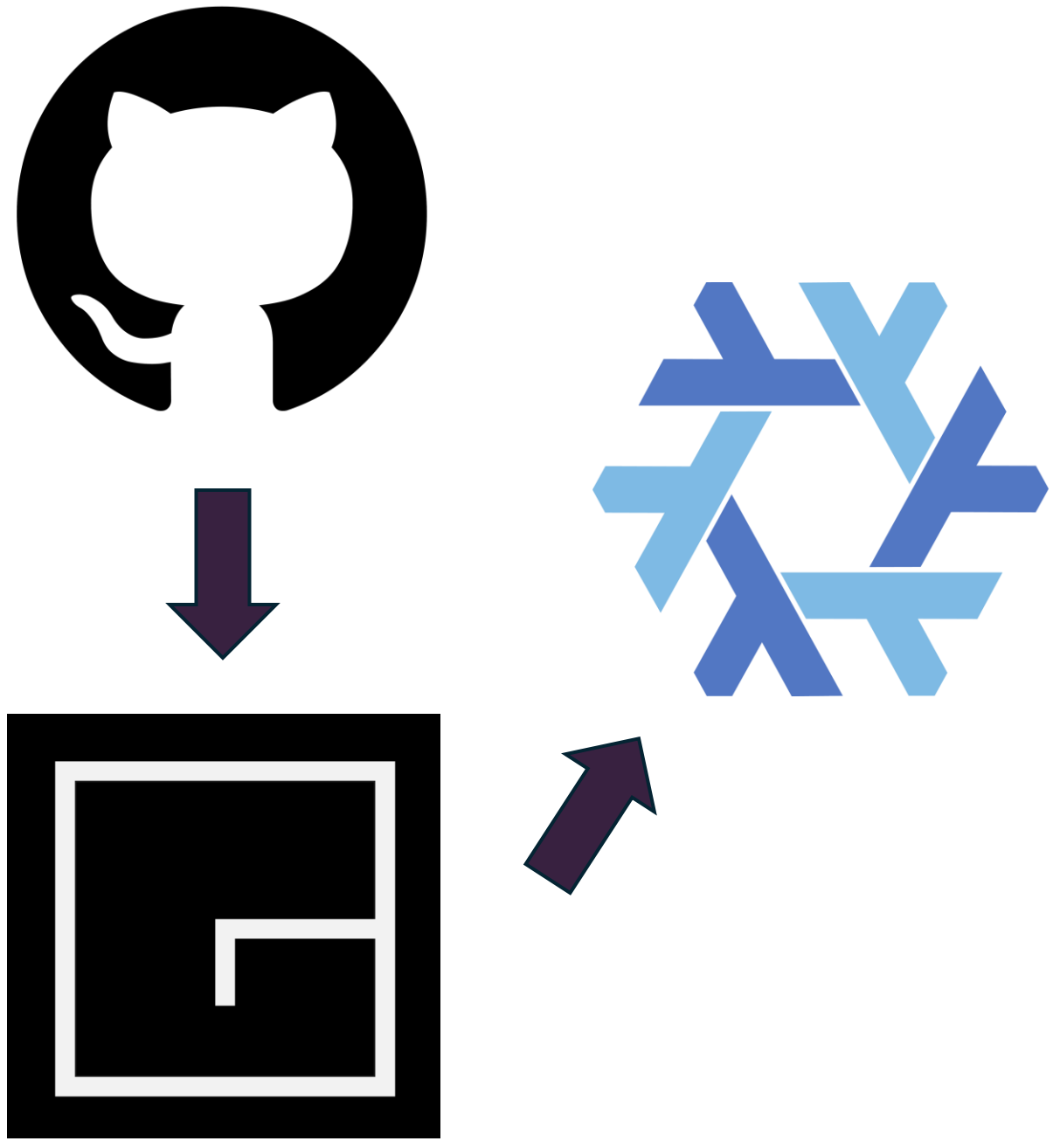
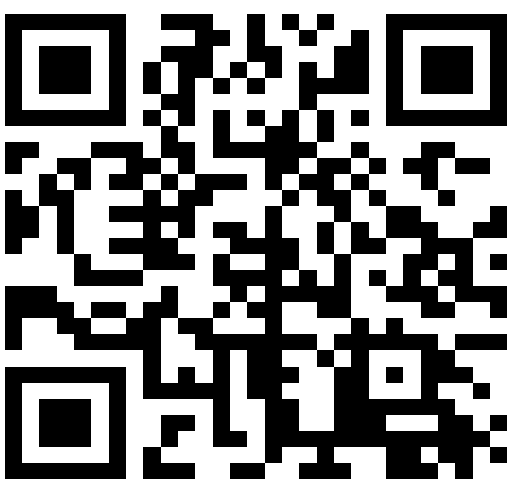


Aggre-Gator RSS

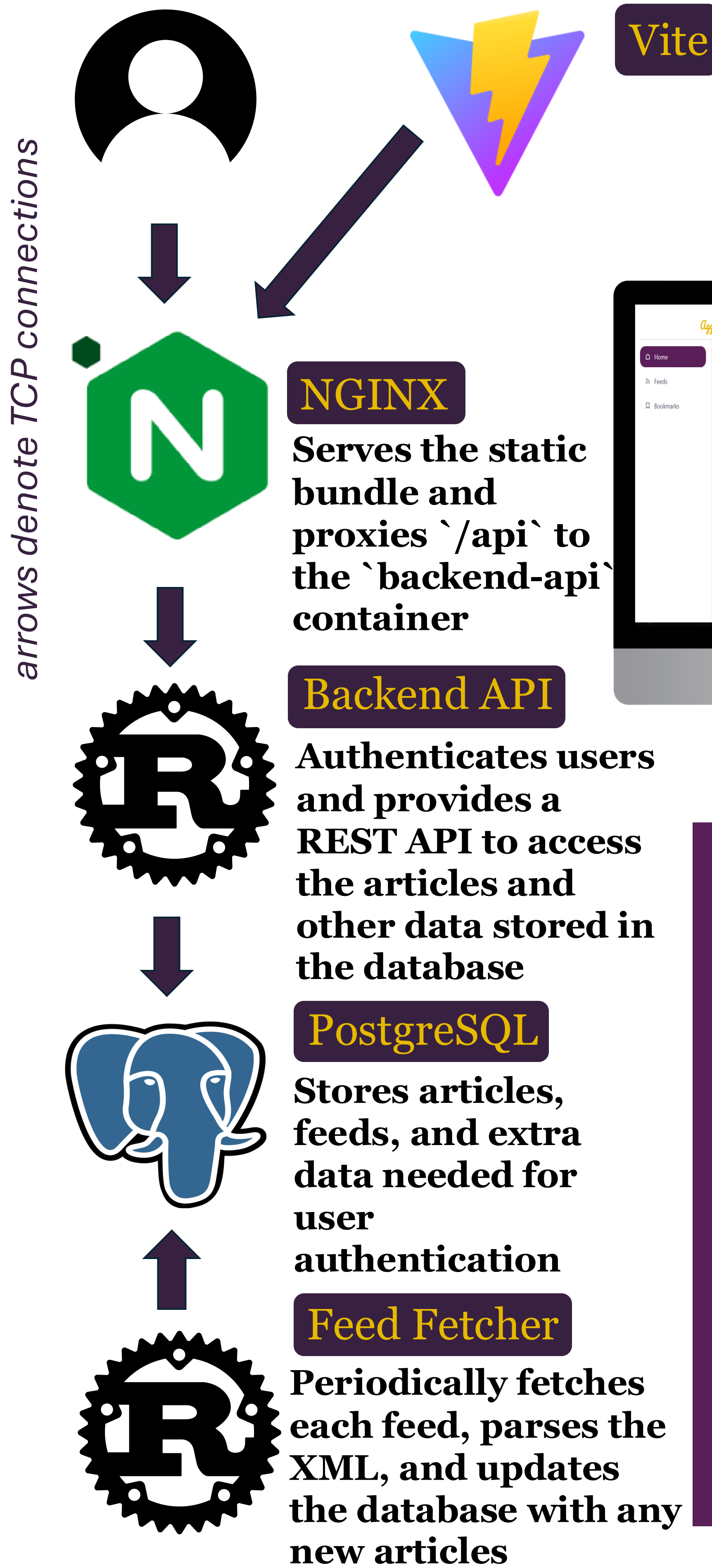
Gus Johannesen, Chris Ross,
Ellis Weaver-Kreider, Yanxi Wei



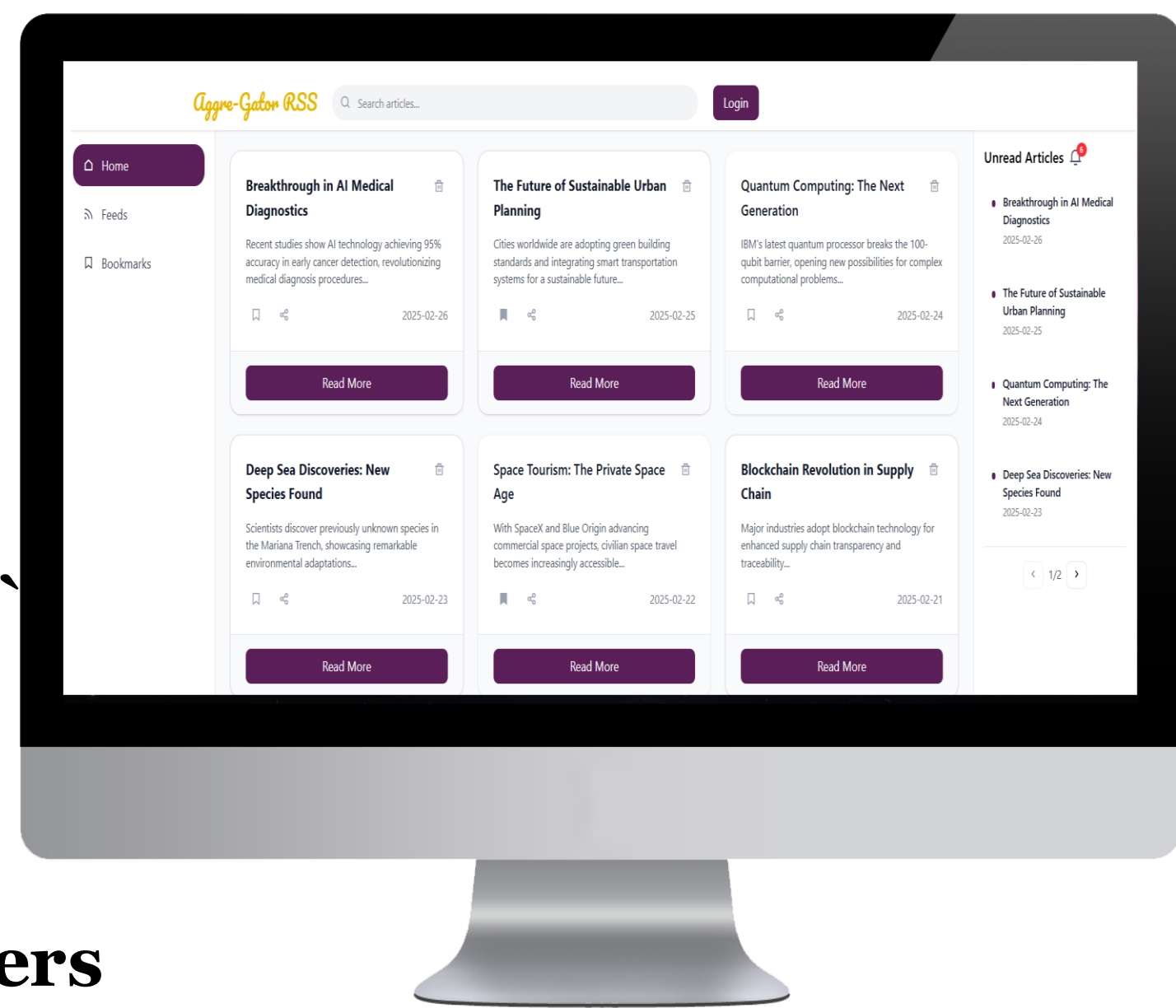
CI with Garnix


Our GitHub repository connects to Garnix CI, which automatically builds and tests using Nix expressions. Nix allows us to define tests and checks that are run hermetically in the same way builds are, ensuring they are not affected by user-specific configuration. This means that we as developers can easily run our CI checks locally. Nix's caching only rebuilds components affected by code changes, accelerating the CI process. When code reaches our deployment branch, Garnix automatically builds and deploys containers to our NixOS server.

Aggre-Gator RSS is a feed aggregator that centralizes content from multiple web sources in one convenient location. Users can subscribe to their favorite websites and receive real-time updates. Built with WCU's purple and gold theme, Aggre-Gator RSS features a minimalist interface developed using TypeScript and styled with Tailwind CSS for a clean and straightforward UX design.



Vite Compiles our Typescript, treeshakes the code, and optimizes our static assets, resulting in static files we can serve with Nginx




TechCrunch
2025-02-20

Breakthrough in AI Medical Diagnostics

Recent studies show AI technology achieving 95% accuracy in early cancer detection, revolutionizing medical diagnosis procedures...

Bookmark Share

Related Articles

Quantum Computing: The Next Generation
2025-02-24

5G Networks Transform IoT Landscape
2025-02-20

Add New Feed

Feed URL

Feed Name (Optional)

Category (Optional)

Add Feed

Deployment with NixOS

We use Nix instead of Docker to build our container images because of its focus on reproducibility and integration with NixOS. Our application components are defined declaratively in our flake.nix, allowing our NixOS server to manage container networking, volumes, and dependencies automatically. This declarative approach enables us to version control our entire runtime environment, not just application code, eliminating configuration drift between environments.

```

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" = { };
    };
  };
};

```



The code to generate our `backend-api` container