

```
1 curl \  
2 -X GET https://api.cardanoscan.io/{User_api_key}/  
v1/network/protocolParams
```

Listing 17: Bash example: Get network protocol details.

Get CCHot details:

```
1 curl \  
2 -X GET https://api.cardanoscan.io/{User_api_key}/  
v1/governance/ccHot?hotHex=string
```

Listing 18: Bash example: Get CCHOT details.

Get CCMember details:

```
1 curl \  
2 -X GET https://api.cardanoscan.io/{User_api_key}/  
v1/governance/ccMember?coldHex=string
```

Listing 19: Bash example: Get CCMember details.

Get Committee Information:

```
1 curl \  
2 -X GET https://api.cardanoscan.io/{User_api_key}/  
v1/governance/committee
```

Get dRep Information:

```
1 curl \  
2 -X GET https://api.cardanoscan.io/{User_api_key}/  
v1/governance/dRep?dRepId=string
```

Listing 20: Bash example: Get dRep Information.

Get Governance Action:

Query parameters: `actionId(string)`.

```
1 curl \  
2 -X GET https://api.cardanoscan.io/{User_api_key}/  
   v1/governance/action?actionId=string
```

Listing 21: Bash example: Get Governance Action.

- We also provide a alternate way for people to interact with the Cardano Explorer Website(CardanoScan).
 - **Step 1:** To open cardanoscan website Click the **website** button under Go to cardanoscan.

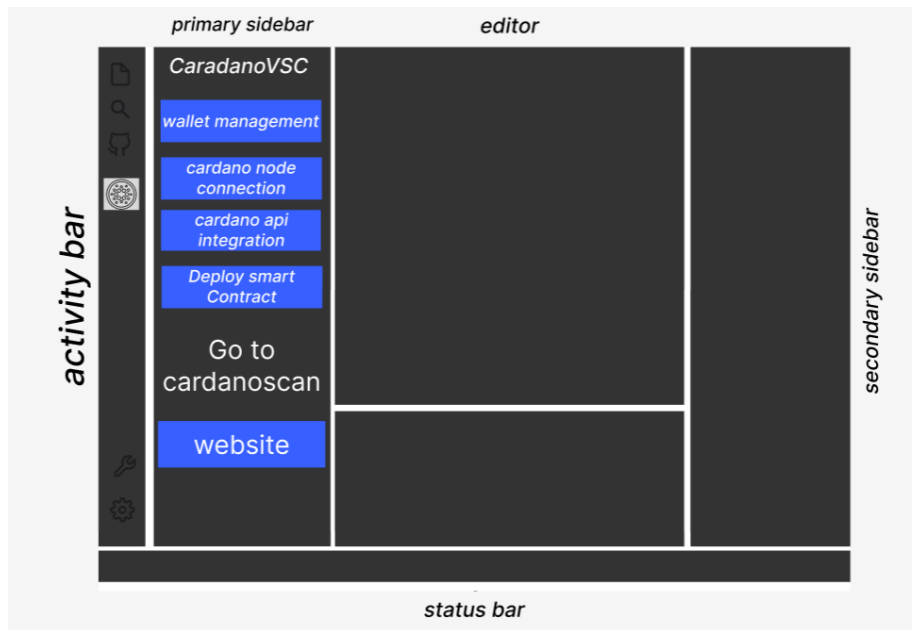


Figure 5: Go to CardanoScan website UI mockup.

- **Step 2:** Then we call default browser to render the CardanoScan.io website.

4.4 Smart Contract Deployment from VS Code IDE

Connecting to a Node

- Allow users to connect to Preprod, Preview, and Mainnet network on Cardano using blockfrost API and API key for node connection.

- Prompt user for a input of their Blockfrost API key.
- Using blockfrost allow the user to connect to the node.

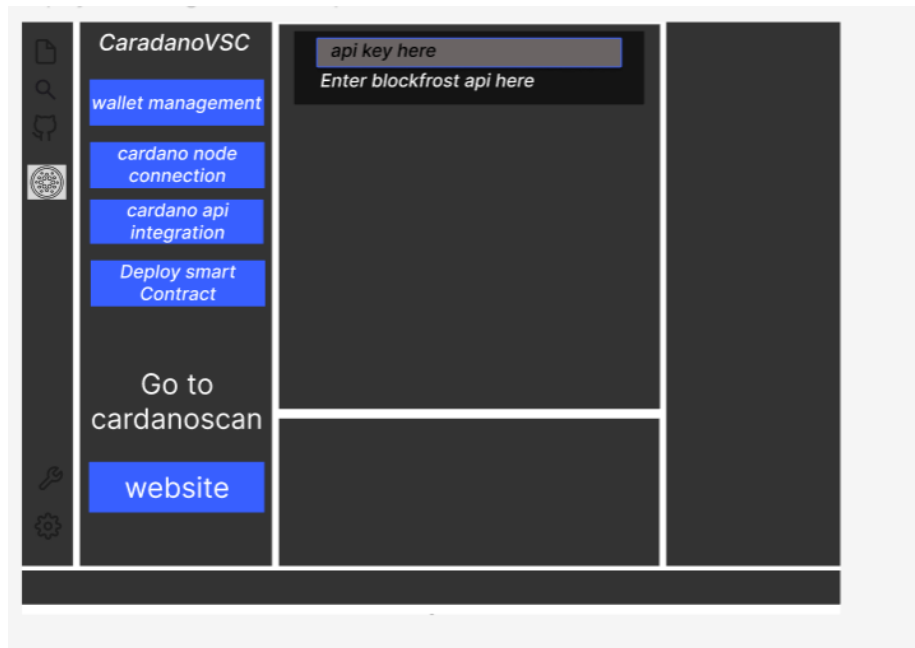


Figure 6: Connecting to blockfrost Node.

- Finally, display the response as a notification.

Smart contract deployment

- User creates a Haskell file and is provided with the boiler plate Plutus code.
- User can Click on a deploy button or execute cabal commands to build a Plutus file.
- Display a Success message and create the Plutus file for the user.
- If user node is connected to a node, then user can deploy the smart contract using Blockfrost directly to blockchain using configured API key stored in VS code local storage state.
- Create a boiler plate code for Plutus. The boiler plate code will be pre-written.
- If user has configured a personal node on their device they can use vscode to run it locally. User then can use terminal to deploy smart contracts as well.

4.5 Wallet Management

- Click sidebar button or execute command for wallet management. This UI component will be available in the vs code sidebar.

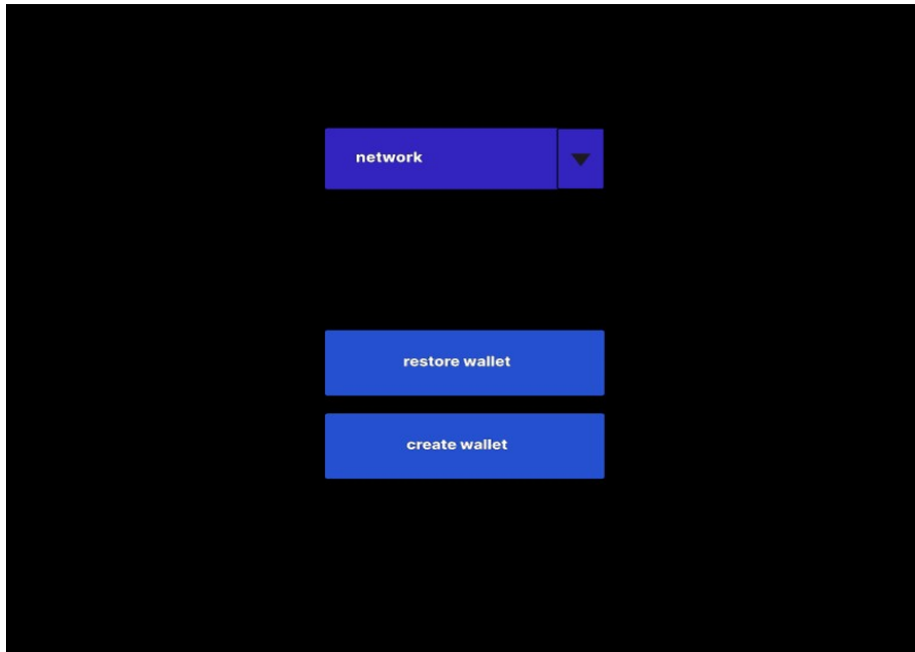


Figure 7: Wallet UI Mockup.

- Allow user to access different networks for wallet **Preview**, **Preprod** and **Mainnet**. Network switching is possible using Blockfrost API. If a person wants to make any network changes they need to change their node network to the network they want (i.e. Preview, Preprod and Mainnet).

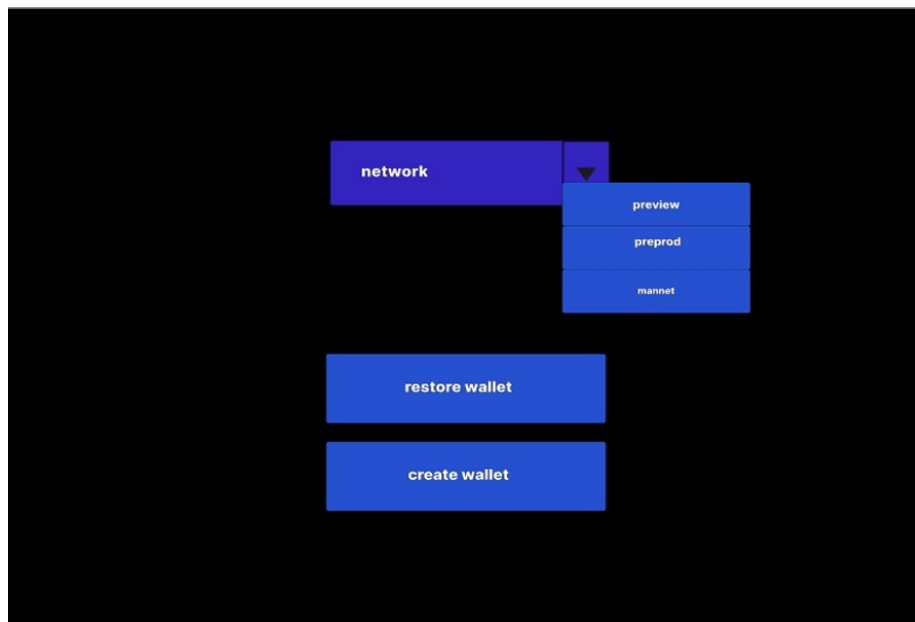


Figure 8: UI Mockup for network selection.

- Create and show the seed phrase to the user the user can choose how to store securely on their own.

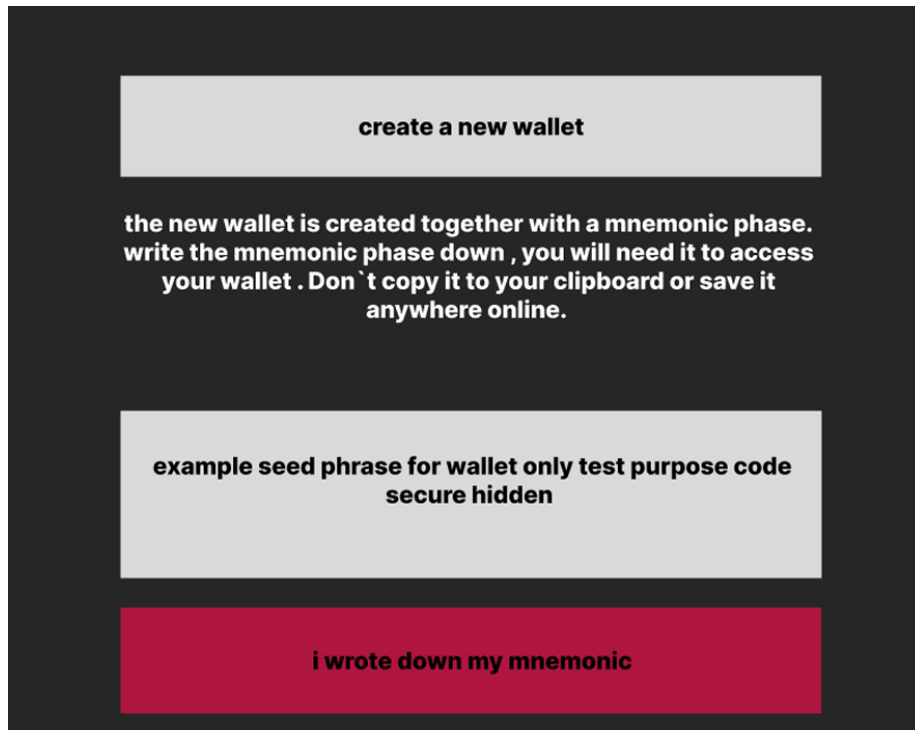


Figure 9: UI Mockup for Wallet seed phrase generation for GUI wallet.

- Alternatively after selecting the Cardano network the user can select to restore wallet by entering their seed phrase.

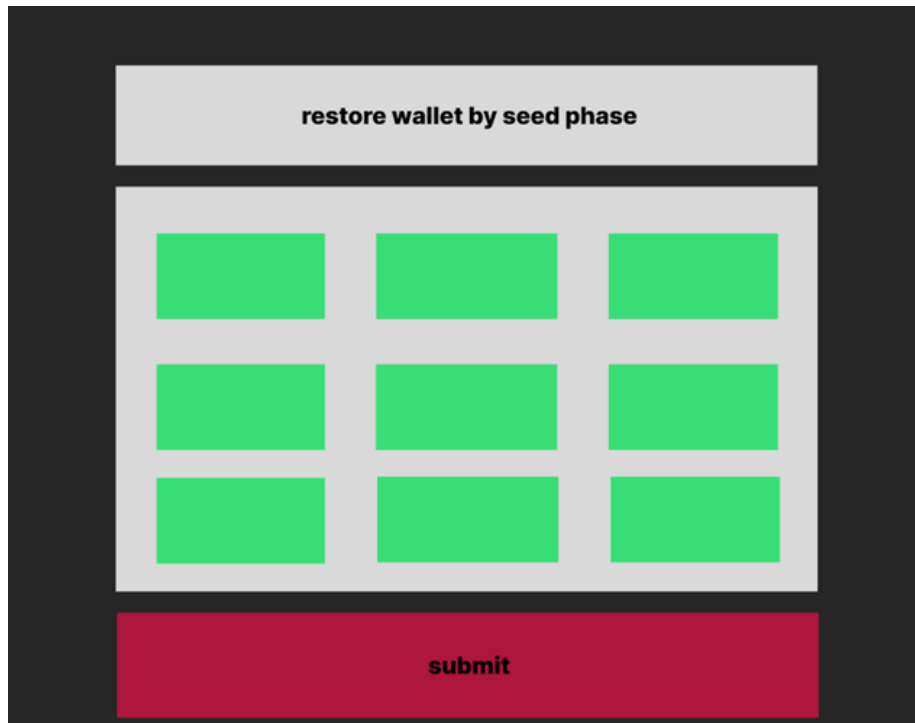


Figure 10: UI Mockup for restoration for GUI wallet using wallet seed.

- Enable wallet functionality such as sending, receiving and checking balance.
- We will position this Wallet in the sidebar of Visual Studio Code. The entire wallet will be bounded inside it, so it is easily accessible to users.

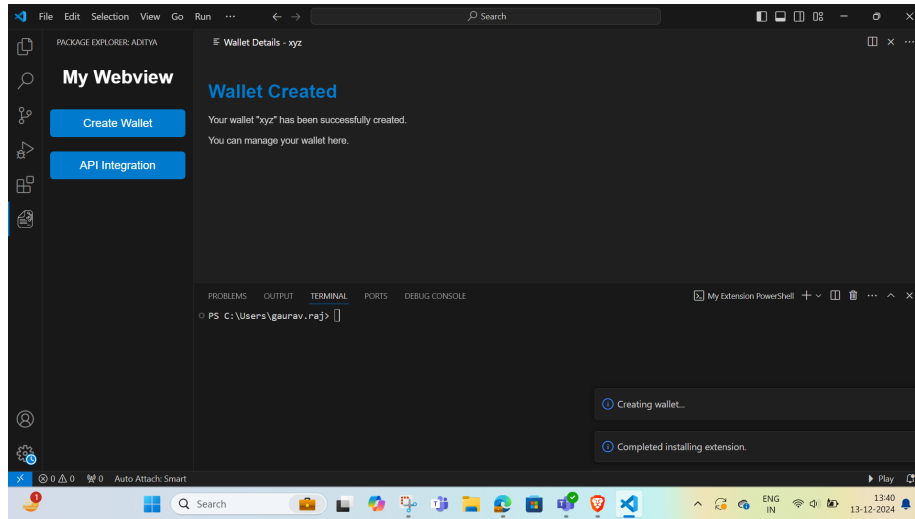


Figure 11: UI Example for a wallet management from vs code.

4.6 Debugging Tools for Smart Contracts and Wallet Transactions

- The CardanoVSC extension will integrate the GHCi interpreter and Haskell debugger, enabling developers to step through Plutus smart contract code. Users will be able to inspect variables and track the flow of execution directly within Visual Studio Code. This will help identify logic errors and optimize contract behavior in a more granular and precise manner.

Workflow in VS code:

1. User imports a smart contract file written in Haskell and Plutus.
2. The user then can start the debugging process by clicking the debug icon in vs code. This starts the debugging process.

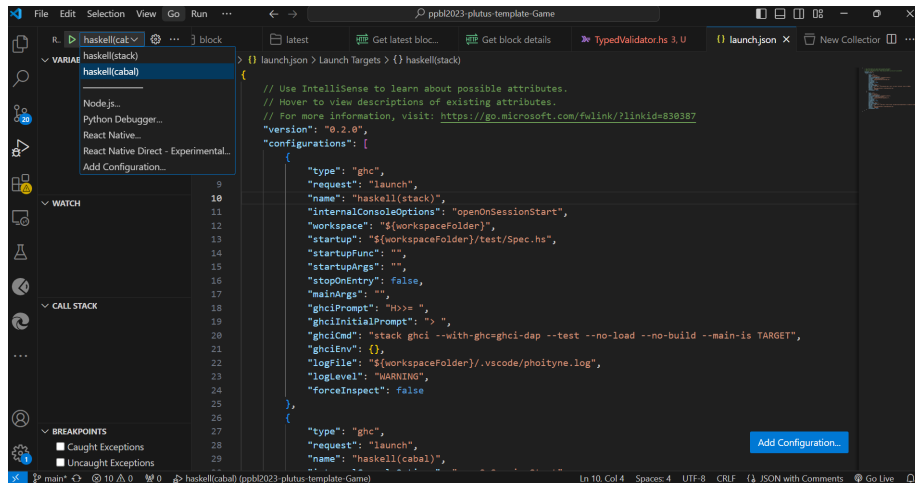


Figure 12: Vs code debugger, here the user can select the debugger they want if they have the extension installed.

3. A new Debug console will open in VS code and the debugger will execute. Here the debugger will go through the file we opened in step 1 and notify us of any errors in any line of the file and display the message in the debug console.

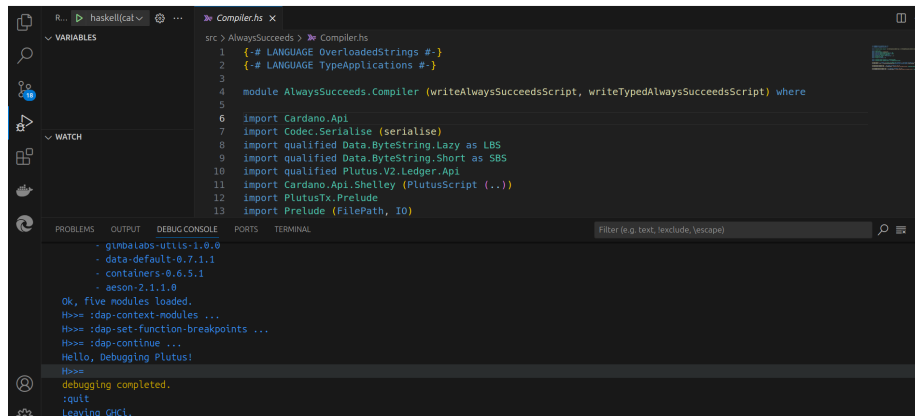


Figure 13: Debug console for vs code.

- To configure this debugger we need to create a launch configurations in launch.json to specify how the debugger should start.
- We also add configuration commands cabal update, cabal build and cabal repl in tasks.json which allows us to execute the required commands in compiling and deployment of Plutus Smart Contracts.

- **Potential Risks:**

- Complex Edge Cases: There is a risk that the system might encounter numerous edge cases, which could complicate the debugging process.
- Debugging Challenges: The system may struggle to effectively debug very complex code.

4.7 Code Examples and Documentation

Include Haskell and Plutus code snippets for:

- Writing smart contracts.
- Creating and submitting transactions.

The examples should be organized and accessible from within the extension.

In the documentation, we will provide a comprehensive guide for:

- Setting up the extension.
- Writing and deploying smart contracts.
- Using wallet management tools.

We will also use links to external resources in the documentation(e.g., Cardano and Plutus documentation). We make the documentation for all the components we are adding to the extension, so if any developer wants to get into details they have full access to the documentation in Github Docs. We will also create a Readme.md file for Step by Step Installation and use, and we will maintain and update the file as the extension gets updated.

5 UI/UX Mockups

Figma design: <https://www.figma.com/design/MiVmXatePUc3UndaG17eGK>

- **Syntax Highlighting:**

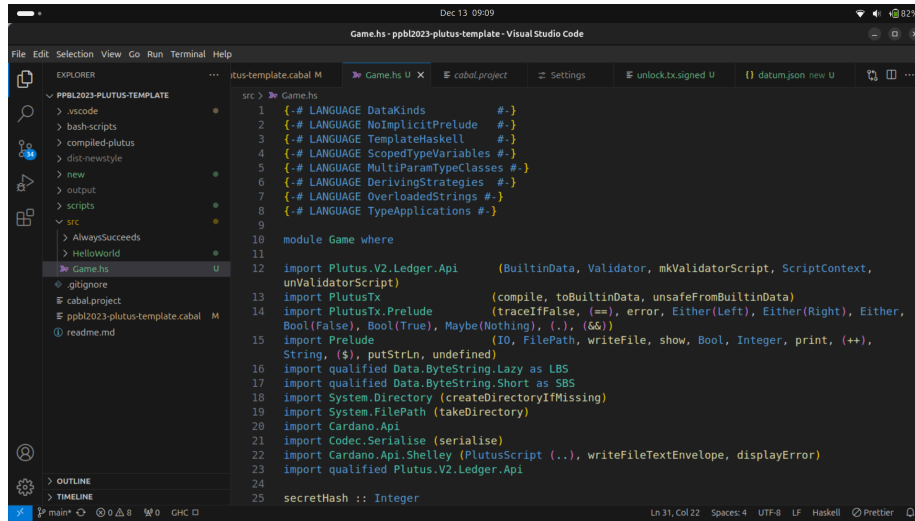


Figure 14: UI for syntax highlighting.

- **Code Completion:**

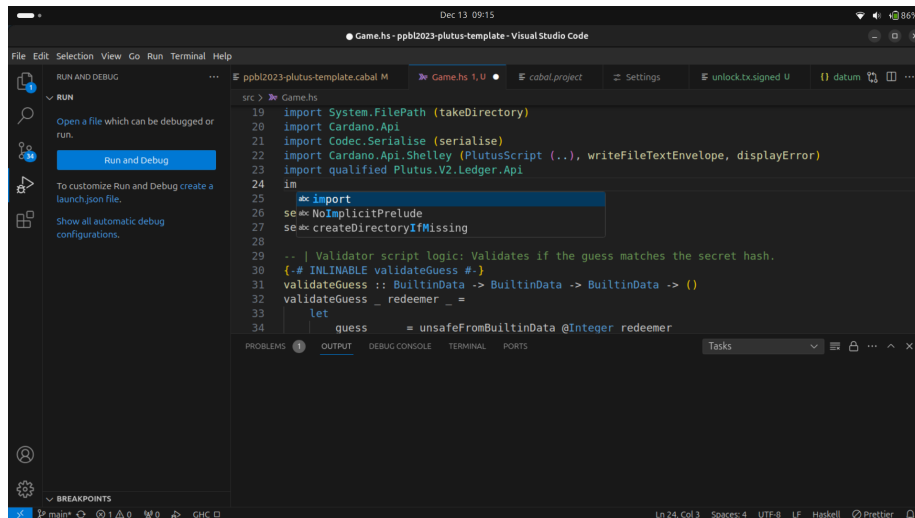


Figure 15: UI for Code Completion.

- **Cardano Blockchain explorer API Data:**

Source code for this can be accessed with this link here:

<https://github.com/AIQUANT-Tech/CardanoVSC/blob/main/DesignDocs>

Inside you will find Postman.CardanoVSC.CardanoscanAPI.json file containing the list of API's. To test this import the json file in postman and enter your cardanoscan API key in the base url.

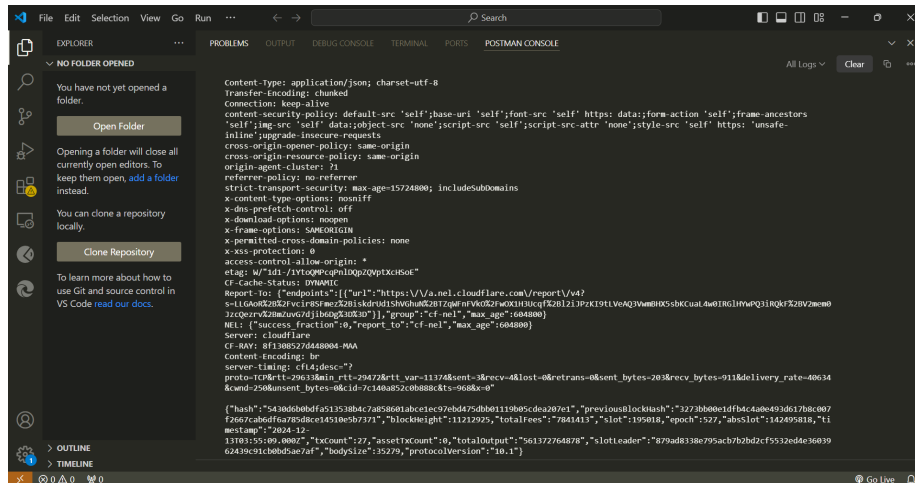


Figure 16: API's for Visual Studio Code Integration with CardanoScan.

- Deployment and Debugging:

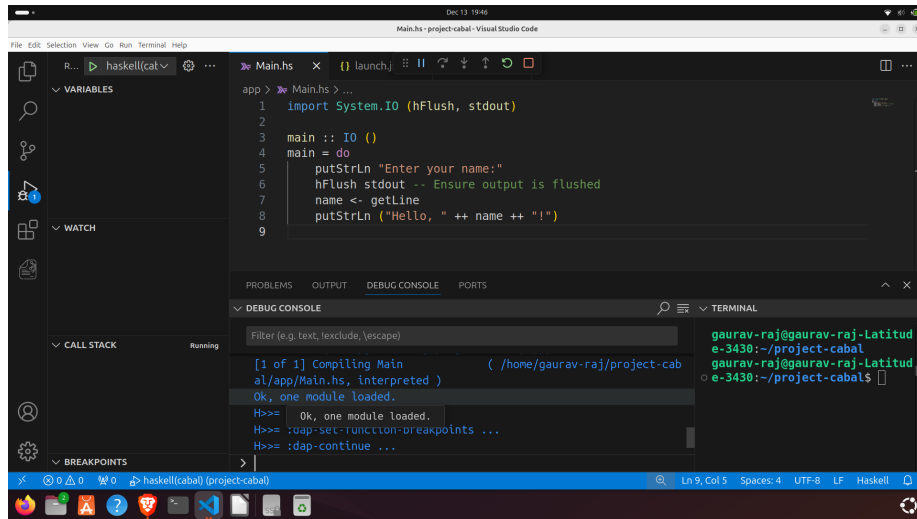


Figure 17: A Visual representation of Plutus debugging and deployment UI in Visual Studio Code.

6 Architecture diagram

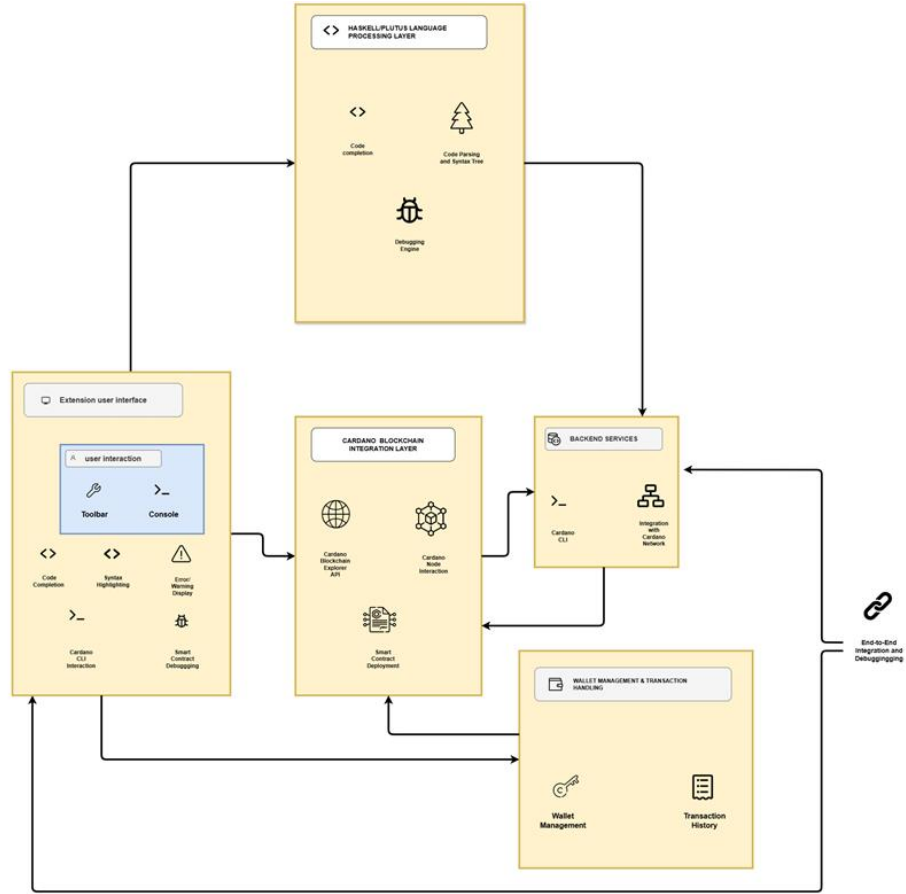


Figure 18: Visual Studio code Architecture.

7 Conclusion

This Plutus and Haskell Extension Integrated within Visual Studio Code IDE (CardanoVSC) aims to enhance the developer experience by providing syntax highlighting and code completion. Throughout this document, we have outlined the design principles, architecture, and implementation details that drive the extension. By adhering to these guidelines, we ensure that the extension remains maintainable, scalable, and user-friendly. Future enhancements should continue to align with these core principles to maintain the integrity and usability of the extension. This scope and design document is fundamental to the development work, which will be carried out in next milestones.