# GSoC 2018 Proposal
# Article finder tool for Wikimedia Education dashboard

## Basic Information:

### Personal Info:


- Name: Pratyush Singhal
- Nickname: psinghal20
- Github: psinghal20
- Email: psinghal20@gmail.com
- Phone Number: +91-8171790257
- Location: India
- Time Zone: India (UTC+5.30)
- Video Chat: https://appear.in/psinghal20


### Meeting with mentors:

- Reachable anytime through email, slack, zulip, well planned video session if required.
- UTC 5:30 AM to 8:30 PM (IST 11 AM to 2 AM)
- Typical Working hours:
    - Weekdays: UTC 10:30 AM to UTC 3:30 PM, UTC 4:30 PM to UTC 7:30 AM
    - Weekends: UTC 4:30 AM to UTC 8:30 AM, UTC 10:30 AM to UTC 3:30 PM, UTC 4:30 PM to UTC 7:30 AM

# Synopsis:

## [WikiEducation Dashboard] Build an Article Finder tool for program leaders and participants to find good topics to work on

The Wiki Education Dashboard/ Programs & Events Dashboard is a Web Application that helps people organize groups of newcomers to contribute to Wikipedia. The initial steps towards successful Wikipedia authorship are finding and selection of appropriate content to work on. An Article Finder tool will help instructor and students to search and select articles in need of contribution and editing. The basic concept is to create a tool that would allow a user to explore a particular topic on Wikipedia to identify and search for articles that require improvements and are also relevant to their program or event. This way, the participants can have a good set of possible topics to work on. An article finder tool utilizing ORES' machine learning services can lead to vast improvement in productivity for many programs and event.

**Mentors: Sage Ross(@Ragesoss), Jonathan Morgan(@Capt_Swing)**

I have been an active member of the community since January and have been active on the slack channel of the project. I have made a few contributions since January and thus have been in regular contact with Sage Ross. After a good amount of familiarity with the project, I was also able to help other newcomers with their issues, helped review pull requests of other contributors and provide them constructive feedback whenever I could.

# Deliverables:

The aim of the project is to rewrite the Article Finder tool into a modern web app with API based backend and integrate it with the current dashboard. The project will involve writing the Front-end for the app in React.js and Redux frameworks with API in Ruby on Rails integrated with the Current project. The project will also be integrated with context-driven tests and feature specs to ensure the correct behaviour of the app in corner cases and make future development easier and efficient. It will also have analytics and modular error logging using matomo(piwik) open analytics platform and Sentry which are also used in the existing project.

Also, I wish to take some of the bigger issues mentioned in the GitHub issues section which are:

- Migration to Redux from McFly flux architecture
- Upgrading to React 16
- Conversion of remaining Front-end pages to React, achieving a single page application architecture

## Implementation Details

The major work will be refactoring and rewriting parts of Article Finder Tool's backend which is written in Ruby on Rails. The backend will be refactored into a REST API which will interact with a frontend application written in React.js and Redux. The backend will use various **Wikimedia APIs** to gather the required data to make recommendation to the user considering his queries such **MediaWiki API**(to gather data about articles), **ORES API**(for Revision score of the article), **PageViews API**(to get the page views of a particular article), Alpha version of **Gap Finder: Missing Section API**(to recommend missing sections which can be improved for a existing article). The recommendation system will help a new User or even a seasoned one to discover and decide what articles to edit.

## Phase 1 (Deliverable before phase 1 evaluation):

- Refactor the Current Article Finder Tool codebase into a REST API
- Migration from McFly flux architecture to Redux.
- Upgrade to React 16

## Phase 2 (Deliverable before phase 2 evaluation):

- Create React front end components
- Write basic reducers to manage the state of the app
- Write server utilities to communicate with the server
- Add hooks in the tool to add available articles or assign articles.

## Phase 3 (Deliverable before final evaluation):

- Writing tests for all the newly introduced codebase and fixing all possible bugs.
- Upgrade the ORES API version
- Optimize the replica server endpoint to improve server response time.
- Implement the Gap Finder Missing section API in Article Finder tool.

## Future Goals :

**Can be implemented if all above mentioned milestones are achieved and completed successfully and the programme is yet to end, also desired to be taken up post GSoC.**

- Work on campaigns page conversion to React

*A couple of missing days between each phase are meant for final touch ups and preparation for the upcoming evaluation.*

# Phase 1(23rd April - 11th June):

- The current Article Finder tool prototype backend application in Ruby on Rails cannot support the Single Page Architecture that we plan to achieve for our project.
    - The views will be removed from the code and replaced with JSON data which will be served to the react-redux application.
    - Add new endpoints displaying complete and detailed info about the Article Recommendation for a particular article based on various data gathered from Media Wiki APIs such as *GapFinder*, *ORES*, *Pageview API*, *Replica Endpoint*.
      Right now, when clicked on a suggested article, the page is redirected to the Wiki Article. I plan to include one more screen between this, displaying detailed info about the article and its revision history. This page will also support the *Gap Finder API* displaying what part of the article will require editors attention.
    - The Media Wiki APIs that will be primarily used in Article Finder are:
        - Wikipedia API (to get details of the articles)
        - ORES API (to get the Revision scores of an article and help editors find articles which are in greater need of their attention)
        - Gap Finder API (to suggest the Missing Sections for an Article)
        - Pageviews API (to suggest article which has more page views and thus require more attention of the editors)
    - The user will be able to search articles based on:
        - Category of articles
        - Sub-Category depth
        - Minimum page views
        - Completeness score

    - The endpoint will provide the following piece of information about the article:
        - Title
        - Average Views
        - Revision Scores
        - Article's Wiki link
        - Last updated on
    - The Detailed article overview will provide:
        - Detailed article information such as Title, updated on, Average Views etc.
        - Provide recommendation sections with the help of Gap Finder API
        - Provide an outlook of all the past revisions made to the article and rate its completeness.
- Apart from Article Finder API refactoring, I plan to work on the migration from McFly to redux state management. I started working on this even before the start of GSoC program coding phase ie. before the GSoC result declaration and all through the bonding phase.
    - I have worked on this issue long before submission of the proposal, working on below mentioned sub-issues:
        - Converting Chat store(PR #1661) which required creating the chat reducer and appropriate actions
        - Converting InputMixin to Input High-order component(PR #1775)
        - Converting Validation Store(PR #1722), creating Validation reducer, actions and it is corresponding utilities.
        - Converting Upload Store(PR #1826) to redux.
    - Most of the remaining stores are of similar type and use a generalized structure using the Stock Store which can be easily converted to redux and all the utilities separated in a separate stock_utils file if required.

# Phase 2(15th June - 9th July):

- With the backend API ready, I will focus working on React components and Server communication utilities.
    - A rough idea of React components that will be required:
        - A form component with input fields for all the parameters with proper client side validations.
        - An Article List component which will receive the Endpoint response as react props and render the list of all the resulting articles related to our query.

        - A Modal component which renders the overlook of a particular article with its missing sections information and past revisions.
    - The component styling will follow the style guide of the project while reusing as many components as possible. Examples of some such reusable components are *TextInput*, *Button*, *List* component etc.
- All the components' state will be managed with help of Redux architecture.
    - Redux store will be used to provide the components with the desired data through props.
    - Add basic reducers and corresponding actions for state management and updating.
    - All the API callings will be done through the actions with the help of the redux-thunk library. The actions will sync the store state with the data received from the API endpoints.
- All the server utilities which will be required for communication with the server endpoint will be abstracted in separate Javascript module to maintain a simplified and clean codebase.
    - Mainly there will be two javascript utilities required, one for each endpoint.
        - POST request method for submitting form and handling response.
        - GET request method to access the detailed information of a selected article.
        - A third optional method can be added to make the call to Gap Finder API to from client side and reducing the burden on the server and hence improving the server response time.
- Several hooks can be added in the tool to add an article to the list of available articles or assign the article to an editor.
    - Addition of redux actions and server utilities for the article hooks.
    - Add buttons to the layout to add the article to the desired list.

# Phase 3(13th July - 6th August):

- The major work in this phase will be to implement the **Gap Finder API** for the overview modal in the Article Finder Tool.
    - Implement a server side request to the Gap Finder endpoint, compiling all the article details to be sent as the response to the client.
    - The data from the API will be used in *Overview Modal* component and can be used for a quick outlook about the scope of improvement in the article by the editor.
- The codebase will be thoroughly tested, fixing all the possible bugs along with handling the edge cases. Also, I will implement several tests for all the newly

introduced codebase including the *feature specs, Rspec specs and the Jest tests* covering all the corner cases.

- The ORES API currently used in the Article Finder Tool prototype is **ORES API v2** whereas **ORES API v3** is now available for use. An upgrade to this new API will be done, handling and updating the codebase for all the breaking changes.
- Currently, from the prototype, the problem of slow server response is evident. I plan to work on this issue which will require optimization of the server with caching and other similar methods. Also, optimizing the Replica endpoint can help improving the server response. As per my discussion with the mentors, we are still looking for ways to reduce the server response time and plan to research in-depth related to this issue.

# Future Goals(Post GSoC):

- Campaigns page will be converted using React based flow to approach a single page architecture for the application.
    - It will require a refactor of current views for campaigns that will be replaced with JSON data.
    - Several components similar to course pages will be required. The course components can thus help with conversion and several of them can be reused. For example:
        - Course Navbar can be used to create the similar Campaign Navbar
        - Similar edit component for the details
        - Reusable input Fields
        - Article list can be reused
        - Course stats component is also reusable for Campaign stats.
    - Along the lines of Redux architecture which is followed throughout the project, all the components will have their state managed by reducers and actions following the redux architecture.
    - Server utils for communication to the refactored API endpoints can be included in the actions or in separate utility file which will then be imported into the actions.
    - By using Reselect, State based filtering will be implemented for several Campaigns components. Similarly, state based sorting will be provided for list based components.

# Participation:

As mentioned above, I will be available anytime through *Slack*, *Email*, *Zulip* or if required, a well planned video session. I have been and will be in constant contact with ***Sage Ross*** through Slack. I have been an active user of Github since my freshman year. I will continue contributing via Github to our project. Apart from working on the project, I would be glad to help other contributors and users solve their issues and help the maintainers in reviewing Pull Requests.

# About Me:

## Current Education(in progress):

- **University**: Indian Institute of Technology, Roorkee, India
- **Field of study:** Electronics and Communication Engineering
- **Current Year of the study:** Second Year

**How did you hear about this program?**

Our college has an active community of open source developers. Every year, many of our institute's students get selected for Open Source Programs such as GSoC and Outreachy. I heard about this program from a senior of mine who himself completed his GSoC with Cloud-CV last year.

**Will you have any other time commitments, such as school work, another job, planned vacation, etc, during the duration of the program?**

I don't have any long term commitments during the internship period. There will be no clashes between May 13, 2018 to July 12, 2018. After July 12, however, I'll be having classes so I may be able to devote around 5 hours on weekdays and 7-8 hours on weekends. But the major part of the project will be done before my classes begin and thus I will easily able to cope up with the remaining work and deadlines.

**We advise all candidates eligible for Google Summer of Code and Outreachy to apply for both programs. Are you planning to apply to both programs and, if so, with what organization(s)?**

I am applying for Google Summer of Code with only Wikimedia organization.

**What does making this project happen mean to you?**

I recently explored the vast world of open source contribution and from my first experience, I was hooked on it. Now, I have gotten a chance to contribute to the world's biggest digital encyclopedia. I am grateful for this opportunity and assure you about my commitment and determination towards this project. Seeing how WikiMedia Education is contributing to raising the education standard in colleges makes me proud to be able to contribute to this project. I don't only get to work with best developers out there in the world but also to contribute to well being of students just like me, making their life easier and helping them focus on learning.

# Pre GSoC involvements and Experiences:

I have been passionate about Software development since my freshman year when I got to explore the joys of development. Since then I have been working continuously on personal projects and trying to learn new technologies. Learning about new technologies, exploring the world of software development grew from a mere hobby to passion for me. I started contributing to open source platform very recently and I find it quite fascinating. I have always been interested in building software that has practical scale implications, and Open source is the best platform for it. I make commits with clean commit messages and well structured Pull Requests - something which I feel is very crucial for any open source project.

## Coding Skills:

## Programming Languages and Frameworks:

- Fluent in HTML, CSS, Javascript, Ruby, PHP, Python, C/C++.
- Sound knowledge of OOPs, MVC architecture.
- Proficient with Node.js, Ruby on Rails(Ruby), Django(python)
- Experienced with front-end frameworks such as React and Redux.
- Having moderate experience with GraphQl APIs

## Development Environment:

- Ubuntu 16.04 LTS (preferable) or Windows 10

- Fully customized Sublime Text 3 as primary editor.
- Vim(Fully customized) and Nano for small edits.
- Strong concepts of Git, can catch up with Mercurial and SVN if required.

## Other Interests:

- Shell Scripting and Batch Scripting
- Security Enthusiast(Often participate in CTFs)
- Learning Machine Learning Algorithms

I have earned a great amount of experience with the codebase of the project. Also, I have made quite a few pull requests now and most of them have been merged. My contributions were focused towards fixing existing bugs, writing shell scripts for setting up of developmental environment, migrating from bower and migration from McFly to Redux. My contributions just piqued my interest more and more into the project.
Here is a list of my contributions:

- #1614(merged) : Upgrade Lodash version for node_module and fix breaking changes. Fixed #1599
- #1616(merged): Fix create campaign passcode field behaviour. Fixed #1613
- #1619(merged), #1636(merged): Remove zeus preloader which was not working with the latest version of ruby ie. ruby-2.5.0
- #1623(merged): Change admin course table to anchor tags so that they can be opened in a new Tab. Fixed #1577.
- #1627(merged), #1736(merged): Add setup scripts for debian, MacOS and Windows to setup development environment. Fixed #1467.
- #1630(merged): Make users sortable by draftspace characters added in students tab. Fixed #1612.
- #1633(merged), #1634(merged): Enable Rubocop Metric/LineLength and Style/DateTime rules. Fixed part of #1593.
- #1651(merged): replace JSON with Oj for JSON parsing. Fixed #1632
- #1661(merged): replace Chat_store flux store to chat redux store.
- #1697(merged): fixes links to programs from campaigns editors view which required URL proper URL encoding. Fixed #1676

- #1700(merged): fix username links on did-you-know and possible-plagiarism tabs. Fixed #1668.
- #1723(merged), #1721(merged): Fixed React related warnings.
- #1725(merged): replace jquery-ujs with rails-ujs. Fixed #1710.
- #1726(merged), #1727(merged), #1729(merged): Migrate away from bower as it has been deprecated. Fixed #1453.
- #1724(merged): add sorting on redux state for recent_edits via sortByKeys from model_utils. Fixes #1638.
- #1734(merged): Fix broken links in the current prototype of Article Finder Tool.
- #1745(merged): Add the feature to allow change in Home Wiki of a course. Fixed #1741.
- #1733(merged): Add post methods for replica service to get articles by titles. Fixed #1575.
- #1752(merged):Update the setup docs to include instructions for setup scripts
- #1778(merged): Add project based filtering for Edited Articles list. Fixes #1686.
- #1779(merged): Adds tooltip for timeline toggle input field. Fixes #1765.
- #1780(merged): Add VCR cassette support for replica endpoint specs. Fixes #1618.
- #1775(merged): Convert InputMixin to a high-order Component. Fixes #1456.
- #1826(merged): Migrate Upload store to Redux Store.
- #1829(merged): URI encode username of a user for contribution stats. Fixed #1828.
- #1722(open): Migrate Validation store to Redux store.

### Contributions to WikiEdu Dashboard Tools:

- #11(merged): Add check to load parameters in case of POST requests. Required for WikiEdu Dashboard issue #1575.

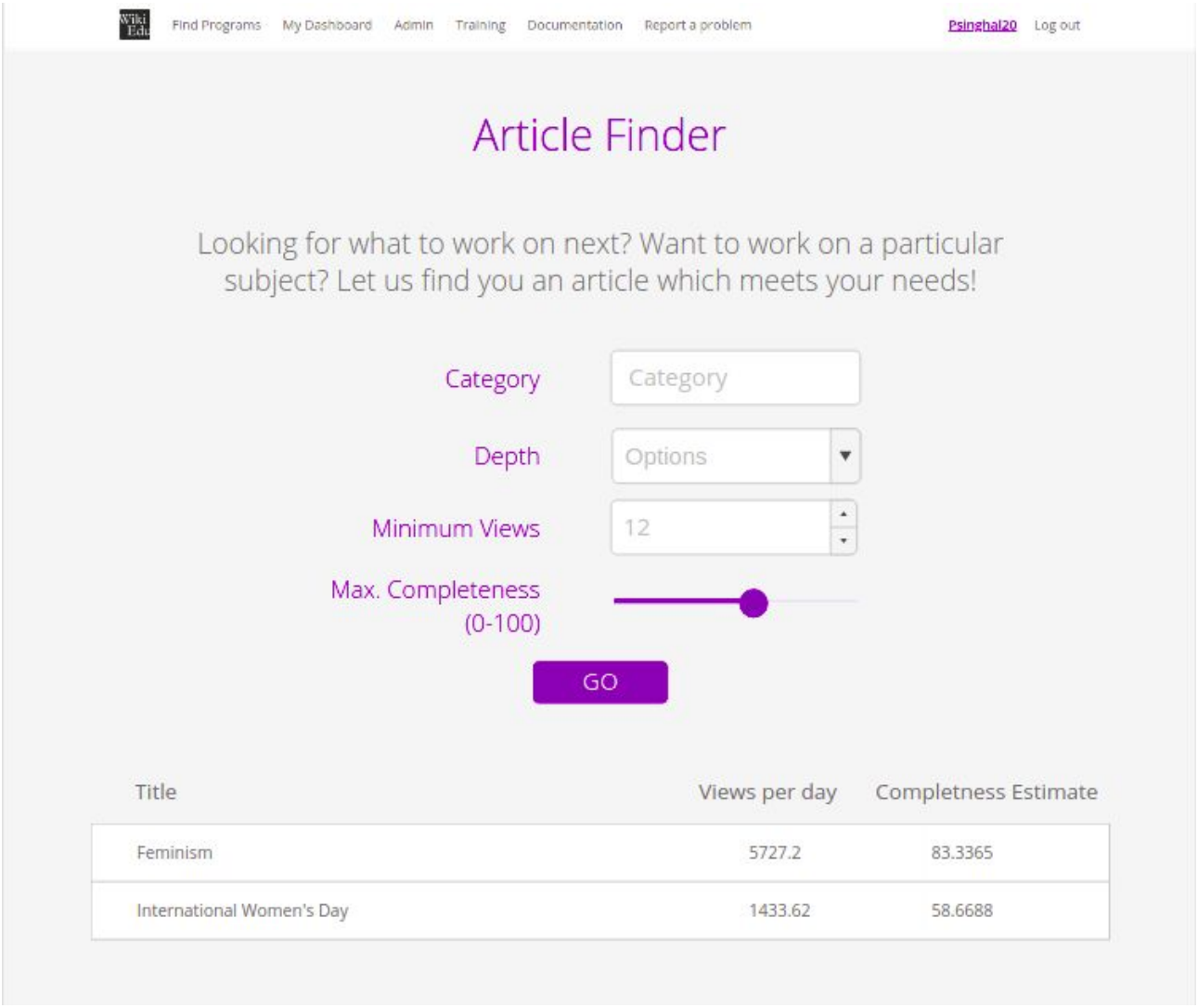
## Mock Ups and Flow Charts:

I have created some design mockups and control flow diagram of back-end and front-end tech stack to give a better insight of how I plan to implement the project. The flow charts

make it easy for anyone to get familiar with the process being executed underneath the project.

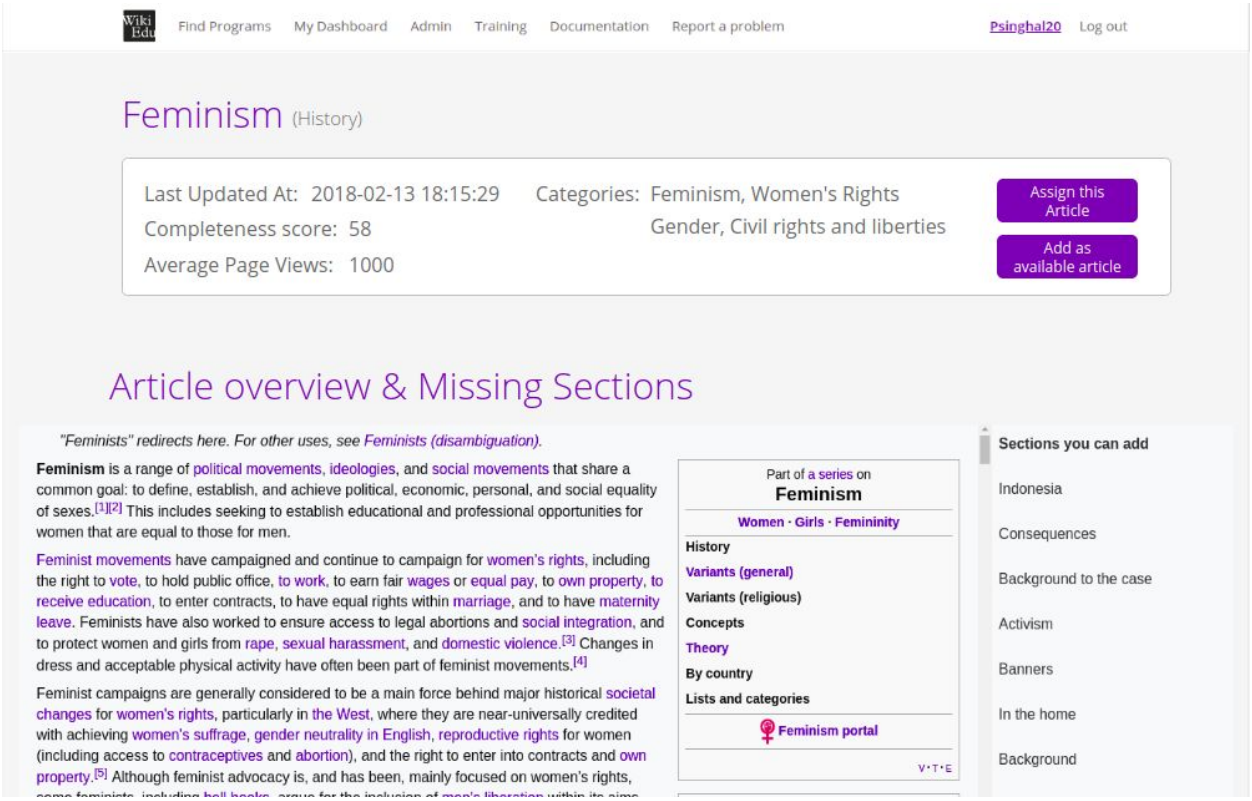
## Article Finder Tool Mockup:

The mockup can be found here.

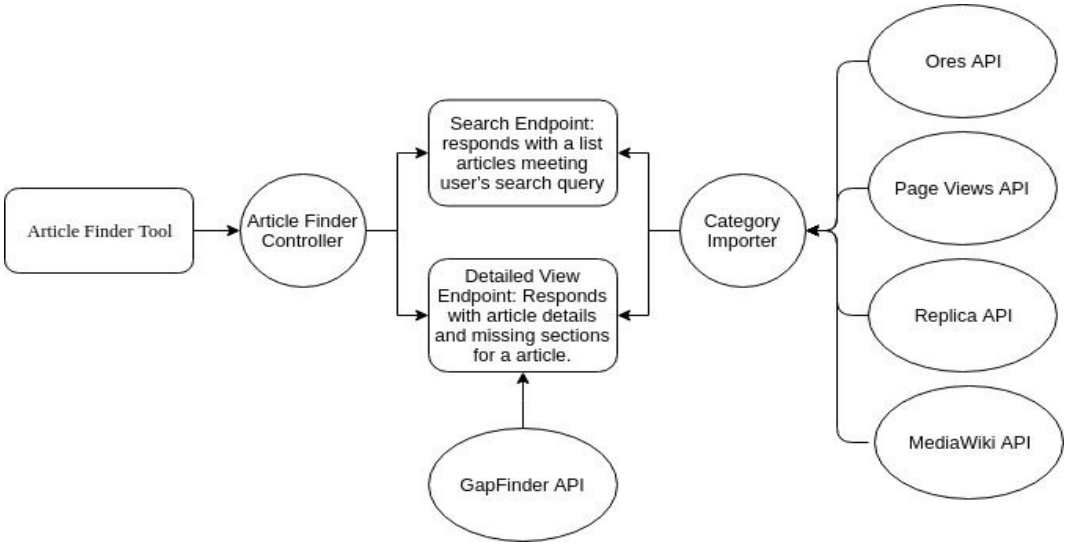


# Detailed Article View with Missing Views section:

The mockup can be found here.



## Back-end Flow:




## Front-end Flow: