

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	15
4.5	Wallet Management	18
4.6	Debugging Tools	22
4.7	Code Examples and Documentation	25
5	UI/UX Mockups	25
6	Architecture diagram	29
7	Limitation	29
8	Conclusion	30

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, managing and restoring 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

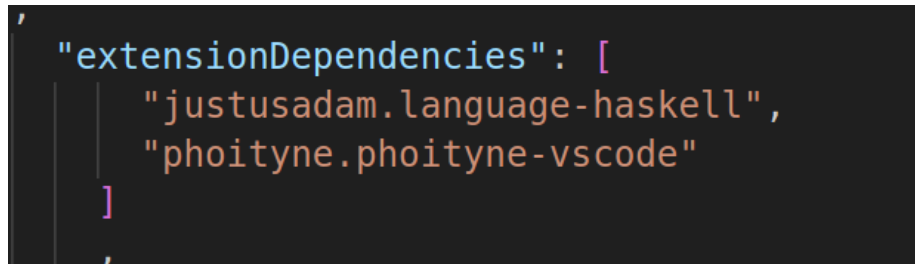
```
npx --package yo --package generator-code -- yo
code
```

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.

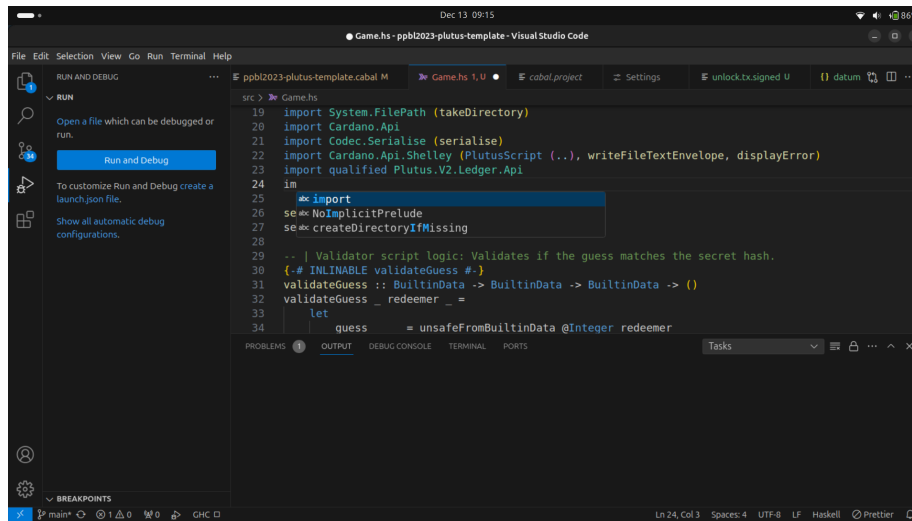


Figure 3: Code completion UI example.

- 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

- **Objective:**
Integrate Cardano Blockchain Explorer API functionality into PyCharm to retrieve essential blockchain data. This integration will allow developers to interact with the Cardano blockchain seamlessly from within the IDE.
- **API Key Requirement:**
A free API key from CardanoScan is required to access their services. Each user must provide their own API key, which can be obtained by signing up on the CardanoScan website.
- **Implementation Plan in VScode:**
Create a configuration panel within VScode to allow users to input and store their CardanoScan API key securely.
Develop backend API integration within VScode plugins or tools to call the CardanoScan endpoints.
Display retrieved data in a structured and developer-friendly manner using VScode's custom terminal.
After configuring api key and cardano network.

1. For running blockchain command first:

- i. To use command Palette press ctrl+shift+p and enter cardanoVSC to get the list of API call commands.
- ii. To use specific extension command ,write "cardanoVSC" in prompt section.

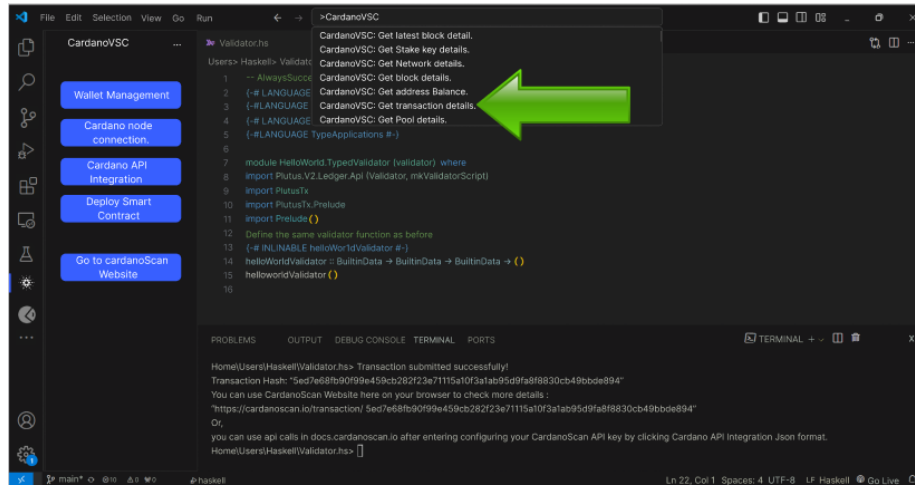


Figure 4: UI for API integration.

- iii. Then select a specific command in given suggestions.
- iv. After selecting that command if it requires any query parameters, a prompt comes for getting user input.

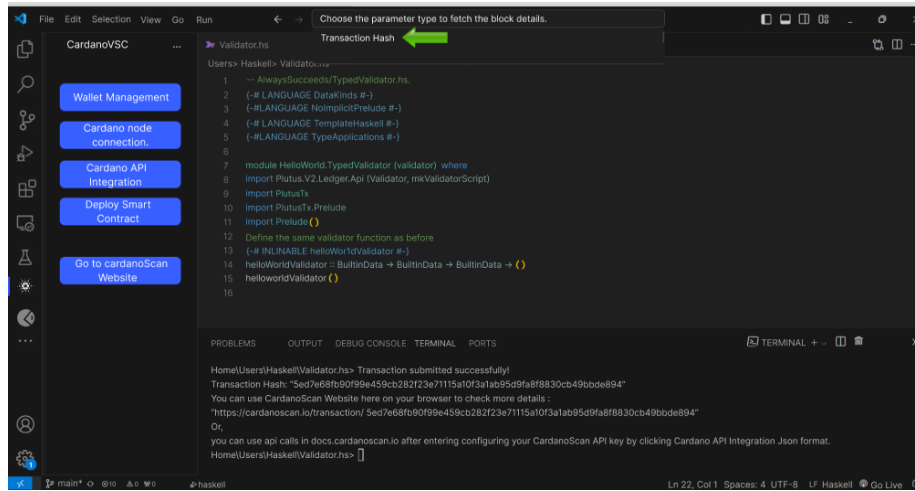


Figure 5: UI for API integration.

v. Finally display result in VSCode output console in json format.

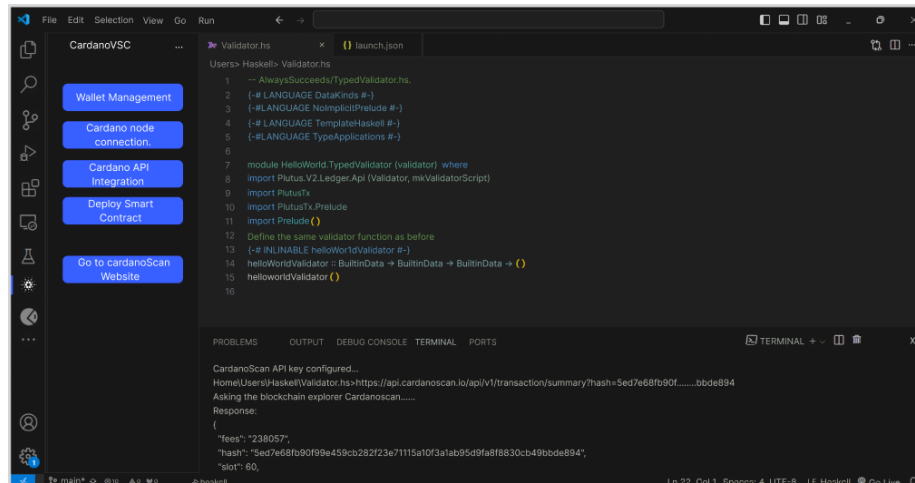


Figure 6: UI for API integration.

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?userhash=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 additional way for people to interact with Cardanoscan. This was **not** in the original requirement, but we thought it would be a nice functionality to add for developers who only need a quick check or are unsure of what cardanoscan url is. If they ever want to use the website we provide them the option to open their recent transaction in their default browser, the Cardano Explorer Website(<https://CardanoScan.io>).
 1. To open cardanoscan website Click the **website** button under Go to cardanoscan.
 2. Then we call default browser to render the CardanoScan.io website.
 3. We have provided the UI wireframing for these in the Figma provided at the start of UI/UX Mockups section. We also have detailed how the user interaction will flow with the layout and details for both methods Cardanoscan API integration, which allows users to use Cardanoscan API queries from the VS code terminal, and Go to CardanoScan website button, which will be aimed for users not wanting to configure a API key, for them we will open their default browser and search the url for their latest transaction only,if the user wants full integration of cardanoscan in vscode they would pick method 1, where he can use API's from within the VS Code IDE.

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.

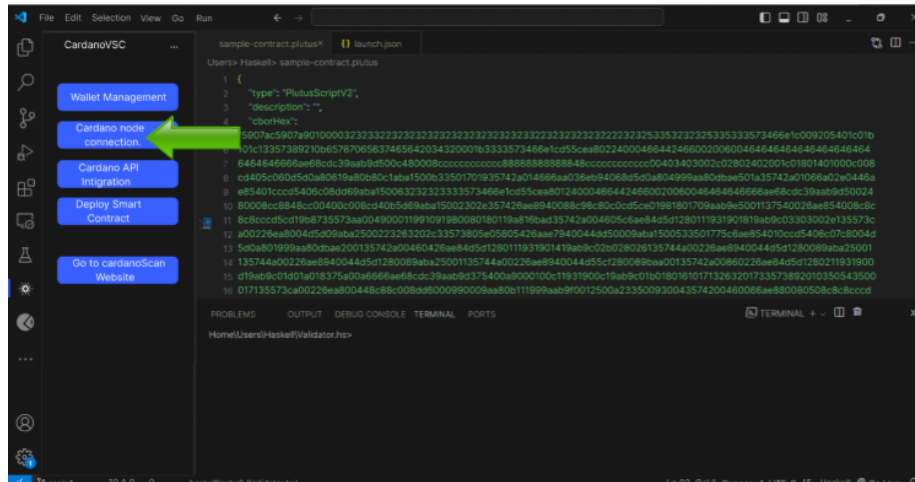


Figure 7: Connecting to a node.

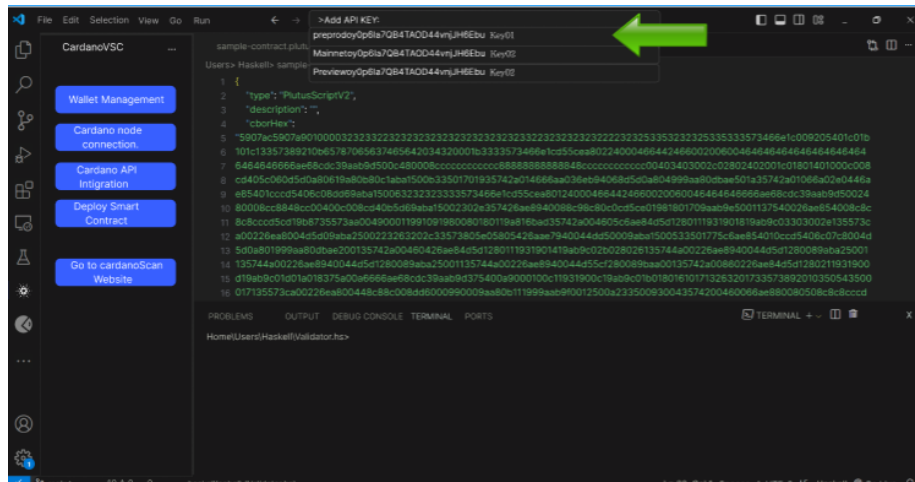


Figure 8: Connecting to a blockfrost Node using blockfrost API key.

This allows us to run:

```

1 #Assuming data is a serialized transaction on the file
  -system.
2 curl "https://cardano-mainnet/preview/preprod.
  blockfrost.io/api/v0/tx/submit" \
3 -X POST \-H "Content-Type: application/cbor" \
4 --data-binary @./data

```

Which allows us to submit transactions with a Plutus application as defined in Content-Type.

Processes: Write your smart contract code using a language supported by Cardano, such as Plutus.

1. Enter your Blockfrost API key. This allows the user to connect to the blockchain node and be ready for deployment.
2. Transaction is created with the api key and the smart contract.
3. Smart contract is then deployed to the blockchain network depending on the API key entered by the user. The blockchain network selection for Preview, Preprod or mainnet is taken from the API key.

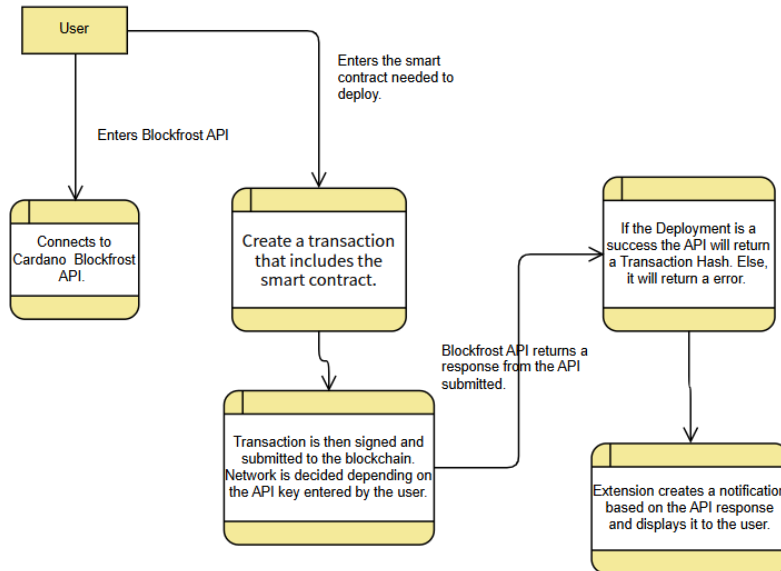


Figure 9: Data Flow diagram for Cardano Smart contract connection and deployment.

Role of API:

1. Intermediary: The Blockfrost API acts as an intermediary between VSCode

IDE and the Cardano blockchain node, facilitating communication and data exchange.

2. Query Handling: The API handles different types of queries, ensuring that the correct data is retrieved and formatted appropriately.

3. Efficiency: The API optimizes data retrieval processes, ensuring quick and accurate responses to user queries. Ease of Access: It is easy to get a API key and deploy smart contracts for users. As opposed to running a node which will take time to sync and then deploy.

Smart contract deployment

- **Select a .plutus File:**
Provide users with an option to select a Plutus file.
- **Deploy Smart Contracts:**
If the user's environment is connected to a Cardano blockfrost API, enable deployment of the smart contract directly to the blockchain.
Use the Cardano blockfrost API with addresses and keys stored in VS-code's local storage state or configuration files.

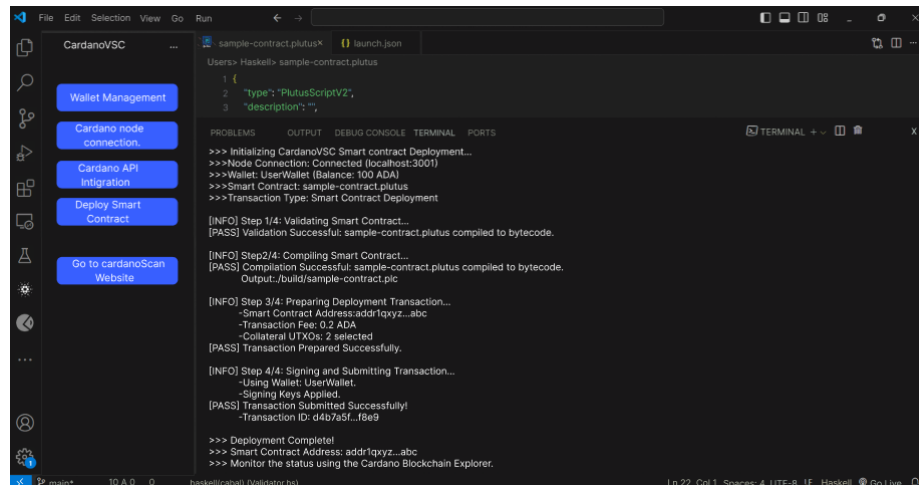


Figure 10: Figma UI Mockup.

- **Error Handling:**
Provide detailed error messages if the build or deployment fails due to issues like Cabal configuration errors or missing dependencies. Suggest potential fixes or guide the user in correcting their setup.
- **Transaction:**
If a transaction is required in the smart contract, user can submit the transaction using a command in VSCode.

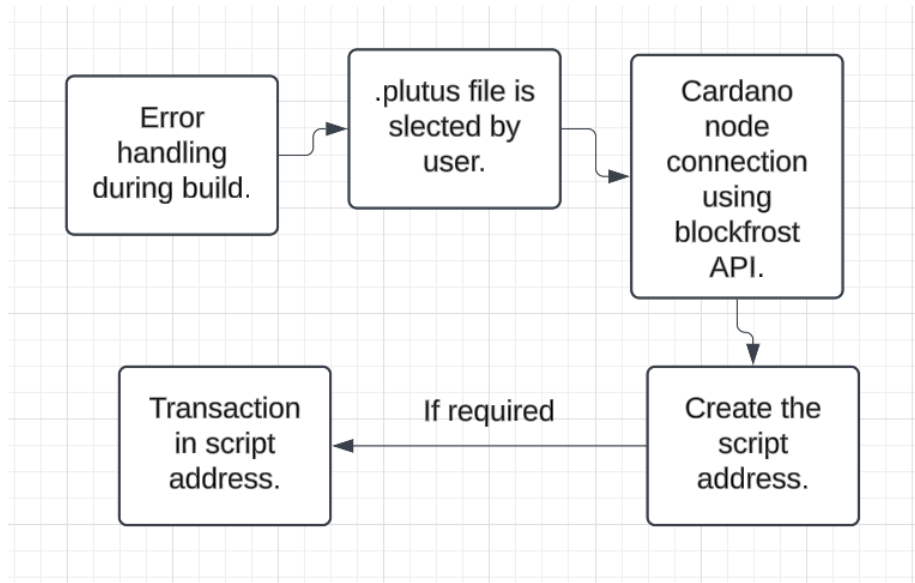


Figure 11: Dataflow of Smart contract deployment.

4.5 Wallet Management

- **Objective:** Provide an intuitive interface within VScode for managing Cardano wallets, enabling users to interact with different networks, create wallets, restore wallets, and perform essential wallet operations like sending, receiving, and checking balances.

Steps

- **Accessing Wallet Management Tools:**

1. Add a dedicated sidebar button or menu command in VScode for wallet management.
2. Clicking this option opens a Wallet Management UI integrated within the VScode interface.

- **On-Chain Connection:**

1. Integrate on-chain connection options using Lucid Cardano, Blockfrost API.
2. Provide step-by-step documentation for the GUI interface to ensure users understand how to configure and connect their wallets to the blockchain.

- **Network Selection:**

1. Allow users to switch between networks (e.g., Preview, Preprod, and Mainnet) using the Wallet Management UI.
 2. For GUI users: Enable network selection directly via dropdown menus using Lucid Cardano.
 3. For CLI users: Document the steps for changing the node configuration file to match the desired network.
- **Seed Phrase Generation:**
 1. Add functionality for users to create new wallets.
 2. Display a 12/24-word seed phrase for secure storage.
 3. Educate users with on-screen tips or tooltips on securely storing their seed phrases.
 - **Restoring Wallets:**
 1. After selecting a network, allow users to restore wallets by entering their seed phrase via the Wallet Management UI.
 2. Validate the entered seed phrase before proceeding to restore the wallet.
 - **Core Wallet Functionalities:**
 1. Sending ADA or tokens.
 2. Receiving funds (displaying a wallet address for a QR code or copy).
 3. Checking balances and transaction history in a tabular or graphical format.
 - Click sidebar button or execute command for wallet management. This UI component will be available in the vs code sidebar.

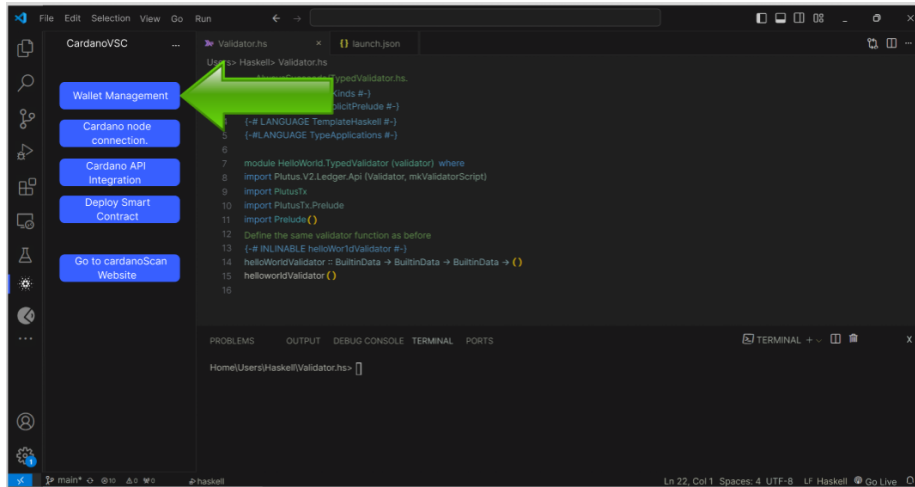


Figure 12: UI Example for a wallet management from vs code.

- 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 the want(i.e. Preview, Preprod and Mainnet).

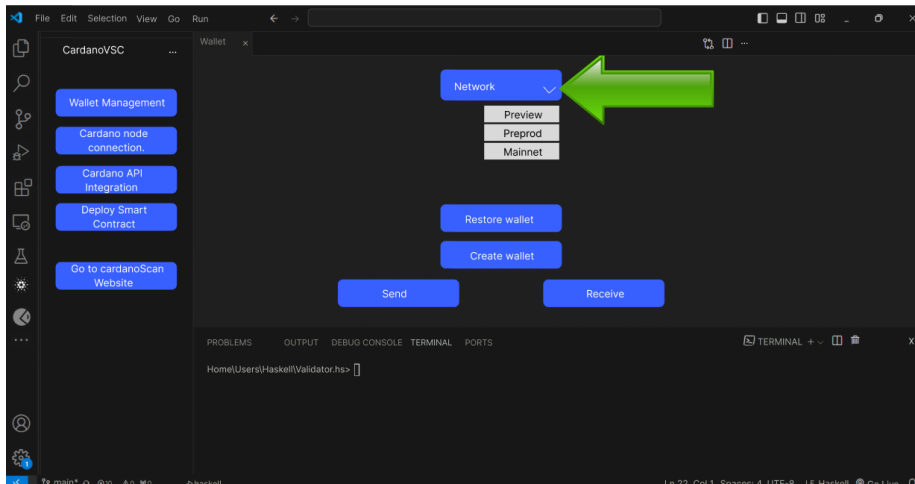


Figure 13: UI Mockup for network selection and Create/Restore Wallet.

- We can then call the notify function in vs code called `vscode.window`. This will then make the extension display the message once during wallet creation.

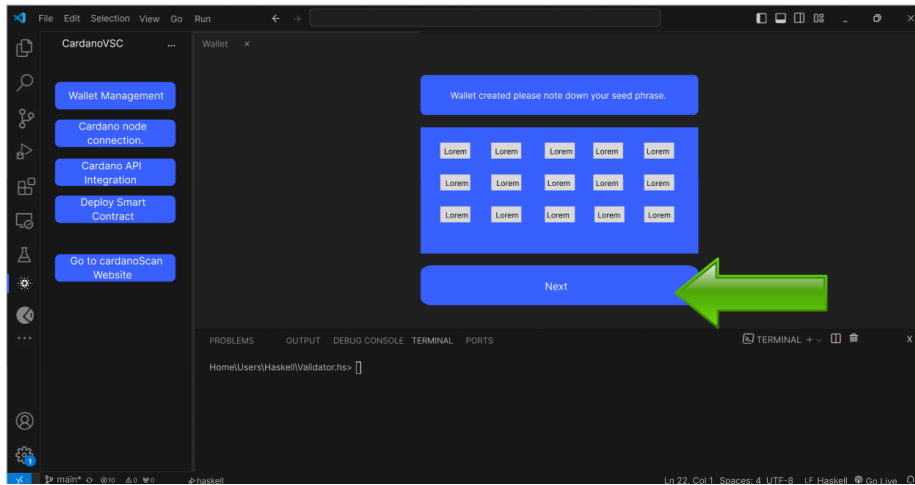


Figure 14: UI Mockup for Wallet seed phrase generation for GUI wallet.

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

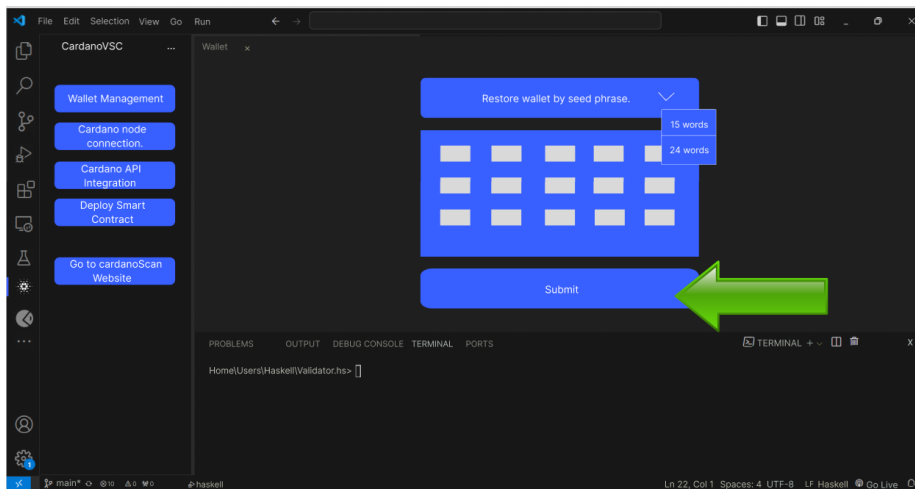


Figure 15: UI Mockup for restoration for GUI wallet using wallet seed.

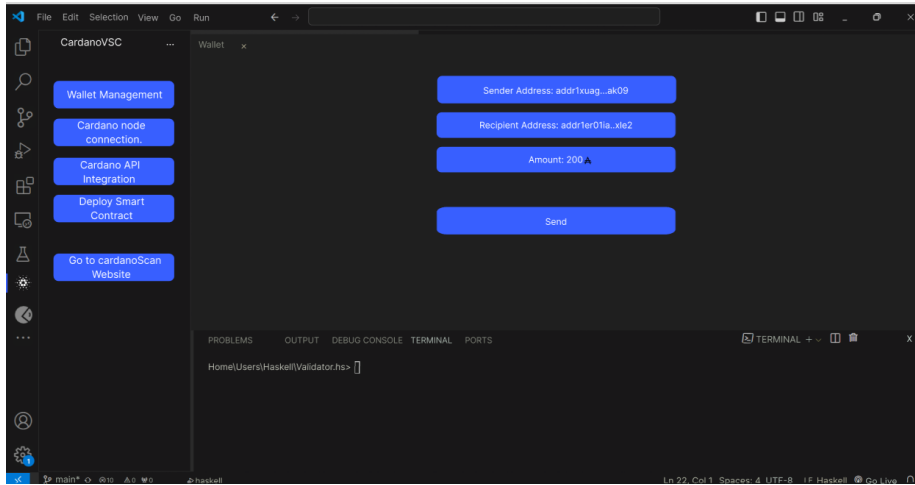


Figure 16: UI Send and receive.

- Enable wallet functionality such as sending, receiving and checking balance. We will position this Wallet inside of the Visual Studio Code IDE.

4.6 Debugging Tools

- 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.

User workflow in VS code:

- **Open Plutus file in VScode:**
 - i. The user opens the Plutus development environment using VScode, which automatically sets up the required tools and libraries.
- **Load and Debug Code:**
 - i. Within VScode, the user loads the Plutus smart contract code.
 - ii. Clicking the Debug button in the CardanoVSC extension starts the debugging session using the a new debugging terminal.
- **Inspect and Refine:**
 - i. The user inspects the code, makes necessary modifications, and re-runs the debugger as needed. We have also provided a UI workflow in the figma.

- **Deliverables:**
 - i. Step-through and interactive debugging.
 - ii. Enhanced Debug Console:
Real-time display of evaluation results, warnings, and errors within VS-code.

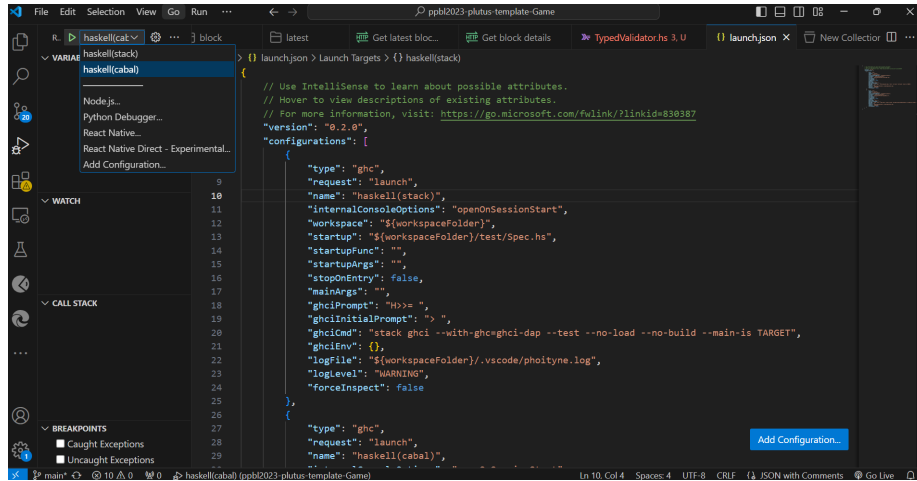


Figure 17: Design for Cardano Haskell and Plutus development debugger `launch.json`.

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.

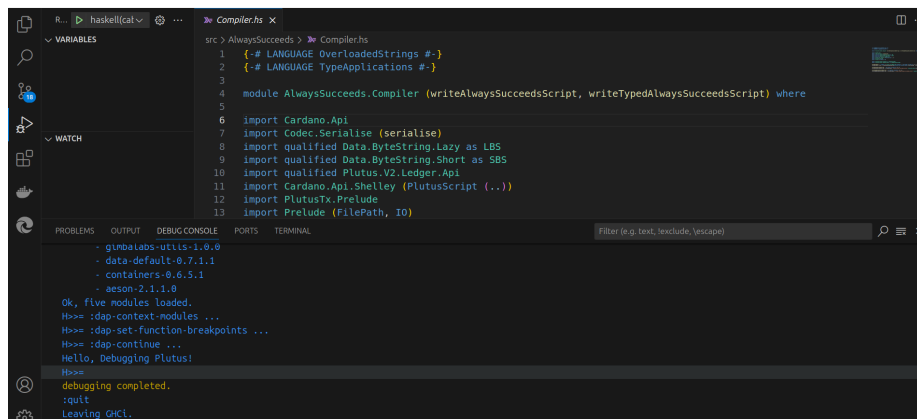


Figure 18: Debug console for vs code. Check the Figma design for more detailed UI workflow.

- 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.

- **Detailed Error Messages:**

Syntax Errors: These occur when the code violates the syntax rules of the programming language.

Message: "Syntax error: unexpected token 'x'" Color: Red

Type Errors: These occur when an operation is performed on incompatible types.

Message: "Type error: expected 'Int' but got 'String'"

Color: Orange

Runtime Errors: These occur during the execution of the program.

Message: "Runtime error: division by zero"

Color: Yellow

Logical Errors: These are harder to detect as they do not produce error messages but result in incorrect output.

Message: No direct message, but incorrect results.

Color: Blue (for highlighting potential logical error areas in debugging mode)

Colors:

error: #FF0000(Red)

warning: #FFA500(Orange)

info: #FFFF00(yellow)

hint: #0000FF(Blue).

- **Potential Risks:**

1. Complex Edge Cases: There is a risk that the system might encounter numerous edge cases, which could complicate the debugging process.

Mitigation: We will implement a robust testing framework that includes unit tests, integration tests, and end-to-end tests. Ensure that tests cover a wide range of scenarios, including edge cases.

2. Debugging Challenges: The system may struggle to effectively debug very complex code.

Mitigation: Break down complex code into smaller, more manageable modules. This makes it easier to isolate and debug specific parts of the system.

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:**

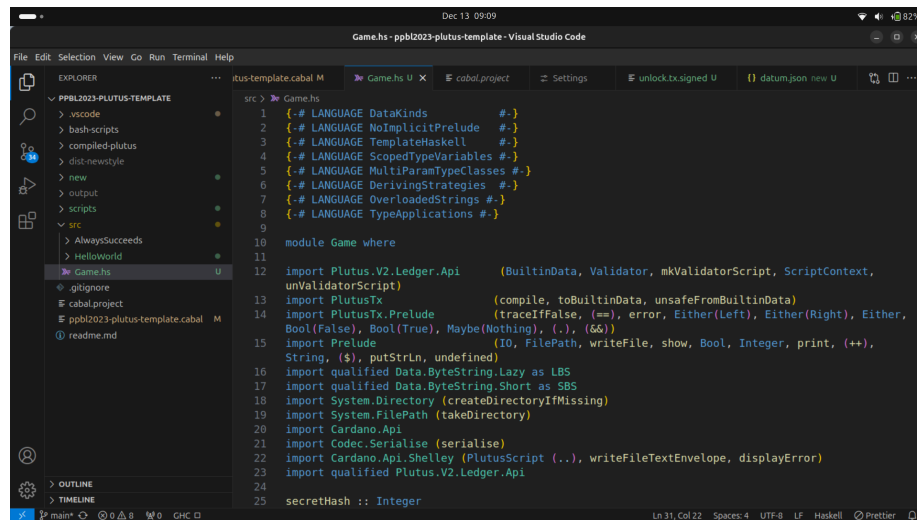


Figure 19: UI for syntax highlighting.

- **Code Completion:**

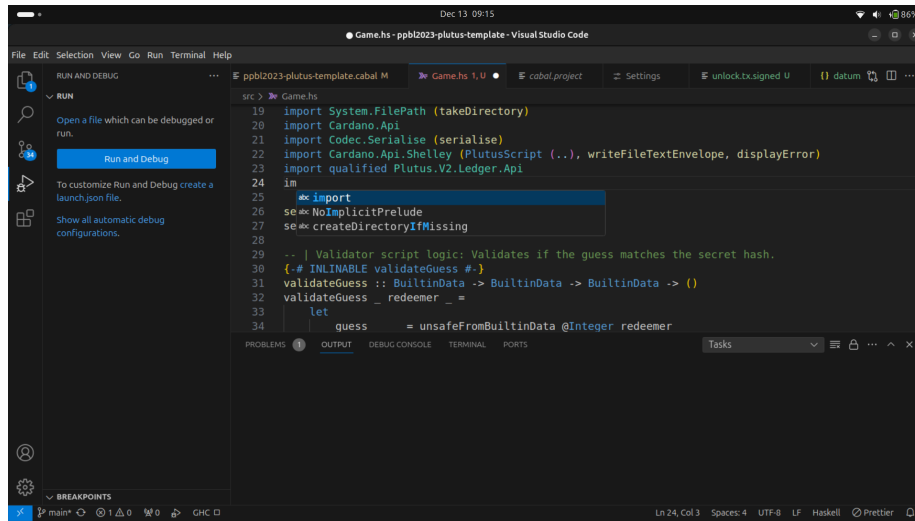


Figure 20: UI for Code Completion.

- **Cardano Blockchain explorer API Integration:**

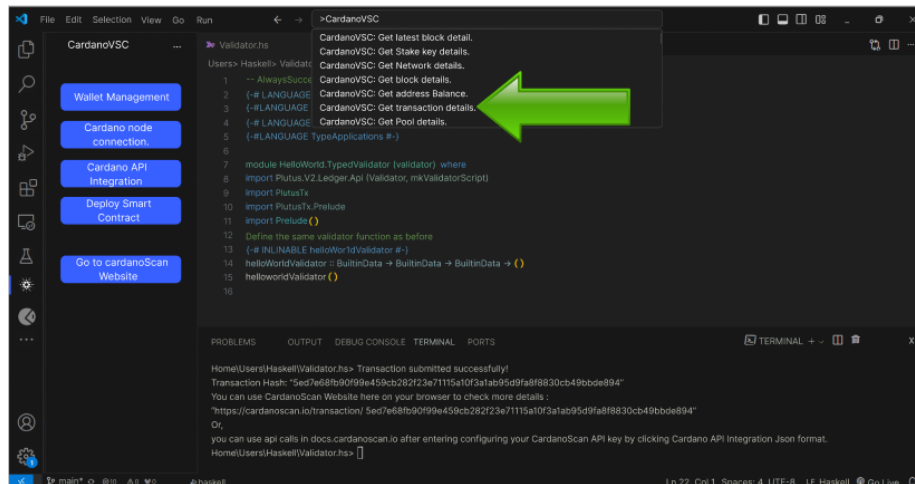


Figure 21: Mockup of API's in Visual Studio Code Integration with CardanoScan.

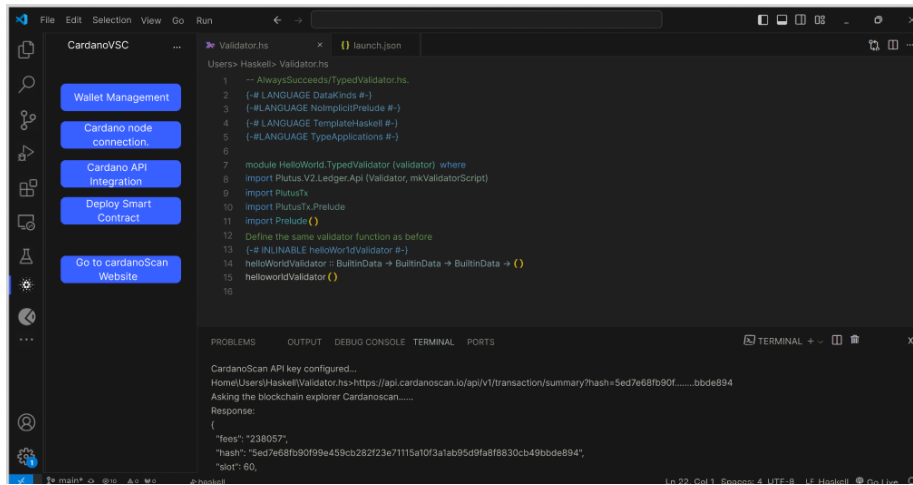


Figure 22: UI Mockup: get Transaction details API response for Visual Studio Code Integration with CardanoScan.

- **Deployment:**

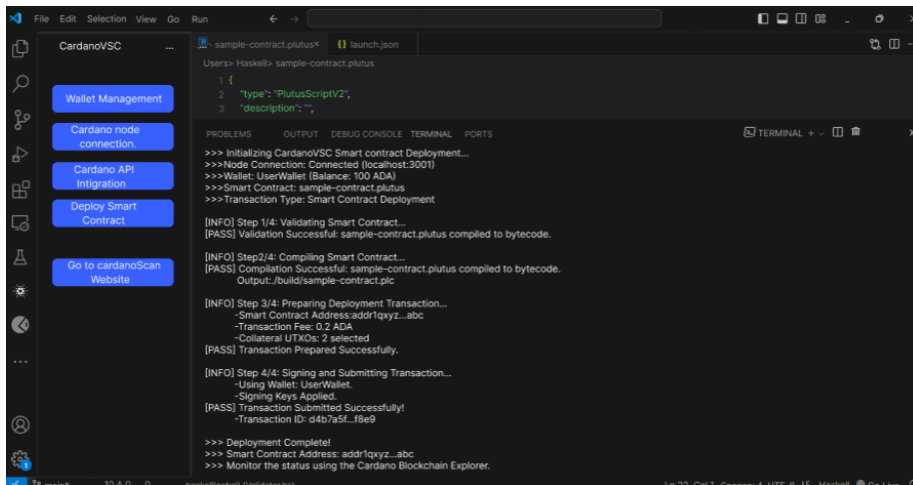


Figure 23: A Visual representation of Plutus deployment UI in Visual Studio Code.

- Debugging:

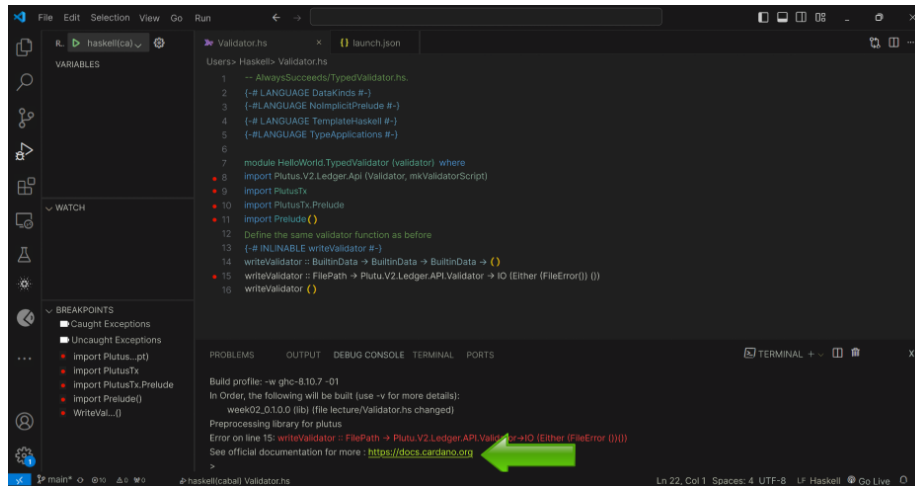


Figure 24: A Visual representation of Plutus debugging with red dots as break-points UI in Visual Studio Code.

6 Architecture diagram

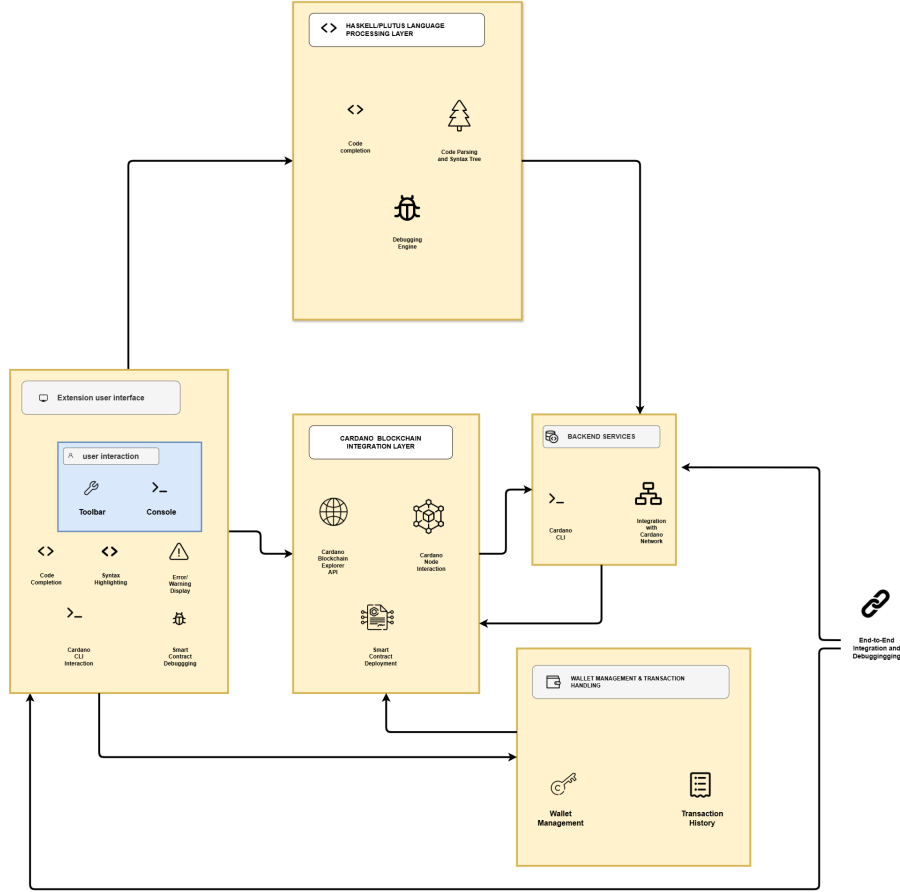


Figure 25: Architecture Diagram for Plutus and Haskell Extension Integrated within Visual Studio Code IDE.

7 Limitation

While the Plutus and Haskell Extension Integrated within Visual Studio Code (CardanoVSC) offers significant enhancements, it is important to acknowledge certain limitations. One notable limitation is compatibility with specific platforms. First, smart contract deployment is dependent on a cardano blockchain node, which is only possible on Linux so to overcome this issue we utilize the most used blockchain node provider blockfrost API to query the cardano blockchain.

Secondly Syntax highlighting is dependent on the vscode theme Color contrast-

ing may not be properly done by user made themes and any user can make and use their own custom theme for vscode, so we cannot provide support for user made themes So to overcome that issue we encourage users to use the default themes provided by vs code.

Third we can say there are numerous ways to set up Cardano node So we can't provide support for all of them during this project so we have picked blockfrost API, However with future updates we can provide support to new methods on different OS systems as required by the community.

Fourth, the debugger might have some uncaught errors that are not clear on first glance to the user so we provide them with error handling.

Finally for addressing any additional issues in future, updates will be crucial to expanding the extension's accessibility and effectiveness.

8 Conclusion

This Plutus and Haskell Extension Integrated within Visual Studio Code IDE (CardanoVSC) aims to enhance the developer experience for Haskell and Plutus by providing syntax highlighting, code completion, Cardano blockchain integration, node connection, smart contract deployment, wallet management, debugging, code examples and documentation. 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.