Open Science Labs

Language Server Protocol for Makim's YAML Configuration

Candidate Info

- **Name:** Ansh Arora
- **GitHub:** ansh808s
- **Email:** ansharora3839@gmail.com
- **Twitter/X:** ansh3939
- **Discord:** ansh3839
- **University Course:** Bachelor of Computer Applications (BCA) (3rd Year)
- **University:** MATS University
- **Time Zone:** GMT+5:30 (India Standard Time)

Bio: My programming journey began with Python just before college, where I built an automation tool to track the price history of a keyboard I wanted to buy which involved web scraping. That small project sparked my curiosity about automation and problem-solving with code.

But curiosity never lets me stay in one place for too long. I ventured into web development, initially building websites with Flask before transitioning to the Node.js ecosystem. Over time, I explored and gained knowledge about building scalable systems.

Currently, I'm diving deep into the Web3 space. At the same time, I'm pushing myself to understand systems at a deeper level, learning Rust to get closer to the metal and improve my understanding of low-level programming.

Project Overview

- **Project:** Makim
- **Project Idea/Plan:** Language Server Protocol for Makim's YAML Configuration
- **Expected Time:** 350 Hours

Abstract

Makim's YAML configuration currently lacks advanced editor support, such as real-time validation, auto-completions, and intelligent suggestions. This proposal aims to introduce Language Server Protocol (LSP) support for Makim's YAML configuration enabling features such as syntax validation, contextual IntelliSense, hover documentation, and error diagnostics directly within the preferred code editor.

The project will involve developing a language server that will leverage Makim's existing schema It will analyze configuration files, enforce schema compliance, and facilitate interaction with IDEs, including Visual Studio Code, Neovim, and JetBrains IDE

A VS Code extension will be developed as the language client. It will include custom commands and actions, such as buttons to execute configuration files or run specific tasks directly from the editor. These additions will provide a significantly improved developer experience, and make Makim more accessible, intuitive, and user-friendly for both new and experienced users.

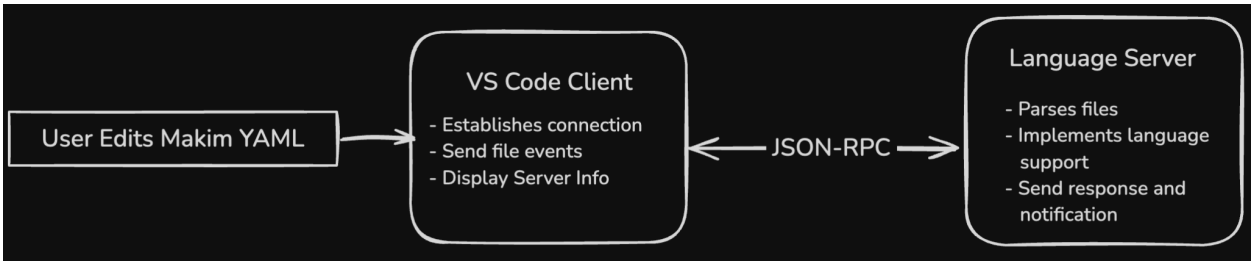
Mentors

Ivan Ogasawara
Abhijeet Saroha

Technical Details

Architecture Overview

The proposed architecture follows the standard LSP model, comprising a Language Client (VS Code extension) and a Language Server, communicating via JSON-RPC.



Language Client (VS Code Extension): It will manage the Language Server's lifecycle, establish communication using the vscode-languageclient library, and act as the interface between VS Code and the server. The extension will also implement custom commands to trigger Makim actions directly from the editor.

Language Server: The server will implement the language intelligence for Makim YAML files. It will parse the files, analyze their syntax and semantics, and provide language features.

JSON-RPC: This protocol will govern communication between the client and the server. Messages including requests, responses, and notifications will be formatted as JSON objects and exchanged with headers specifying content type and length. The transport mechanism will be standard input/output (stdio)

Technology Stack

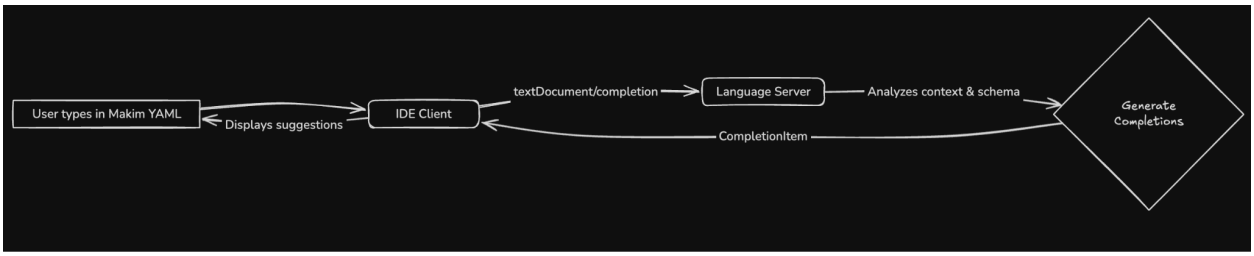
- **Language Client:** TypeScript
  - VS Code Extension API
  - vscode-languageclient library
- **Language Server:** Node.js (TypeScript)
  - vscode-languageserver library
- **Communication:** JSON-RPC over stdio

Feature Implementation Details

Autocompletion

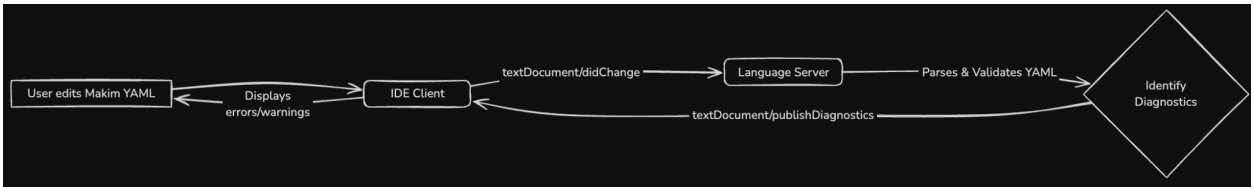
- **Client**: When the user types or triggers autocompletion, the client sends a textDocument/completion request to the server.
- **Server**: The server analyzes the context (e.g., current position in the YAML structure, preceding characters) and provides suggestions for keys like:
    - Top-level keys (e.g., groups, scheduler, vars, etc)
    - Task names and keys within tasks (e.g., help, args, run ,etc)

  The server uses the CompletionItemProvider API to handle the request and sends a CompletionItem with the suggestions.



Syntax Validation

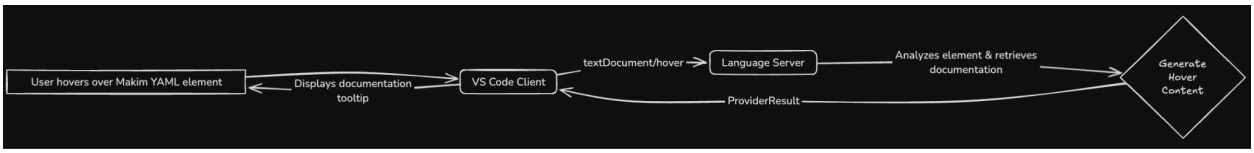
- **Server**: The server performs static analysis to identify errors and warnings. This includes:
    - YAML syntax errors (using a YAML parser)
    - Makim-specific semantic errors (e.g., invalid task names, incorrect argument types, undefined environment variables) The server uses the Diagnostic API to create diagnostic objects and sends textDocument/publishDiagnostics notifications to the client.
- **Client**: The client receives the diagnostic and displays it to the user.



Hover Documentation

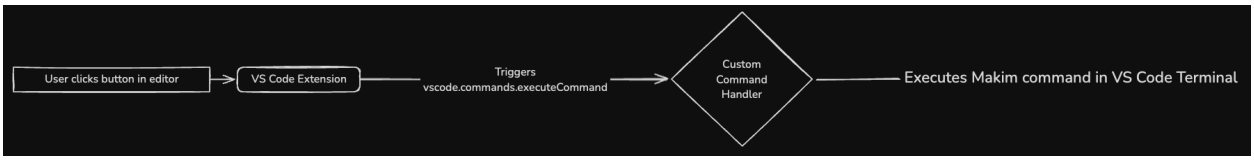
- **Client**: When the user hovers their mouse cursor over a keyword, the client sends a textDocument/hover request to the server.

- **Server**: The server analyzes the element at the cursor position and retrieves relevant documentation. This documentation is based on Makim's schema (e.g., descriptions for properties). The server uses the HoverProvider API to handle the request and sends a ProviderResult containing the documentation content.



Custom Commands and Actions

- **Client (VS Code Extension)**: The VS Code extension will register custom commands using the vscode.commands.registerCommand API. These commands will correspond to actions like executing the Makim configuration file or running specific tasks.



Reference (Generate button runs a terminal command to generate Prisma client):

Testing Strategy

- **Unit tests**
  - Verify individual components of the Language Server.
  - Framework: Jest
- **End-to-end tests**
  - Simulate user workflows in VS Code (typing, triggering autocompletion, error checking, navigation, custom commands).
  - Framework: VS Code's testing APIs

Benefits to the Community

This project will directly benefit users by improving their efficiency. It will make writing Makim configuration files more intuitive, making them more accessible and thus improving the developer experience.

Increased Efficiency: Features like autocompletion, hover documentation, and real-time error diagnosis will speed up the process of writing the configuration.
Interoperability: Developing a language server would help extending LSP support for other editors with minimal efforts
Simplified Workflows: Custom Buttons will allow users to execute files and run individual tasks within the editor.
VS Code Relevance: VS Code is the most widely used editor, so this extension will benefit the majority of users.

Deliverables and Timeline

I will deliver a VS code extension and NPM package for Language Server including all the features mentioned in the technical details section, complete with proper documentation and tests. I will also add biweekly blog posts regarding topics such as the detailed workings of LSP, how to create and publish a VS Code Extension, and core feature implementations, etc. Additionally, I will create a video tutorial explaining all the features of the VS Code extension and how to use LSP with other popular IDEs, along with a tutorial page in Makim's documentation site. During the GSoC period, I will also help resolve bugs in the Makim repository and add new features. If time permits after completing the core objectives, I plan to implement support for other popular IDEs such as Neovim or JetBrains IDEs by developing additional client extensions.

Time frame	Tasks
Community Bonding Period	
May 8 - June 1	• Discuss and finalize the proposed design and features to be added with the mentors. • Set up the new repository with essential configurations, including GitHub Actions, and establish initial project structure.

	• Create a milestone to keep a proper track.
Coding Period Begins (Phase 1)	
June 2 - June 23 (3 weeks)	• Setup the basic Language Server structure. • Setup the basic VS Code client extension. • Implement detecting Makim YAML files. • Implement Syntax Validation for Makim-specific rules. • Setup Jest for unit testing. • Blog Post 1
June 24 - July 14 (3 weeks)	• Implement Autocompletion. • Begin work on Hover documentation. • Write initial unit tests for implemented features. • Setup end to end testing. • Blog Post 2
Phase 1 Evaluation	
July 14 - July 18	• Complete pending work (if any). • Address initial bugs and feedback. • Blog Post 3
July 19 - Aug 2 (2 weeks)	• Complete the work on Hover Documentation. • Implement custom run task UI buttons in the editor • Write more unit tests for implemented features • Start working on documentation site page • Blog Post 4
Aug 2 - 23 Aug (3 weeks)	• Implement any other VS code specific UI features if needed or discussed with the mentor. • Write comprehensive end-to-end tests. • Add more unit tests for features implemented. • Create a video tutorial. • Finalize all code, tests and documentation. • Package VS code extension and Language Server as NPM package. • Blog Post 5
Phase 2 Evaluation	
Till 1 Sep	• Complete pending work (if any) • Address final feedback and bugs. • Prepare final submission. • Blog Post 6

Previous Contributions to the Project

I have actively contributed to Makim during the past month. My contributions include **12 merged pull requests**, focusing on areas such as implementing core features, fixing failing test cases. A notable contribution involved adding an option for tasks to specify a log file, log level and log format with tests cases (#162). I am familiar with the project's codebase and contribution workflow.

Pull Requests

Pull Request	Status
feat: Implement failure hook to run tasks after a task fails (#191)	**Merged**
test(retry): Add smoke tests and update documentation for retry mechanism for tasks (#187)	**Merged**
feat(core): Add conditionals to validate execution of tasks (#183)	**Merged**
docs(spec): Update documentation to include information about log configuration (#177)	**Merged**
feat: Implement task retry mechanism (#175)	**Merged**
feat(ci): Add docker on macos github action (#170)	**Closed**
fix(core): Enforce explicit group naming for tasks (#169)	**Merged**
docs: Update python notebook to showcase latest Makim release (#167)	**Merged**
docs: Update hooks documentation to include info about --skip-hooks flag (#163)	**Merged**
feat: Add an option for tasks to specify log file, log format and log level (#162)	**Merged**

feat: Implement --skip-hooks flag to bypass hook execution (#159)	**Merged**
tests: Fix multiple test failure due to /tmp symlink on macOS (#157)	**Merged**
fix(test): Schedulers cannot be serialized resulting to scheduler test failure (#149)	**Merged**

Why this Project?

My first encounter with a task automation tool was during my previous internship, where I utilized Makefile. I found its syntax to be quite counter-intuitive. In contrast, I was very impressed with Makim's approach to defining the configuration file. Its syntax is intuitive and thoughtfully designed for user experience.

I am eager to contribute to this project because I believe that my past experience with task automation tools have given me valuable insights into the importance of user-friendly design and intuitive syntax. I am excited by the potential of Makim to streamline workflows and improve productivity.

My Node.js environment experience, along with my understanding of Makim's codebase and LSP, gives me confidence in my ability to significantly contribute to Makim. I am confident that I can implement the proposed idea, which would encourage wider adoption by making Makim more developer-friendly.

Availability

With no other obligations this summer, GSoC takes precedence as my sole focus.I am flexible and have no problem adjusting to my mentor's time zone if needed. I am going to commit full time to this project and am ready to devote around 40 hours per week. And in case any delay occurs, I would first assess the situation and determine the reason for the delay. Then, I would communicate with the mentor and dedicate additional hours to ensure that the requirements are met.

I am 100% dedicated to OSL and have no plans whatsoever to submit a proposal to any other organization.

Post-GSoC

After my completion of GSoC, I would love to integrate similar LSP support for the Sugar project since it also uses YAML for its configuration. Additionally I would love to maintain Makim and continue learning from Ivan and other contributors and hopefully mentor for the GSoC 2026.