Extending Recogito Studio with Plugins

Jamie Folsom[1]

[1] Performant Software Solutions, Boston MA, United States

Abstract

This paper describes the Recogito Studio collaborative annotation software platform, and how it can be extended by software developers, using a new extension mechanism. I discuss several existing extensions, or ”plugins” to illustrate what an extension can do. The paper concludes with some reflections on how to manage collaborative software development across multiple institutions with different priorities and needs.

Keywords: annotation, software development, extensible code, collaborative development, open source

1 Introduction

Recogito Studio is a web-based platform for collaborative annotation of text and images. It supports a wide range of annotation types. The platform is designed to be user-friendly, with a focus on accessibility and ease of use for both technical and non-technical users.

It has been developed by Performant Software Solutions and Rainer Simon, in collaboration with a series of partners, whose use cases sometimes converge, and sometimes diverge, which presents the project with a challenge: how to design a system which can accommodate a range of use cases. This paper describes the solution we have arrived at: a modular architecture that allows for easy extension and customization through plugins.

1.1 Details

Scholars and practitioners from seven different organizations in five countries, and dozens of adopting institutions from major public research universities to small research organizations and software development companies, have worked together to produce a new open source software platform for collaborative annotation of text and images, called Recogito Studio.

Developed starting in January 2023, with a first version released at the end of 2024, Recogito Studio incorporates ideas from earlier projects, chief among them Recogito (from the Pelagios Project), and Annotation Studio (built at MIT), and adds a number of new features for collaboration, extensions and integration with other tools and workflows.

Several of its features are important for all project stakeholders (like support for widely-adopted standards like TEI-XML for text and IIIF for images), while others are specific to research or teaching in one field, like those aimed at art historians or archaeologists. In a project funded and developed jointly by several organizations, this raises several interesting questions, including:

- How can we collectively sequence and prioritize features for development?
- Which features should be included in the “core” of the software and who decides?
- How can non-core features be identified, developed and deployed as needed?

Jamie Folsom. “Extending Recogito Studio with Plugins.” In: *Digital Humanities Tech Symposium 2025*, ed. by Julia Damerow and Rebecca Sutton Koeser. Vol. 2. Anthology of Computers and the Humanities. 2025, 19–24. https://doi.org/10.63744/yYHjphZVP1PZ.

© 2025 by the authors. Licensed under Creative Commons Attribution 4.0 International (CC BY 4.0).

- How can we design features with two or more stakeholders?
- What roles should users, developers, and funders play in project governance?

This paper provides an overview of the resulting software and of the strategy we have adopted to answer the above questions: a plugin mechanism that allows the development of new features without adding them to the core codebase. Additionally, it covers:

- Existing plugins and what they add to the platform.
- What plugins can and cannot do currently.
- Starting points for plugin development.

We conclude with some of the benefits of this plugin architecture for the Recogito Studio project and community, which may be relevant to other digital humanities software projects.

2 Overview of Recogito Studio

2.1 Purpose

Recogito Studio is a web-based platform for collaborative annotation of text (including TEI-XML) and images (including IIIF images). It supports import of text documents, IIIF manifests, and PDFs, and export of annotations in CSV and JSON-LD formats; TEI annotations, both standoff and inline, and embedded in derivative IIIF Manifests. The platform is user-friendly and accessible, designed with a focus on ease of use for end users.

It's in use by over 20 institutional adopters of the COVE Collective, the ATRIUM project in the EU, the University of Bonn, the University of Exeter, and Duke University.

2.2 Architecture

Recogito Studio is a web application with client and server components. The client component is built using the Astro website framework, while the server uses the supabase platform, based on the open source PostgreSQL database. Together, these two components form the core platform, which provides essential features such as management of users, documents, projects, and assignments (subgroupings of project documents and members); annotation storage, and basic annotation types.

The plugin mechanism we discuss in this paper allows the client to be modified using plugins written in JavaScript and installed into the Astro client at build time, using documented extension points to add new functionality or modify existing features without otherwise altering the client or server codebase.

2.3 Major Features

Recogito Studio includes several features that differentiate it from other annotation tools:

- Support for annotation on IIIF images, TEI text and PDF files.
- Real-time collaboration, allowing multiple users to annotate simultaneously.
- GDPR-compliant user and user data management, with support for customizable role-based access.
- Support for both password-based authentication and integration with SAML IdPs for single sign on.

2.4 Differentiation

Recogito Studio differentiates itself from other annotation platforms by its support for widely-adopted standards like TEI-XML, IIIF and W3C Annotations (as JSON-LD) to ensure compatibility with existing tools and workflows in the humanities. The plugin mechanism allows for rapid development and deployment of new features, enabling users to tailor the platform to their specific needs.

3 Plugin Mechanism

The plugin mechanism in Recogito Studio allows developers to extend the platform's functionality without modifying the core codebase. A good example of a common use case for this option would be connecting to an externally managed controlled vocabulary server with terms specific to a given project. That's enabled by the Reconciliation API plugin, for which there's a link in the appendix. Because each project can turn every available plugin on or off, and configure it differently, even projects within a single instance of the platform could use different externally managed controlled vocabularies with this plugin, or none at all.

Plugins are developed using the Recogito Studio SDK, for which we include a link in Appendix A. The SDK includes a boilerplate plugin template that developers can use as a starting point for their own plugins.

The Recogito Studio SDK is designed to facilitate plugin development for the Recogito Studio Client without the need to set up the Recogito Server application in a local development environment or forking the code. It provides essential tools and abstractions to help you build useful extensions for the Recogito Studio ecosystem. It provides a test application that can be run locally before installing a plugin in a running instance of Recogito Studio, so developers can do most of their work locally.

3.1 Astro Integrations

The Recogito Studio Client is an Astro application. Recogito Studio plugins are implemented as Astro Integrations, the extension mechanism for the Astro framework, plus some additional features introduced by Recogito Studio's own UI extension framework, as described in the next section.

3.2 UI Extension Framework

Astro allows developers to write components in React to add interactivity to otherwise static content pages. Recogito Studio uses this capability to implement its annotation interfaces in React. Building on that foundation, Recogito Studio uses the React lazy loading and Suspense mechanisms to allow plugins to dynamically inject their own React components into extension points that the Recogito Studio Client defines. The SDK also provides built-in CSS style classes, making it easy for extensions to re-use Recogito's native styling.

To inject UI extensions, plugin developers must register their extensions following a specific process. This involves a small amount of boilerplate code, which the SDK helps simplify.

3.3 What Plugins Can Do

Currently, plugins can make changes to many of the core features of the user-facing client application, including:

- Adding to the annotation interface (like the geotagging plugin).

- Connecting to external data sources and services (like the NER plugin).
- Replacing existing UI elements in the editor, such as swapping the default tag widget with a custom version that integrates an external vocabulary service.
- Enhancing page and API capabilities, e.g. introduce new API routes that export annotations in different data formats.

As of now, plugins extend the Recogito Studio Client only. Plugins cannot extend the Server component of Recogito Studio. For example, it is not possible for plugins to change or add to the database schema, or add additional database functions or edge functions.

3.4 Example Plugin Code

The complete documentation for the development of plugins is available in Recogito Studio SDK, but here is the code of a simple plugin to provide an idea of the skills required to create one: knowledge of modern JavaScript and TypeScript development, publication of packages on npmjs.com, and domain-specific knowledge required to make the plugin itself. The following code defines and registers a new plugin, but does not install it.

```
// index.ts
import type { AstroIntegration } from "astro";
import { Plugin, registerPlugin } from "@recogito/studio-sdk";

const HelloWorldPlugin: Plugin = {
  name: "My Hello World Plugin",

  module_name: '@performant/plugin-hello-world',

  description: "An example Hello World plugin.",

  author: "Performant Software",

  homepage: "https://www.performantsoftware.com/",
};

const plugin = (): AstroIntegration => ({
  name: "plugin-hello-world",
  hooks: {
    "astro:config:setup": ({ config, logger }) => {
      registerPlugin(HelloWorldPlugin, config, logger);
    },
  },
});

export default plugin;
```

3.5 Existing Plugins

Several plugins have already been developed for Recogito Studio, showcasing the platform's extensibility and flexibility. Some notable examples include:

Geotagger. Adds geotagging capabilities, configurable with multiple external gazetteers, allowing users to identify place names in text and image documents, visualize them on a map, and export them in GeoJSON format.
NER Plugin. Enables connection to an external named entity recognition service, allowing automated annotation entities such as people, organizations, and locations in text documents.
Reconciliation API Plugin. Enables connection to an external name authority service for use as a source for tags from a controlled vocabulary.

A full list of existing plugins with brief descriptions and links is available in Appendix B.

4 Conclusion

Plugins enhance the flexibility of Recogito Studio, by making it possible for developers to extend the Recogito Studio client component with relative ease. The plugins developed to date have been developed by the members of the core development team, in collaboration with a range of external stakeholders.

We hope that by reducing the barrier to entry for developers to get involved in the project, we can encourage a broader community of contributors to participate in the platform's evolution and in turn make the platform more useful to a wider range of audiences.

Since plugins can be developed using JavaScript, the world's most widely used programming language, and independently of the core codebase, developers can focus on specific features or integrations without needing to understand the entire platform. A plugin mechanism with these characteristics offers one possible set of answers to the questions in our introduction.

Since there is no permanent institutional support for the Recogito Studio project, we often *can't* predictably sequence and prioritize features; rather, we must respond to requests which originate with adopters who can fund the features they require.

In this context, implementing a smaller set of core features in the server component, and allowing developers to add new features easily to the client component as plugins removes the necessity of a more-rigid governance process. Hence, it makes sense to include only features that are clearly essential, or universally useful, in the core of the software; anything specific to a more esoteric use case should be implemented as a plugin, so it's available to those who need it, without pulling the core feature set in a direction that is less than optimal for the more common use cases.

For the more esoteric use cases, plugins remove constraints. There is a low technical barrier to entry for developers, and no requirement to achieve consensus with a dispersed user community to inform design and development. Plugins don't take long to develop, which means that they are inexpensive, reducing the requirement to raise significant funding. Stakeholders whose requirements are similar to a point, but diverge subtly or significantly can develop and deploy whichever version of the feature they need.

While annotation is a widely used technology, it takes innumerable forms, so it makes sense to provide a core set of features, and to allow adopters to define new features without consideration for how their requirements may affect others' use cases. Developers should likewise be able to develop and deploy new features as plugins, without needing to modify the core codebase, and funders should be able to fund specific features or integrations as plugins, without needing to fund an entire platform.

This plugin architecture supports these roles by making it easy for developers to add new features as plugins, without needing to modify the core codebase. While there is currently no funding or organizational capacity to support developer communications and other desirable activities, both developers and adopters have contributed time and effort to those maintenance tasks.

We hope these factors will encourage adoption of Recogito Studio, allowing for rapid prototyping and deployment of new features, and there is evidence that it will, in the form of the several

plugins developed by the community to date.

Acknowledgments

We would like to acknowledge the contributions of the Recogito Studio design and development team, including but not limited to: Rainer Simon, Lorin Jameson and Chelsea Giordan. Many collaborators have participated in the codesign of the application, and their institutions have contributed significant funding to the project as well. Jann Mueller and Christian M. Stracke at the University of Bonn; Dino Felluga of the COVE Collective; Ed Triplett at Duke University; Fabrizio Nevola and Leif Isaaksen at University of Exeter; and Elton Barker of the Pelagios Network. The ATRIUM Project and Ruhr University Bochum have also adopted Recogito Studio and have provided funding for development.

A Resources

- Project Website: `https://recogitostudio.org/`
- Self-Hosting Documentation: `https://github.com/recogito/recogito-studio`
- Plugin Template: `https://github.com/recogito/plugin-template`
- Plugin SDK: `https://github.com/recogito/recogito-studio-sdk`
- Astro Website Framework: `https://astro.build/`
- Plugin SDK: `https://www.supabase.com/`

B Existing Plugins

A current list of existing plugins can be accessed by visiting: https://github.com/recogito/?q=plugin. As of now, that list includes the following:

- Recogito Studio plugin that can perform Named Entity Recognition on TEI and Plain Text: `https://github.com/recogito/plugin-ner`
- Allows users to make a read-only annotation editable, by cloning it into the active layer: `https://github.com/recogito/plugin-revisions`
- A plugin for exporting TEI annotations as inline markup: `https://github.com/recogito/plugin-tei-inliner`
- Use a Reconciliation Service API endpoint as a tag source in Recogito Studio projects: `https://github.com/recogito/plugin-reconciliation-service`
- A custom annotation export for Duke University's Unreal 3D importer: `https://github.com/recogito/plugin-sandcastle-export`
- Recogito Geotagging Plugin: `https://github.com/recogito/plugin-geotagging`