A Brief Overview on The Orange Onion's Software Design Choices

Brandon Nguyen
Tung Nguyen
Garret Feng

California State University, Fullerton

CPSC 254: Software Development with Open Source Systems

Abstract

As part of the curriculum and in line with the philosophy of open-source development, our team has put together this project by only using open-source packages and text editors. We have also elected to license our work via the GNU General Public License Version 3 – a strong copyleft license. This document will go on to explore the design considerations behind our project, as well as explain the gist of how the program works. While the development side of the project was completed using only open-source tooling, attempting to do the same with documentation turned out a too challenging given the limited amount of time we had to learn new tooling for documentation. As such, the contents of this document were generated using a mix of proprietary and partially open-sourced software. Also worthy of note is that this project is only aiming to provide a minimum viable product as a proof of concept; the limitations of this project will be discussed in greater detail in further sections.

About The Orange Onion

The Orange Onion (TOO) is a satirical news website that is home to articles centered around California State University, Fullerton (CSUF). TOO gets its name from one of the school's official colors, as well as the name of the American satirical news network that inspired this project. TOO's namesake project is essentially a full-stack web application written in Python. Being a proof-of-concept work, TOO contains the bare essential functionalities and user interface styling for evaluation purposes. The project is not production ready but has been checked and tested for correctness to the best of our abilities.

Project Scope

As a proof-of-concept, we expect to provide users with the ability view and access saved articles; contributors with the ability to log in and save articles. As aforementioned, all dependencies of the project are open-sourced projects, and TOO itself is open-sourced as well. This should serve as a good starting point for future iterations and derivatives to be working off from.

TOO is not meant to be a production-ready solution. As such, we will not be concerned with various features and functionality that one might expect from enterprise software, such as high availability, throughput, and reliability; low access time; polished user interface (UI) and user experience (UX) refined through pilot tests and other trials; test-, build-, and deploy-related automation as part of Continuous Integration and/or Continuous Delivery (CI/CD), to name a few.

While documentation is provided and cursory testing has been performed, we can only support x86_64 Windows 11 and x86_64 Ubuntu Server 22.04 systems as those platforms are what our development machines and test benches are running. The website will also not provide proper Secure Sockets Layer (SSL)/Transport Layer Security (TLS) for securing network traffic, nor Domain Name System (DNS) registration since those are irrelevant for our intents and purposes.

Technology Stack

Flask. This project is built and tested with Python 3.10 through 3.12, and uses Flask - a web application framework, since Flask is lightweight despite being "batteries included" (or so to speak) and thoroughly documented. Some of these "batteries" include a Werkzeug Web Server Gateway Interface (WSGI) development server for the back-end, and Jinja2 templating engine for the front-end. Since deployment-related concerns are beyond the scope of this project, we chose to make use of its built-in WSGI server (NOT RECOMMENDED).

Jinja2. For the front-end, while we elected to make use of Jinja2, we also decided to style our UI elements using Bootstrap's compiled distribution files via jsDelivr's content delivery network (CDN). The rationale for using Bootstrap is such that we can create a modern-looking UI without re-inventing the wheel; the former is an important point, considering the Principal of Least Astonishment (PoLA). After deciding to make use of Bootstrap, we deliberated and opted for using Bootstrap's pre-compiled distributions over their source files after having decided that given the project scope, the provided utilities and/or Cascading Style Sheets (CSS) variables from the compiled distribution would suffice.

Bcrypt. User authentication and session persistence are both part of our project scope. In accordance with standard practices, we chose to use the bcrypt library to hash passwords as part of the authentication logic. Along being a former industry standard, bcrypt boasts being free and open-source software (FOSS).

SQLite3. For data persistence, we opted for SQLite3 (which is included as part of the Python Standard Library) given that it is sufficient to meet the needs of the project.

Software Requirements

Primary Requirements

The primary requirements for TOO can be segregated into two categories: those that apply to both our target audience (whom would be the student body of CSUF, or perhaps more generally, the public), and those that apply solely to contributors of the site.

Requirements for Target Audience

End users of the website should be able to:

- Should be able to look through a list of excerpts from published articles to select those that are of interest.
- Articles should generally be ordered by date of publishing and have the option for page selection.
- Upon discovering article excerpts that strike their interest, users should be able to expand the excerpts to expose the full article contents.

Requirements for Contributors

Contributors should be able to:

- Be able to do everything an end user should be able to (above).
- Write articles for the site.
- TOO should require contributors to authenticate (login) prior to article publishing.

Secondary Requirements

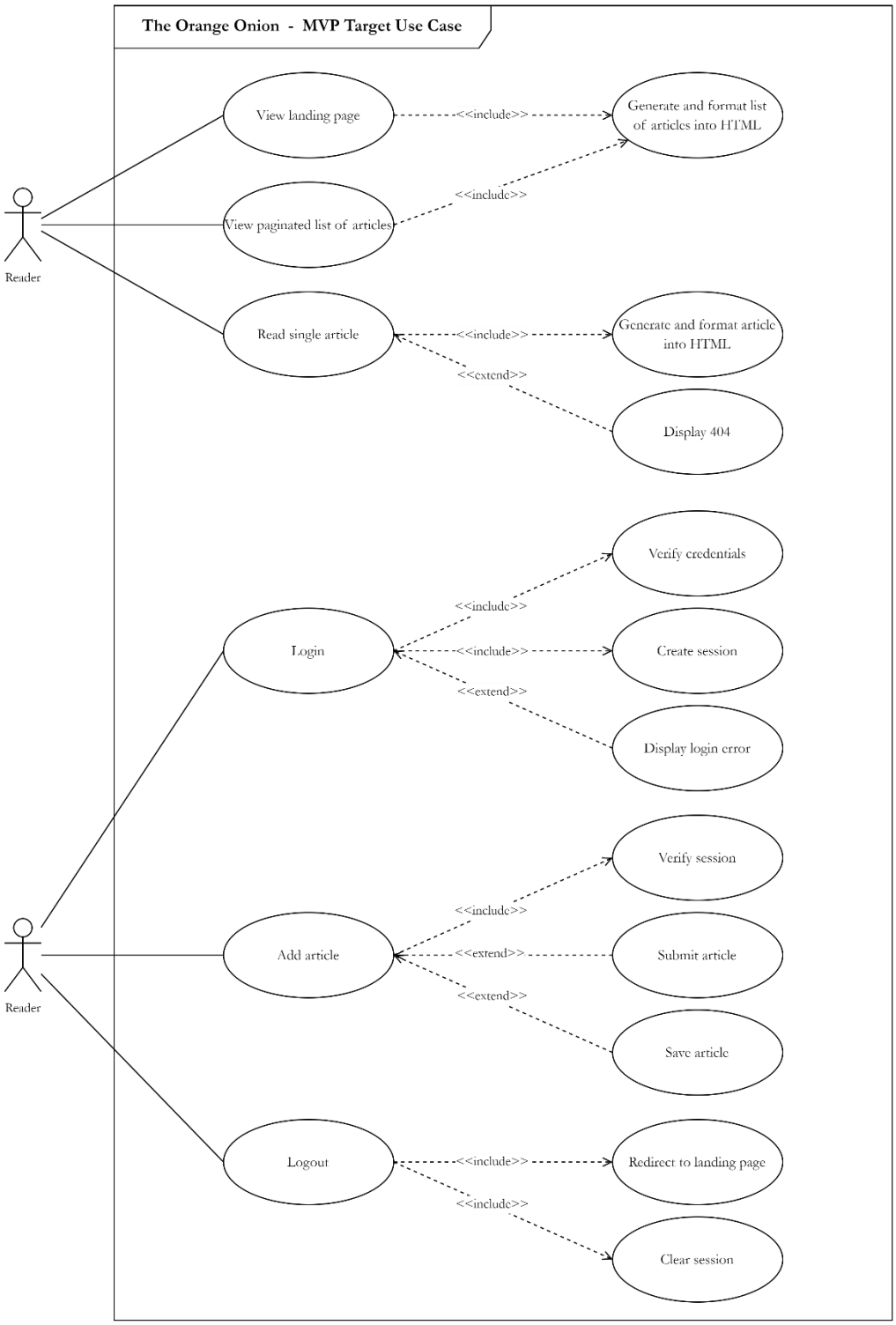
The website should also be able to fulfil the following:

- TOO should provide a consistent UI/UX across the entire site to avoid confusing the users.
- TOO should respect responsive web design, to remain usable even on mobile devices.

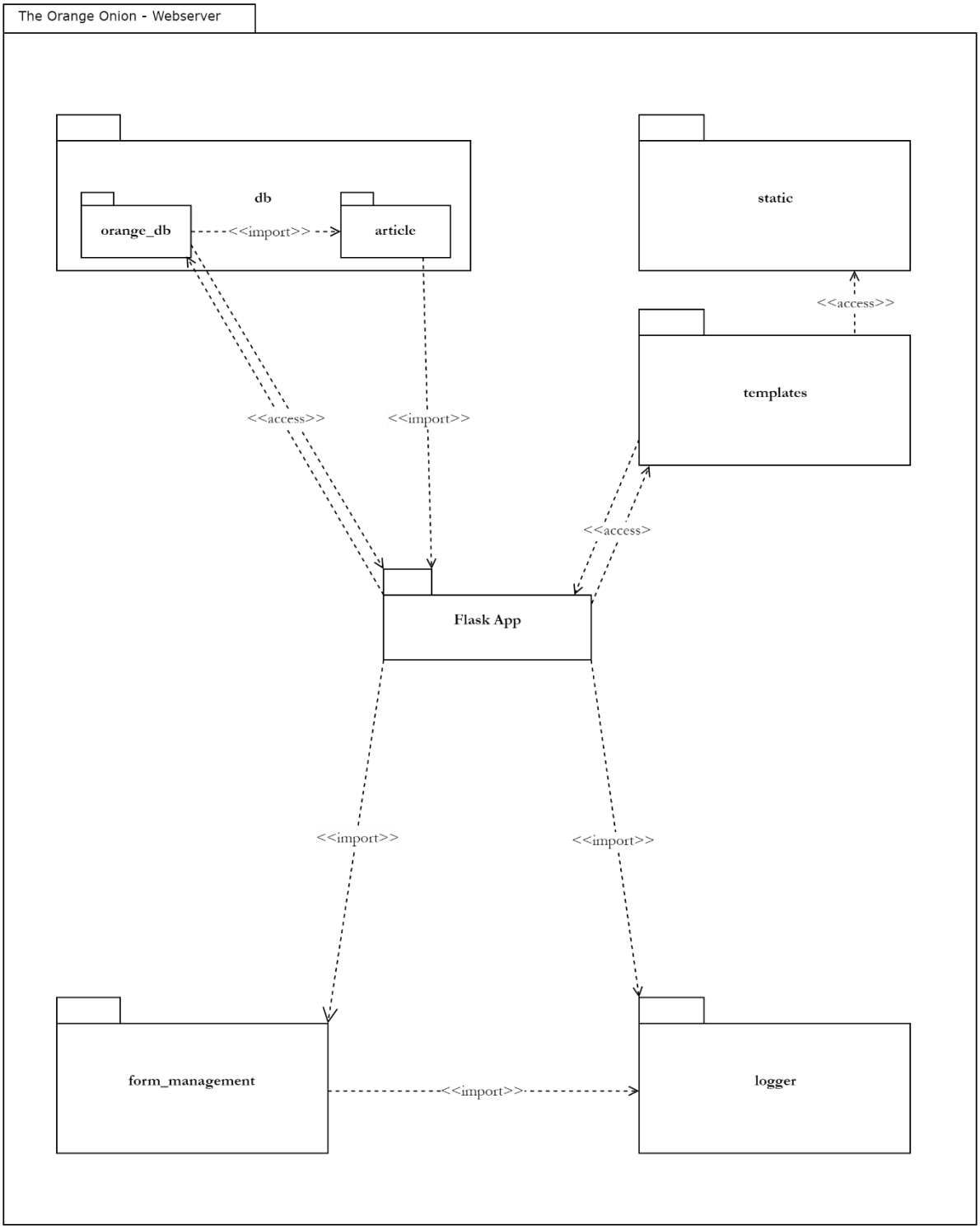
Software Architecture

Having identified the key requirements, the next step is to represent them with high level diagrams to weed out obvious flaws in logic. We first drew a UML Use Case Diagram to model the behavior of the system, followed by a UML Package Diagram to provide a high-level plan for how we will split the logic into discrete maintainable pieces. These diagrams have been included in the following pages:

UML Use Case Diagram



UML Package Diagram



Data

Aside from modeling behavior and structure with the above diagrams, we also modelled the database structure with an Entity-Relationship Diagram (ERD), as well as the data flow with a series of Data Flow Diagrams (DFD):

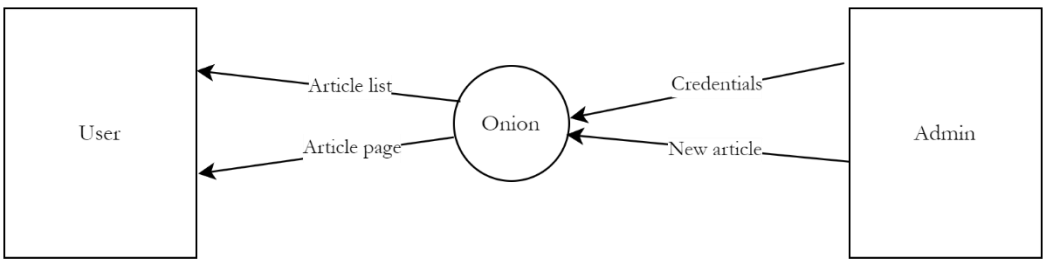
ERD

accounts	
PK	id INTEGER (not null)
Key	username TEXT
Key	password TEXT

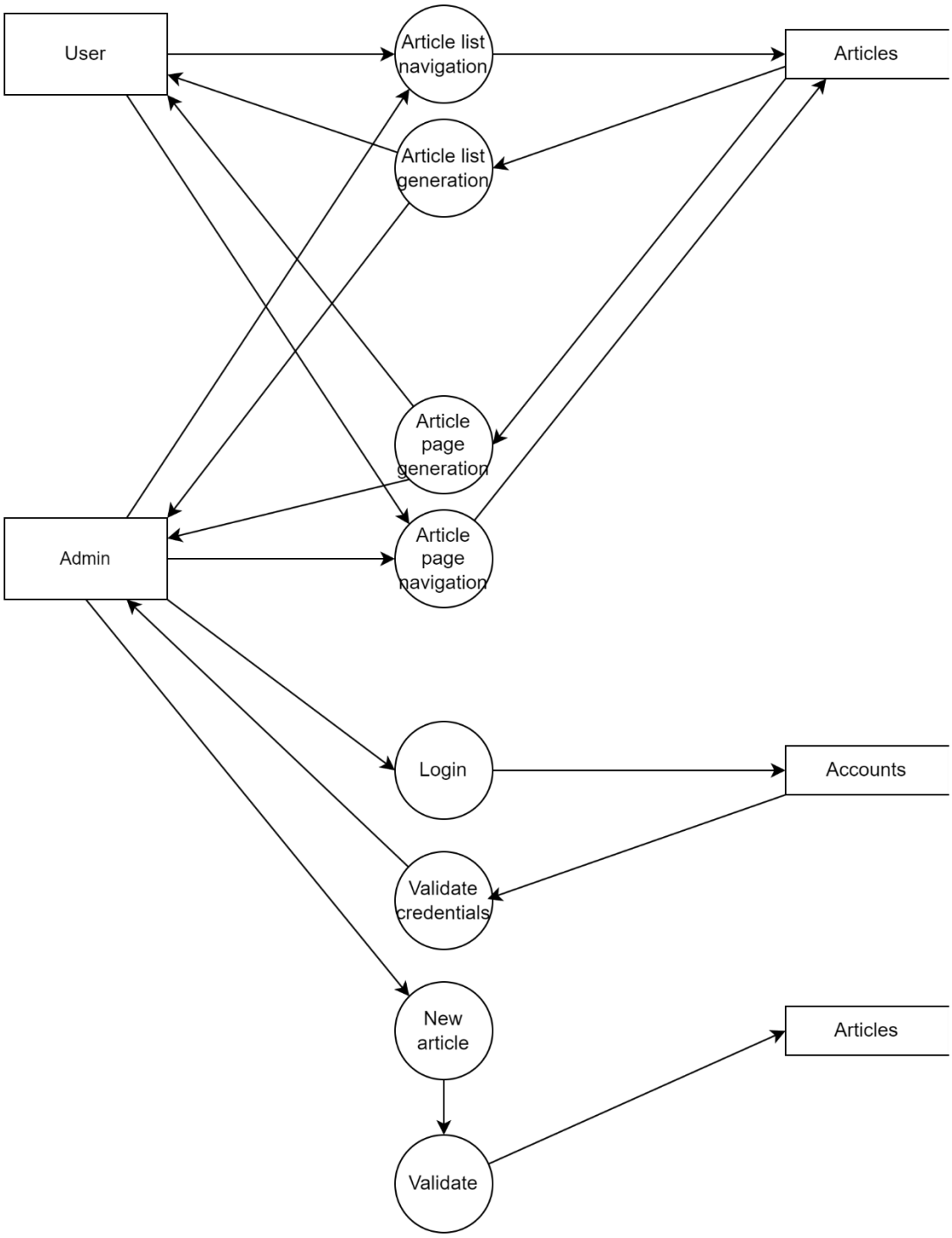
articles	
PK	id INTEGER (not null)
Key	title TEXT
Key	content TEXT
Key	long_content TEXT
Key	date datetime

Note: as it turns out, we did not need any relationships between entities for this project.

Level 0 DFD



Level 1 DFD


User
Article list navigation
Articles
Article list generation
Article page generation
Admin
Article page navigation
Login
Accounts
Validate credentials
New article
Articles
Validate

Limitations

As previously mentioned, there are a number of areas which TOO's capabilities are limited in due to the project scope – one of which is the use of Werkzeug's development WSGI server on our demo machine. This is not a particularly stable nor efficient solution and is intended solely for development use. Another topic that was briefly touched on in an earlier section is the use of a compiled Bootstrap distribution. While this is sufficient for creating a simple, minimal, yet modern UI, it also limits the styling options we have access to.

Another area of concern is the bare-bones text styling options. The web page for creation of new articles makes use of vanilla HTML forms (with UI styling) and does not support rich text/What You See Is What You Get (WYSIWYG) editor input. Notwithstanding that most articles will probably not require rich text editors; contributors are probably more used to having access to these anyway – after-all most forums and blogs have had WYSIWYG editors for years.

Aside from that, one other major limitation is Python's Global Interpreter Lock (GIL). While we noted that high throughput is not necessary for a proof-of-concept, this is no doubt a major pain point, nonetheless. The GIL only allows a single thread to hold control of the interpreter, will be a significant performance bottleneck if the TOO project is slated for production. The obvious solution is to switch languages, however as a proof-of-concept, python was completely fine, and flask was an obvious choice for this type of web application.

Areas for Future Extension

Aside from the limitations we identified earlier, we have also noted some other features and functionality that would improve the user experience of TOO. These have been compiled into the following sub-sections for the consideration of any developers looking to take TOO one step further, or indeed, any developers looking to create derivative works.

Security Improvements

While TOO employs password hashing, the algorithm used is bcrypt. This was chosen out of familiarity since it is formerly the industry standard. Moving forward, we would recommend the use of a more modern option instead. Argon2 stands out, given that it is not only the winner of the 2015 Password Hashing Competition, but also that the reference C implementation; the low-level C Foreign Function Interface (CFFI) bindings for the reference implementation; the derivative Python wrapper for the CFFI (argon2-cffi); as well as the Flask-Argon2 wrapper library for argon2-cffi are all open-sourced. The reference implementation of Argon2 is dual-licensed under the Creative Commons CC0 License and the 2.0 version of the Apache License; the other three are licensed under the MIT license.

Besides the hashing algorithm, there are also various areas of cyber security that would be important for future versions of TOO to improve upon. Since it is not part of the project scope, we have not performed penetration testing, and are unsure of whether input sanitation have been sufficiently taken care of in every possible attack vector, be it SQL injection attacks or cross-site scripting (XSS) attacks. At present, TOO does not use resources from any other origin, but, as the inevitable feature creep progresses, future iterations will probably need to pay close attention to the possibility of XSS attacks.

Scaling Considerations

Having mentioned penetration testing, on a slightly tangential note, TOO does not currently have unit nor integration tests. Moving forward, test coverage is an area we could improve upon.

Another area which could be extended in the future is scaling. As previously discussed, the use of non-production ready WSGI server should be rectified when the project is being considered for production. Unfortunately, most production-capable WSGI servers only run on Linux – using these may mean having to migrate away from and drop support for Windows-based workstations for development. If, ironically, future iterations of TOO were to want to retain compatibility with Windows machines, care must be taken to ensure that the WSGI server of choice is cross-platform – for example, Waitress, which is written in native Python and consequently cross-platform.

To take this one step further, TOO can also consider making use of NGINX for load balancing and/or reverse proxy (should there be a need for RESTful APIs to be supported down the line), and perhaps even migrating to an Apache HTTP Server.

In a similar vein of tech stack upgrades, future developers might want to reconsider TOO's database as well. At present, TOO relies on an embedded database which suits the needs of the project, but it is hard to say whether this will continue to be the case. Shifting project requirements may result in SQLite3 not scaling well enough in production. One option to remedy this could be to migrate to a NoSQL option, such as MongoDB. This allows TOO to continue to use simple and flexible database designs, whilst having the added option of horizontal scaling.

One last point of consideration we would like to bring up about scaling, is CI/CD. While the current project is simple enough to deploy manually (and is in a loose sense "GitOps"), TOO would certainly benefit from having a CI/CD pipeline fleshed out, to reduce the likelihood of human errors causing irreversible damage in production as the project grows in complexity.

Additional Features

TOO's search functionality always returns a placeholder at moment, despite the database wrapper already having logic for string searches. This was an intentional choice since string search is not a requirement for our MVP, and we have not had the chance to sufficiently test it. Moving forward, developers could dedicate development efforts towards improving the search functionality: for example, fuzzy search support. Furthermore, future developers could also index the database at start-up, to offer dynamic search suggestions in a drop-down modal dialog when users are typing in their search query.

Beyond the search UI, TOO's current UI may look a little too generic. If the development team does not require the project to remain a full-stack Python project, the front-end can be offloaded to a NextJS server, and styling migrated to TailwindCSS. TailwindCSS has a similar syntax to Bootstrap whilst offering significantly more built-in utilities. Furthermore, this migration would open the possibility of making use of Quill Rich Text Editor to afford contributors a WYSIWYG editor for creating new articles, whilst keeping all of TOO's dependencies open-sourced.

Lastly, TOO contributors currently have no admin graphical user interface (GUI) for adding new contributors, removing inactive or retired contributors, nor any means for modifying or deleting existing articles. This is another feature that future iterations of TOO could plausibly support.