Aggre-Gator

An RSS & Atom Feed Reader

Gus Johannesen Chris Ross
Ellis Weaver-Kreider Yanxi Wei

CSC 468 – Introduction to Cloud Computing
Dr. Linh B. Ngo — Spring 2025
West Chester University of Pennsylvania

Overview

Many sites publish RSS or Atom feeds, including YouTube, GitHub, and many others. An RSS reader allows a user to subscribe to several of these feeds, and displays updates. We propose a web-based reader, Aggre-Gator, that syndicates an arbitrary number of RSS and Atom feeds, presents them in a user-friendly way, and syncs across devices.

Chapter 1

The core functionality of Aggre-Gator is fairly similar to other feed readers that run as local applications. However, it has some notable differences on account of being cloud-based. The basic function of an RSS/Atom reader is:

- The user configures a list of feeds they are interested in.
- The user instructs the reader to refresh the feeds.
- The reader fetches a list of articles from each feed.
- The reader presents the articles to the user, both as a link and potentially as content.
- The reader saves the articles for later viewing.

Being cloud-based adds additional advantages:

- We can refresh feeds in the background, between site visits, so the user doesn't miss articles.
- Instead of a local application tied to a specific operating system or platform, our app is accessible everywhere.
- The user's subscriptions and article library follow them between devices.

Chapter 2

Aggre-Gator is divided into several logical blocks. The frontend is a static site with client-side interactivity, and is compiled to a static asset bundle which we serve with *nginx* [16]. *nginx* also proxies the `/api` route to the `backend-api` service, which interacts with the database. The feed fetcher service will periodicly update all feeds, and write the updated content to the database.

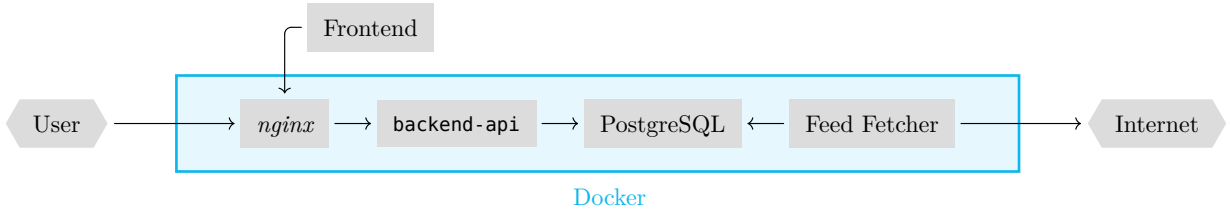

Figure 1: Design of Aggre-Gator

Frontend

Because the frontend is a static site, we have considerable flexibility deploying it. Currently, we simply serve it with *nginx*, but we have the flexibility to serve it using a CDN or with any sort of intermediate caching. In addition, we can set all assets except HTML and the favicon (and *maybe* a couple other things) to be cached indefinitely in the browser, which decreases both load times and the load on our servers.

This caching is possible because our build tool, Vite [19], gives all assets names based on the hash of their contents. Vite also bundles and minifies assets, and can be extended with plugins. We use plugins to further optimize and process images and CSS.

Instead of plain JavaScript, we opted to use Typescript [7] for better error checking and IDE integration. We also have ESLint [9] set up to catch errors and mistakes that the Typescript compiler might miss. Given that we don't need to manage any client-side UI state, we didn't use a Javascript framework. For styling, we are using Tailwind CSS [18] with WCU theme colors.

Nginx

nginx is the only publicly exposed network service in our design. This means that user-facing concerns like SSL/TLS, rate limiting, and HTTP/2 and HTTP/3 are managed in one place, rather than scattered throughout the codebase.

We have the root `/` route configured to serve assets from our frontend bundle. `/api/` is configured to proxy to the `backend-api` container. By separating our content server and backend, we are able to scale them independantly in the future. *nginx* allows us to proxy a route like `/api` to a collection of backing servers, configured in an `upstream` block [17]. We also make sure that `backend-api` is stateless, ensuring the application behaves consistently regardless of which *nginx* or `backend-api` instance a request happens to be routed through.

`backend-api`

The `backend-api` service provides a REST API to the frontend to manipulate the database, and does relatively little data processing itself. Because only the frontend is publicly exposed to the internet, the communication from *nginx* to `backend-api` and from `backend-api` to the database is unencrypted. Sending traffic on the internal container network requires access to the container runtime (Docker [1] or Podman [13]), and if an attacker has access to the runtime, they also gain the ability to read service credentials.

Database

We use PostgreSQL [10] to manage our database. We have a small binary written in Rust, [15] called `db-init`, which initializes the database cluster if it doesn't exist, and replaces itself with the Postgres server if it does exist.

Postgres has the ability to store text data such as article content in the `text` type [11], and, should we need it, can also store JSON efficiently and in a queryable manner in the `jsonb` type [12]. We believe that for this project, any document-style data is best stored in a battle-tested DBMS like Postgres, as opposed to a dedicated document-only database like Mongo [8].

In the future, we may deploy at multiple global points of presence. We may then consider a different DBMS, such as Cassandra, aimed at global-scale data replication.

Feed Fetcher

The feed fetcher will periodically get a list of feeds to refresh from the User DB, fetch the content from the web, and store the result in the Article & Feed DB. Having this as a separate component will hopefully allow us to scale it separately from other parts of the backend.

Chapter 3

To build the Docker (or more accurately OCI) containers, we use Nix [3], specifically the *nixpkgs* [2] function `pkgs.dockerTools.streamLayeredImage`. We chose this both because of the advantages to `streamLayeredImage`, and because we were already using Nix. *nixpkgs* is the largest single repository of software in the world, and one of the most up-to-date [6]. In addition, `streamLayeredImage` is reproducible [14], which makes our deployment more auditable, and less likely to break because of changes to external systems. `streamLayeredImage` also includes only the software we run and its dependencies. This means that our container images are smaller and faster to deploy. It also means that should an attacker get the ability to execute arbitrary commands, they do not have access to a shell or standard tools like `/bin/rm` and similar (Postgres requires `/bin/sh` to run, but still does not contain things like `chmod`). Below is the code to generate an image for our `backend-api` component, taken from `flake.nix` in our repo [5].

```
backend-api-container-stream = { /*...*/ }: dockerTools.streamLayeredImage {
  name = "backend-api";
  tag = "latest";

  config = {
    Entrypoint = [
      "${lib.getExe backend-api}"
    ];
    Env = [
      "DB_HOST=db"
      "LISTEN_PORT=80"
    ];
    ExposedPorts = {
      "80/tcp" = { };
    };
  };
};
```

Figure 2: Nix code for our `backend-api` OCI container

Nix in CI Tests

We also use Nix in our CI testing. In Nix, we can create a derivation (similar to a package) and require that it build for every commit. We require that all derivations for all our

components build before a PR can be merged, to ensure `main` is always in a functional state. We can also create derivations that do not produce meaningful output, but simply check something. For instance, we have a check called `formatting` that ensures that all code in the repo is formatted according to a list of formatters. We also use Nix to build this report, using Typst [4]. We have a check to ensure that before a PR that changes the report source can be merged, the `report.pdf` file must be updated to match.

Building our software and running our tests in Nix gives us some significant benefits. Because Nix carefully tracks all inputs to a derivation, it can avoid rebuilding/retesting when it can be reasonably certain the outcome will be the same. If we try to build something using the exact same source code, dependencies, and environment variables as a previous build, Nix will simply reuse that result. We can achieve a somewhat similar result with more conventional CI systems like GitHub actions, but it requires manually configuring what paths trigger rebuilds, and that these configured paths be kept perfectly up to date.

Using Nix Flakes allows us to pin all our dependencies, including all compilers and even `glibc`, to ensure our build and test environment doesn't change without us changing it deliberately. This pinned environment is the same between development, testing, and deployment. This means that in dev, we can run the exact checks that would run in CI with `nix flake check`, before commiting or pushing.

References

[1] Docker, Inc. Docker: Accelerated Container Application Development. Retrieved from https://www.docker.com/

[2] Eelco Dolstra and Nixpkgs/NixOS contributors. NixOS/nixpkgs. Retrieved from https://github.com/NixOS/nixpkgs

[3] Eelco Dolstra. 2006. The purely functional software deployment model. Doctoral dissertation. Utrecht University. Retrieved from https://edolstra.github.io/pubs/phd-thesis.pdf

[4] Martin Haug and Laurenz Mädje. Typst. Retrieved from https://typst.app/

[5] Gus Johannesen, Chris Ross, Ellis Weaver-Kreider, Yanxi Wei, and Ellis Weaver-Kreider. Aggre-gator Source Code Repository. Retrieved from https://github.com/Spoonbaker/csc468-project

[6] Dmitry Marakasov. Repology - Repositories Graphs. Retrieved from https://repology.org/repositories/graphs

[7] Microsoft and TypeScript contributors. TypeScript: JavaScript with Syntax for Types. Retrieved from https://www.typescriptlang.org/

[8] MongoDB, Inc. MongoDB. Retrieved from https://www.mongodb.com/

[9] OpenJS Foundation and ESLint contributors. ESLint: Find and fix problems in your JavaScript code. Retrieved from https://eslint.org/

[10] PostgreSQL Global Development Group. PostgreSQL Database Management System. Retrieved from https://www.postgresql.org/

[11] PostgreSQL Global Development Group. PostgreSQL Documentation: Character Types. Retrieved from https://www.postgresql.org/docs/current/datatype-character.html

[12] PostgreSQL Global Development Group. PostgreSQL Documentation: JSON Types. Retrieved from https://www.postgresql.org/docs/current/datatype-json.html

[13] Red Hat, Inc. and Podman contributors. Podman. Retrieved from https://podman.io/

[14] Reproducible Builds. Reproducible Builds Website. Retrieved from https://reproducible-builds.org/

[15] The Rust Project Contributors. Rust: Empowering everyone to build reliable and efficient software. Retrieved from https://www.rust-lang.org/

[16] Igor Sysoev and Nginx, Inc. Nginx. Retrieved from https://nginx.org/en/

[17] Igor Sysoev and Nginx, Inc. Nginx Documentation: Module ngx_http_upstream_module. Retrieved from https://nginx.org/en/docs/http/ngx_http_upstream_module.html#upstream

[18] Tailwind Labs, Inc. Tailwind CSS: Rapidly build modern websites without ever leaving your HTML. Retrieved from https://tailwindcss.com/

[19] ZoidZero, Inc. and Vite contributors. Vite: Next Generation Frontend Tooling. Retrieved from https://vite.dev/

Gus Johannesen

484-639-2123 gusjohannesen@gmail.com www.linkedin.com/in/gus-jo

Professional Summary

Upcoming Computer Science graduate seeking employment. Hands-on experience in full-stack development, Agile workflows, and API integration. Highly motivated and eager to build high-impact software solutions. Strong collaborator with a proven ability to enhance digital experiences and drive engagement through innovative technology.

Education

West Chester University of Pennsylvania **September 2021 – May 2025**
Bachelor of Science in Computer Science *West Chester, PA*

Experience

JCI **May 2024 - August 2024**
Application Developer *West Chester, PA*

- Developed full-stack solutions for the Cold Plunge app using Swift and Xcode.
- Collaborated with cross-functional teams in Agile workflows to meet milestones and maintain code quality.
- Enhanced SEO strategies and web traffic analytics to boost user engagement.

Hills Quality Seafood **April 2021 – Present**
Sales and Operations Associate *Glen Mills, PA*

- Manage customer transactions, product prep, and inventory for smooth operations.
- Took on leadership tasks, optimizing market workflows and managing high-volume holiday operations.
- Coordinated 300+ pre-placed orders, creating efficient systems for logging and retrieval.

Covenant Fellowship Church **April 2021 - Present**
Technical Event Support *Glen Mills, PA*

- Operate light-board software and train junior technicians.
- Design visual elements to enhance event ambiance and success.

Projects

Cold Plunge App **May 2024 - August 2024**

- Developed key features using Swift/Xcode, transitioning from timer-based to real-time stopwatch.
- Collaborated on backend integration with Azure/GitHub for version control and deployment.
- Performed QA testing to identify and resolve bugs, ensuring a polished user interface.

Chesco Association for the Blind and Visually Impaired Website **August 2024 - December 2024**

- Collaborated to improve UI/UX for accessibility and user-friendliness.
- Authored documentation for future updates and maintenance.
- Redesigned website pages, enhancing visual appeal and navigation.

Leadership / Campus Involvement

Campus Cru **Sept 2021 - Present**

- Plan and execute weekly meetings and community-building events.
- Serve and invest in the community, inspiring others.

Volleyball Coach **Sept 2021 - May 2023**

- Trained youth volleyball players, fostering a winning mentality.
- Strengthened the team physically and mentally.

Scouting BSA **May 2015 - Mar 2021**
Eagle Scout *Troop 93*

- Achieved Eagle Scout and served as a leader on the executive board.
- Demonstrated leadership, service, and perseverance.

Technical Skills

Languages: Java, Swift
Developer Tools: XCode, VS Code, Jupyter Notebook
Technologies/Frameworks: Git, GitHub, Azure DevOps

CHRIS K. ROSS

215-667-5793 chriskross192@gmail.com

Education

West Chester University of Pennsylvania | B.S. in Computer Science
August 2024 – present

Delaware County Community College | A.S. in Computer Science
August 2021 – May 2024

Interboro High School | Graduated with Honors
September 2017 – June 2021

Work Experience

Chipotle

Service/Kitchen Manager | Multiple locations |May 2022 – present

- Led and trained teams of employees to deliver exceptional customer service and improve operational efficiency.
- Oversaw daily operations, including inventory management and compliance with health and safety standards.
- Collaborated with upper management to implement process improvements, reducing waste and increasing efficiency.

Projects

Text Processor

Developed a Java-based program that parses .txt files, extracts relevant data, and generates .csv files with summarized information to streamline data analysis.

Air-Traffic Control Simulator

Built a Java-based simulation that models air traffic patterns, managing multiple aircraft and simulating real-time control scenarios to ensure safe and efficient airspace coordination.

Relevant Coursework

- Foundations of Computer Science
- Computer Science I, II, & III
- Data Structures and Algorithms
- Computer Systems
- Computer Security & Ethics

Awards & Achievements

- Dean's List Fall 2024, Spring 2024, Fall 2022
- 4.0 GPA

Ellis Weaver-Kreider

717-318-3451 ellisweaverkreider@gmail.com

Work Experience

Production Crew, *Mennocon*
Kansas City, MO

National Convention for Mennonite Church USA
July 2023

- Operated fixed and mobile cameras according to direction of video director
- Mixed audio for subsequent climate summit
- Worked closely with other volunteers and employees to ensure seamless experience for attendees

Tech Crew, *Lancaster Mennonite School*
Lancaster, PA

Thrice-weekly school assembly and outside events
August 2017 - June 2023

- Mixed audio for band, orchestra, and external music groups
- Managed slideshows and other graphical content for presenters
- Operated stage lighting system for dance productions
- Operated fixed and mobile cameras
- Mixed video signals for projection and recording
- Worked effectively as part of a team

Drama Production Crew, *Lancaster Mennonite School*
Lancaster, PA

Set construction, lighting, and sound for stage productions
2017 - 2023

- Worked with other students and alumni to construct complex sets including life-size building facades
- Helped lighting engineer hang and remove lighting fixtures
- Worked as primary audio engineer to mix all actors on stage and a band/pit orchestra for musicals
- Assisted Radio Frequency Technicians in miking and de-miking actors
- Operated sound effect system

Honors & Awards

Academic
- AP Scholar with Distinction
- PHEAA Certificate of Merit
- National Merit Scholarship Letter of Commendation

Programming
- 2024 PACISE Programming Competition – First Place
- 2024 International Collegiate Programming Competition – First Place Regional Division 2

Education

West Chester University of Pennsylvania
Pursuing B.S. in Computer Science

August 2023 - present

Lancaster Mennonite High School

August 2019 - June 2023

Yanxi Wei

1002 Wharton Court • Newtown Square, PA 19073
yw1021529@wcupa.edu • (856) 522-3470

EDUCATION

West Chester University of Pennsylvania, West Chester, PA
Bachelor of Science in Computer Science, Graduation: Spring 2025
GPA: 3.9/4.0

Relevant Coursework :

- Computer Science I, II & III
- Foundations of Computer Science
- Data Structures and Algorithms
- Computer Systems
- Computer Security
- Data Science
- Digital Image Processing
- Program Lang Concepts/Paradigm
- Modern Web Applications

Honors and Achievements

- Dean's List, College of the Sciences and Mathematics (Fall 2024)
- Inducted Member of Upsilon Pi Epsilon (UPE) – The International Computer Science Honor Society (Spring 2025)

Technical Skills

- Programming Languages: Java, Python, C
- Web Development: HTML, CSS, JavaScript, REST APIs
- Cloud Technologies:Docker, CloudLab, Virtualization, Containerization
- Tools & Platforms: GitHub, IntelliJ IDEA, PyCharm, VS Code

ACADEMIC PROJECTS

Email Classification and Spam Analysis System, Team Project (Fall 2023)
Role: reader and Parser- Made CSV parser and reader.
Developed a CSV parser and reader for a spam classification system using a bag of words approach. Assisted in training the classifier to identify common words and phrases in spam and ham emails, and calculated accuracy and Euclidean distance metrics.
Worked with team members to integrate CSV reading with algorithm development for classification, ensuring clean data processing and accurate results.

Obesity Analysis Based on Socioeconomic Factors, Personal Project (Fall 2024)
Role: Data Analysis and Visualization
Conducted statistical analysis on the relationship between obesity rates in the U.S. and socioeconomic factors such as age, income, education level, and gender. Utilized the BRFSS dataset from the CDC and applied regression models, ANOVA, and interaction effect analysis to evaluate key influencing variables. Performed data preprocessing, statistical modeling, and visualization to derive insights for public health interventions, ensuring data-driven conclusions.

Additional Information

- Languages: English (Fluent), Chinese (Native)
- Interests: AI technology, Cloud Computing, Web Development, Data Science