Design Overview

URStreamSight

Raymond Knorr, Avery Cameron, Noah Rowbotham

Version: 1 2021-04-08

Table of Contents

Table of Contents **2**

Backend **3**

API: 3

RBAC: 3

Database: 3

Testing: 4

CICD: 4

Frontend **4**

React: 4

AWS: **4**

Machine Learning **5**

Backend

API:

Our API is built on the NestJS framework and TypeScript. NestJS is a highly opinionated framework which provides a clear folder structure which helps with consistency and maintaining projects long term. It provides scalable, efficient and easy to read code for API routing.

NestJS uses a Controller/Service/Repository Architecture to ensure separation of concerns and the single responsibility principle. API endpoints are put inside of a controller which uses a service to get data from the repository.

Another big plus of NestJS is the fact that it uses dependency injection throughout its framework.

We investigated alternatives such as Express and Routing-Controllers, the decision is outlined in our Rationalized Choices Doc.

RBAC:

We use a signed JSON web token from AWS Cognito in our API to query the database for a user's roles. These roles are checked against the allowed roles for a given endpoint to see if the user should be given access. If the user lacks any of the acceptable roles, the API returns a 403 Forbidden HTTP error.

Database:

Our database is a MySQL database. We normalized our database to the third normal form, no cells contain multiple entries, single column primary keys and no change in one column requires changes in another.

We are using sequelize, an object-relational mapping tool (ORM), which allows you to query and access the database through object oriented code. Using sequelize allows for cleaner code with dot notation, and keeps the code-base using a consistent language.

The Schema for our database was created with the support of Prairie Robotics. Many-to-many relationships use join tables setup and managed by Sequelize. The tables are generated from Typescript files at the startup of the server if they do not currently exist.

We also created seeding data which is used in testing to ensure the reliability of endpoints.

Testing:

For our tests, we created integration tests for our API with the Javascript testing framework Jest. We are testing endpoint happy paths, permissions, invalid data type or missing parameters and existing or nonexistent records. We have achieved approximately 95% code coverage for our API. In our CICD pipeline, it runs the tests and checks that the code coverage is greater than the set threshold of 80% and will fail the pipeline if any of those steps fail.

CICD:

We are using a Continuous Integration/Continuous Deployment Pipeline. A CICD pipeline provides fast feedback to developers on every commit for whatever checks are specified in the pipeline and can automatically deploy a passing solution. Our CICD pipeline is hosted on GitHub Actions and contains steps for linting, testing and deployment.

Linting provides a way to ensure our coding standards are enforced and that code will remain consistent across all commits by any team member. We are using ESLint which provides sets of linting rules as well as Prettier, a code formatter, and shopify's typescript linting rules. This catches things like line length, unused imports or variables, naming conventions, string quotes and other standards.

Frontend

React:

Our frontend is built using the React library and TypeScript. React uses components which promote code reuse and the DRY principle. You can create a small component for something like a button or checkbox and reuse that component in multiple larger components. We are using basic React components from MaterialUI and creating reusable custom components when needed.

React uses a combination of Typescript and HTML using the TSX file format. TSX allows code resembling HTML, or actual HTML, to be embedded in regular TypeScript. Other frameworks like Vue or Angular use TypeScript embedded in HTML files, which leads to confusing syntax and the use of multiple languages within one repository.

Our frontend follows Prairie Robotics Styling and layout. This helps the product flow for users of Prairie Robotics products. Also, several of our group members had previous experience using React.

AWS:

The process starts with the Recycling truck with the camera. The truck uploads images captured during a bin tip to an S3 bucket for data storage. These uploaded images trigger a Lambda function to perform inference on EC2 and push the results to SQS. Another Lambda triggered by SQS uploads the inference events to a MySQL RDS instance. Our

main API is hosted on Lambda and triggered by API Gateway. It retrieves data from the database and sends it to our frontend. Our frontend is hosted in an S3 bucket setup for Static Website Hosting, we are using Route53 to route our DNS traffic from our custom domain to the obfuscated domain of our S3 bucket. The Municipality Analyst accesses our frontend which pulls in the data from the API for reporting and analyzing contaminants and trends.

Machine Learning

Our training set is 2000 cropped frames of the video recorded by the top camera. The frames are cropped to only have the bin in view or not in view of the hopper opening. We also have a test set of another 500 frames, plus three full videos that were used for verifying the classifier's ability to successfully segment the video into bin tips, that is, a bin tip being a segment of film from when a bin is emptied into the collection vehicle ending when the next bin is lifted. This way we have videos of waste from a single bin, which will then be used to train the contaminants classifier.

The bin detector model had four variations created. The model is based on the AlexNet model which we sourced from pytorch. We initially attempted to train the model from scratch, but found we could get better results from training a pre-trained AlexNet model trained on the ImageNet dataset. Our final fourth version of the bin detector model achieved an accuracy of 98% with only 7 false positives. This is really good, especially the low false positives as we strived to train a model that would minimize false positives as the model detecting a bin when there is not one is particularly disruptive to segmenting a video into individual bin tips.