Multi-Language IDE Implemented in JS

Scope and Architecture

Overview

This document details the scope of the IDE framework implemented in JavaScript, HTML and CSS. It also details the architecture choices that will guide the development of the software.

Scope

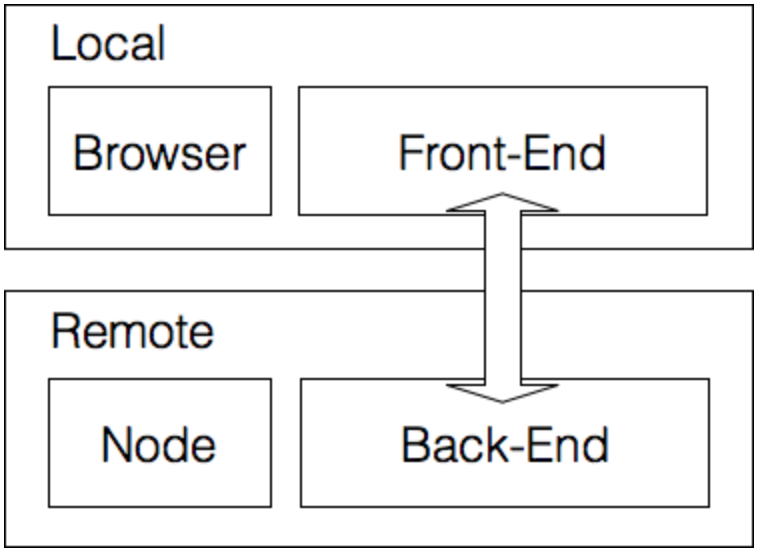
The Javascript IDE project has the mandate to:

- Provide the end-user with a full-fledged multi-language IDE (not just a smart editor)
- Support equally the paradigm of Cloud IDE and Desktop IDE
- Provide extenders with a platform on which to build their own IDE-like products
- Provide support for multiple languages via the language and debug server protocols
- Provide modern GUI with javascript UI libraries
- Decouples frontend plugins from backend services through JSON RPC protocols and/or REST APIs.
- Provide a maintainable code base, with statically typed APIs and clear API life-cycles.
- Use state-of-the-art technology (e.g. TypeScript 2.2)

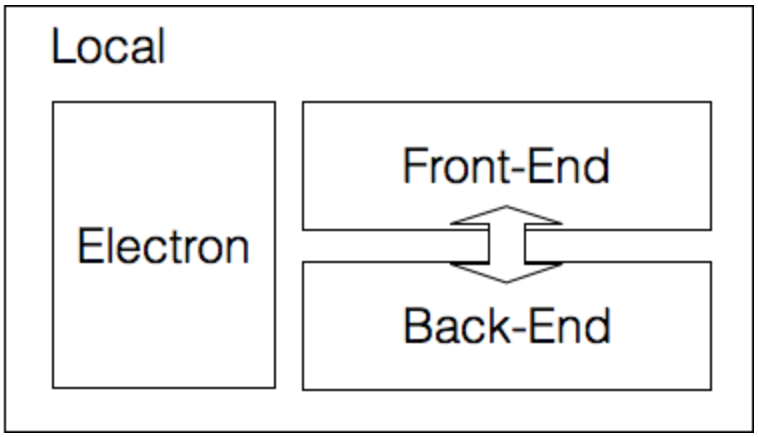
Supported Architectures

To support both native desktop IDEs as well as cloud based IDEs, the basic framework needs to be separated into a front-end part and a back-end part. We see three kinds of deployments we want to support.

1) Web Client, Remote Back-end (Cloud IDE)
   The front-end is served from a remote server to a local browser, connecting to a remote back-end.

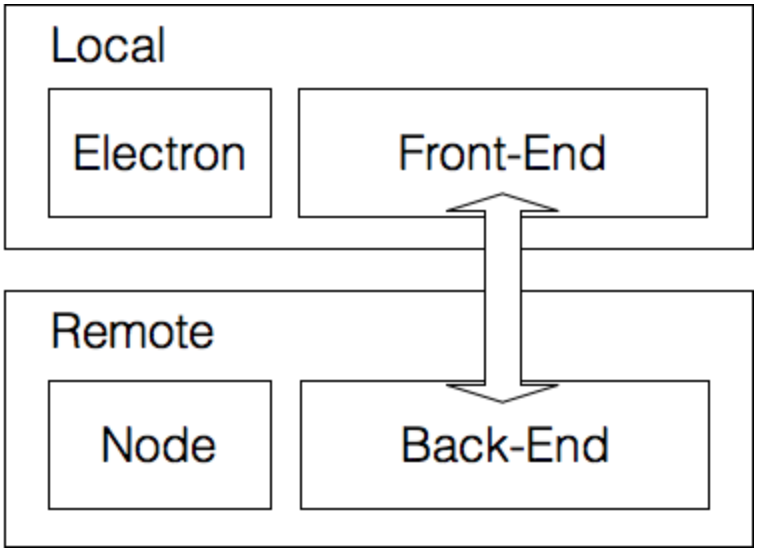


2) Native Front-End, Local Back-end
   Based on Electron an IDE would run the front-end as well as the back-end locally.



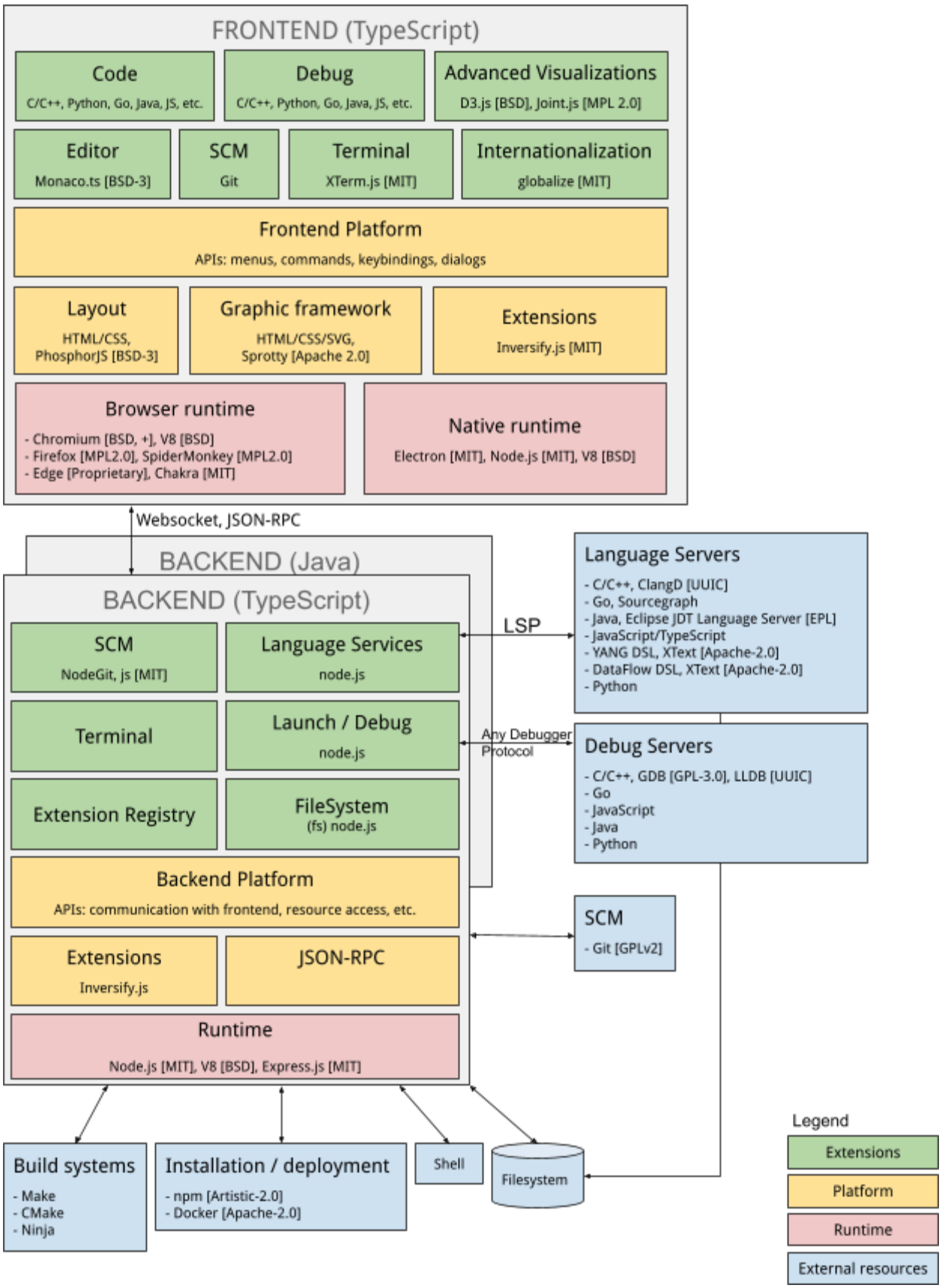
3) Native Front-End, Remote Back-end
   Based on Electron only the front-end would run locally, connecting to a remote

back-end.



Architecture Diagram

The below diagram illustrates the main components and how they are connected. The list of dependencies can be found in in appendix.


FRONTEND (TypeScript)
Code
C/C++, Python, Go, Java, JS, etc.
Debug
C/C++, Python, Go, Java, JS, etc.
Advanced Visualizations
D3.js [BSD], Joint.js [MPL 2.0]
Editor
Monaco.ts [BSD-3]
SCM
Git
Terminal
XTerm.js [MIT]
Internationalization
globalize [MIT]
Frontend Platform
APIs: menus, commands, keybindings, dialogs
Layout
HTML/CSS,
PhosphorJS [BSD-3]
Graphic framework
HTML/CSS/SVG,
Sprotty [Apache 2.0]
Extensions
Inversify.js [MIT]
Browser runtime
- Chromium [BSD, +], V8 [BSD]
- Firefox [MPL2.0], SpiderMonkey [MPL2.0]
- Edge [Proprietary], Chakra [MIT]
Native runtime
Electron [MIT], Node.js [MIT], V8 [BSD]
Websocket, JSON-RPC
BACKEND (Java)
BACKEND (TypeScript)
SCM
NodeGit, js [MIT]
Language Services
node.js
LSP
Language Servers
- C/C++, ClangD [UUIC]
- Go, Sourcegraph
- Java, Eclipse JDT Language Server [EPL]
- JavaScript/TypeScript
- YANG DSL, XText [Apache-2.0]
- DataFlow DSL, XText [Apache-2.0]
- Python
Terminal
Launch / Debug
node.js
Any Debugger
Protocol
Debug Servers
- C/C++, GDB [GPL-3.0], LLDB [UUIC]
- Go
- JavaScript
- Java
- Python
Extension Registry
FileSystem
(fs) node.js
Backend Platform
APIs: communication with frontend, resource access, etc.
SCM
- Git [GPLv2]
Extensions
Inversify.js
JSON-RPC
Runtime
Node.js [MIT], V8 [BSD], Express.js [MIT]
Build systems
- Make
- CMake
- Ninja
Installation / deployment
- npm [Artistic-2.0]
- Docker [Apache-2.0]
Shell
Filesystem
Legend
Extensions
Platform
Runtime
External resources

Single User

For all three scenarios, we assume a single user scenario. That is, if the back-end runs remotely it should run in a sandboxed environment (e.g. docker container), with a dedicated file system. It should be possible to have more than one client connected to the backend, but it will serve the same file system. So for multiple workspaces an additional server (out of scope for this effort, but e.g. provided by Eclipse Che) would be responsible to start / stop such container based remote workspaces.

TODO: Authentication

TODO: Explain how multi-user would work (i.e. workspace servers)

Communication

Communication between frontend components and backend components should be done through well defined JSON-RPC protocols and/or REST APIs. This allows not only to change the actual implementations if needed, but also to distribute the backend into multiple small backend agents running implemented in different programming languages and running independently from each other maybe even on different systems.

That said at the core we will target one backend running in Node.js.

https://wiki.eclipse.org/Orion/Server_API

Frontend

The frontend application should run as a single page application, that can be hosted in modern browsers [1] and in an Electron BrowserWindow (Chromium). The Electron version should leverage Electron specific APIs to integrate with the native desktop as much as possible. For that matter certain services will be implemented specifically for browsers and for electron. Menus, for instance, are supported natively by electron. When running in the browser the framework needs to render it using HTML.

The right implementation will be picked up by configuration (dependency-injection).

Dependency Injection

We believe that a loosely coupled architecture is key to allow integrating the many different components that already exist or may be developed in future. At the same time we want to enable everyone to use the framework as a basis for their IDE-like products and with that fine grained control over how the entire application is configured should be provided.

Dependency Injection delivers on these needs and in addition increases testability of the code base.

Language support

The Language Server Protocol is a widely used JSON-RPC protocol for providing editing services for languages. Our Framework shall fully support this protocol and in fact do any advanced language services mainly through this protocol. Therefore, the editor component we use should support the LSP. It should be possible and relatively easy to extend the protocol with addition language-specific messages.

Client side configuration for things like lexical coloring and bracket matching should be supported.

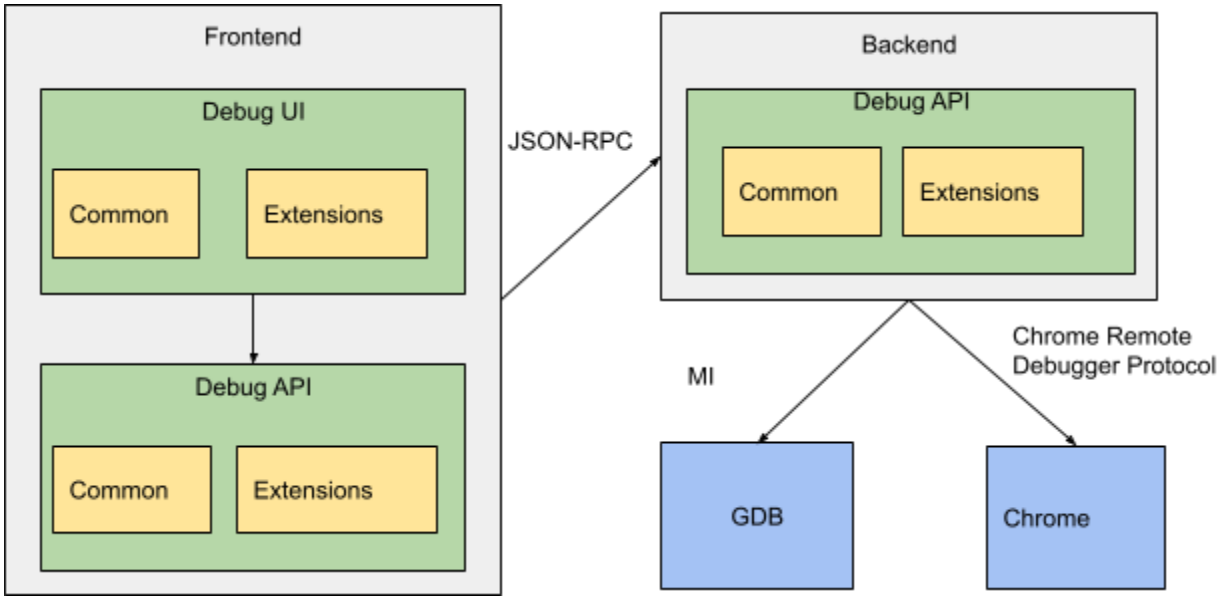
Theia extensions will be able to add logic to decide when to launch a specific Language Server. For example, when opening a .cpp file, it could spawn the clangd process on the backend in order to support C/C++ or when opening a .java file, it could spawn the Eclipse JDT Language server. Extensions could also expose user preferences that will be sent to language servers in order to configure them in a certain way.

Debugging support

The debug component in Theia enables any debugger to be integrated with Theia. Theia offers an extendable framework for debugger UI and external debugger protocol integration.

For example one can integrate GDB or Chrome Debugger inside Theia and be able to extend common debug UI components for each debugger and support the MI protocol for GDB and the Chrome Remote Debugger Protocol for Chrome.

The high level architecture behind this is defined as such:



Extendability

A frontend application as well as the main backend application consists of multiple extensions. An npm package can expose one or more extensions, that can contribute the to the frontend and main backend application.

Such contributions are

- Service hooks for other extensions
- Service implementations of other extension's service hooks
- Singleton Services
- Resources (e.g. CSS) (frontend application)

Npm packages containing Theia extension are published and consumed through regular npm registries.

Provided a consistent set of extensions a Theia application can automatically be generated. The generator can be used at build-time to pre-package Theia applications as well as at runtime through the dynamic extension system.

The dynamic extension system provides a user interface to search for available extensions and change the set of active extensions at runtime. The extension system will regenerate the frontend and backend application and restart them after the user applied changes to the set of active extensions.

Preferences

A Preferences extension should allow users to override default preferences on user and project level. Preferences are stored in a readable textual format (YAML or JSON).

The preferences should be accessible by other frontend components.

Layout

A layout system shall allow other frontend components to contribute UI widgets. Any UI widget based on HTML and CSS should work. The layout should be dock-layout like and allow for splitting views and laying them out using drag and drop.

Command Palette

A command palette should allow to discover and access all available commands. A central command registry allows to contribute commands.

Menu

A global menu service, allows to configure, register and contribute to the main menu and the different context menus. The actual menus should be rendered natively in Electron and using HTML when running in a standard browser.

Menus can be enabled and disabled based on contexts.

A context is a globally registered, named predicate.

Keybindings

A global keybinding service allows to register keybindings which are accelerators mapped to commands. They can be enabled and disabled based on contexts.

File System

Access to a workspace is a central requirement for any IDE-like application. A backend service shall allow accessing a workspace through a well defined JSON-RPC protocol. URIs shall be used to identify resources.

Navigator

The navigator is a UI widget that allows the user to interact with the workspace. It should represent the workspace in a tree. It registers a context menu to which common file commands (copy, paste, delete, create, open) shall be contributed.

Search

It should be possible to search for textual occurrences in the workspace. Regex as well as glob patterns shall be supported. Replace should be supported, too.

Terminal

A terminal extension shall provide the user with a terminal widget that is connected to the shell of the backend.

Long Running Task Support

It should be possible to start longer running tasks on the backend and frontend that can be monitored (progress and cancellation) from the frontend.

Git Support

Support for Git should make it easy to stage and prepare commits, as well as reviewing the history of changes.

Internationalization

A central component for internationalizing labels and other textual information provided to the user.

Builds

A Theia-based IDE shall be able to trigger a build, of the code in the workspace, reporting to the user any errors/warnings that the build tools might output. Upon successful build, we should obtain an executable or application.

A builds extension shall allow the user to define, edit and execute build commands. A build command will be configured to call a build script on the backend, with arbitrary CLI parameters. It shall have a console log parser associated to it, that will identify and report warnings and errors, from the build's terminal or process output.

It shall be possible for extenders to provide their own "console log parsers", to customize the way errors are parsed for currently supported languages or to to support building for new languages.

Appendix

Dependencies

Theia has first-level dependencies on the following node modules (excluding development dependencies):

Module name	version	License	Src repo
electron	1.6.2	MIT	github
express	4.15.2	MIT	github
inversify	3.1.0	MIT	github
monaco-editor-core	0.8.2	MIT	github
monaco-editor	0.8.3	MIT	github
monaco-languageclient	0.0.1-alpha.2	MIT	github
ws (WebSocket)	2.2.0	MIT	github
reconnecting-websocket	3.0.3	MIT	github
@phosphor/application	0.1.5	BSD-3-Clause	github
@phosphor/algorithm	0.1.1	BSD-3-Clause	github
@phosphor/domutils	0.1.2	BSD-3-Clause	github
@phosphor/messaging	0.1.2	BSD-3-Clause	github
@phosphor/signaling	0.1.2	BSD-3-Clause	github

@phosphor/virtualdom	0.1.1	BSD-3-Clause	github
@phosphor/widgets	0.1.7	BSD-3-Clause	github
reflect-metadata	0.1.10	Apache-2.0	github
vscode-ws-jsonrpc	0.0.1-alpha.1	MIT	github
vscode-languageserver	3.2.0	MIT	github

The following node.js modules are not yet used but are candidates for future development

Module name	version	License	Src repo
xterm		MIT	github
d3		BSD-3-Clause	github
jointjs		MPL-2.0	github
nodegit		MIT	github
sprotty		Apache-2.0	

sprotty Dependencies

sprotty (graphics framework) has first-level dependencies on the following node modules (excluding development dependencies):

Module name	version	License	Src repo
inversify	3.1.0	MIT	github
snabbdom	0.6.4	MIT	github
snabbdom-jsx	0.3.1	MIT	github
snabbdom-virtualize	0.6.0	MIT	github
file-saver	1.3.3	MIT	github