

# Final project

# AI & machine learning

Evi Vermeêren – 3XD

## Table of contents

Links .....	3
Introduction .....	4
Use case scenarios .....	4
Data gathering.....	5
Model output integration.....	7
Challenges and considerations (general) .....	8
Challenges (while making this project) .....	8
Overview of project development process .....	9
If I had more time and knowledge.....	19
Conclusion .....	19

## Links

Github link front-end: <https://github.com/EviVermeeren/PneumoniaDetector>

Github link back-end: <https://github.com/EviVermeeren/PneumoniaAPI>

Live link to frond-end: <https://pneumonia-detector-seven.vercel.app/>

Live link to back-end: <https://pneumoniaapi.onrender.com> (you can test this in Postman, do note that render “falls asleep” when not being used, so the server needs to reboot before doing a request the first time and this can take a while. Just open the link to trigger the “awakening” before opening Postman.).

## Introduction

The pneumonia detector model holds promising potential for various applications in the realm of digital ai-driven products. Its primary function is to analyze chest x-ray images and determine the likelihood of pneumonia. This introduction outlines possible use cases, providing a glimpse into the model's versatility and its implications for creating impactful ai-driven solutions.

## Use case scenarios

### 1. Medical diagnostic assistance

- *Vision:* the pneumonia detector can serve as a diagnostic aid for medical professionals, assisting in the early detection of pneumonia through rapid analysis of chest x-ray images.
- *Implementation:* integrated into existing medical imaging systems, the model can provide quick assessments, helping healthcare providers prioritize cases and allocate resources efficiently.

### 2. Telemedicine platforms

- *Vision:* telemedicine platforms can leverage the model to support remote diagnostics. Patients can upload chest x-ray images, and the system, powered by the pneumonia detector, can provide preliminary assessments before a virtual consultation with a healthcare professional.
- *Implementation:* the model becomes an integral part of a telemedicine application, enhancing its diagnostic capabilities and improving the overall efficiency of virtual healthcare services.

## Data gathering

Gathering data for the pneumonia detector involves obtaining a diverse set of chest x-ray images. Collaborations with healthcare institutions, clinics, and medical imaging repositories can facilitate the collection of a comprehensive dataset. Ensuring diversity in terms of age, gender, and ethnicity is crucial for training a robust and unbiased model.

My model has undergone extensive training with a robust dataset of over 3000 images, including individuals with and without pneumonia sourced from <https://www.kaggle.com/>. In practical settings such as hospitals, i've incorporated a feature to facilitate continuous improvement for the ai. While my current application includes a simulation of this process, the backend development is not yet implemented to retrain the model based on doctor feedback. The envisioned functionality involves doctors pressing a 'is this correct?' button, enabling the model to enhance itself by incorporating feedback and utilizing new x-ray images for further training.

I'm currently not able to retrain the model with the uploaded pictures and the given feedback, but i did some research on how this could be possible and applied what I could in my code.

Retraining a machine learning model automatically based on user feedback and new data involves a complex process. This requires a server-side backend, a database to store the labeled data, and a mechanism to trigger the retraining process. The retraining process would involve updating the model with the new data and feedback.

Here's a simplified explanation of the steps involved:

### 1. Backend server (I did this):

- Set up a backend server (using a framework like node.js).
- Implement endpoints for uploading images and submitting feedback.

### 2. Database (I did this):

- Use a database (e.g., mongodb, mysql) to store image data and corresponding feedback labels.

### 3. Feedback submission (I did this):

- When a user submits feedback along with an image, store the image and feedback in the database.

### 4. Trigger retraining (I did not do this):

- Implement a mechanism to monitor the database for new feedback and images.
- When new data is added, trigger a retraining process.

### 5. Retraining process (I did not do this):

- Retrieve all labeled data from the database.
- Train a new model using the existing model architecture and the new labeled data.
- Save the new model.

## 6. Update frontend (I did not do this):

- Once the retraining is complete, update the frontend to use the new model.

Here's a very basic example using a fictional backend and database. Note that this is a conceptual overview.

I asked chatgpt for help, so i did not write all of this code myself. We did learn how to work with node.js and express during our course of development 5, so i do understand everything that happens in this code snippet.

In my actual code (see the steps below), I did setup a Node.js server with a database using MongoDB to collect the image upload and feedback submission.

```
// backend (node.js and express example)
Const express = require("express");
Const bodyparser = require("body-parser");
Const model = require("./your_model_module"); // your machine learning model
module
Const database = require("./your_database_module"); // your database module

Const app = express();
App.use(bodyparser.json());

// endpoint for submitting feedback and image
App.post("/submitfeedback", async (req, res) => {
  const { image, feedback } = req.body;

  // store image and feedback in the database
  database.savedata(image, feedback);

  // trigger retraining (this is a simplified example)
  const labeleddata = database.getalldata();
  const updatedmodel = await model.retrain(labeleddata);
  model.savemodel(updatedmodel);

  res.json({ message: "feedback submitted, model retrained." });
});

// your actual server setup goes here

// your database module (simplified example)
Const data = [];

Module.exports = {
  savedata: (image, feedback) => {
    data.push({ image, feedback });
  },
  getalldata: () => data,
  // other database functions...
```

```

};

// your machine learning model module (simplified example)
Const trainmodel = (data) => {
  // implement your model training logic here
};

Const savemodel = (model) => {
  // implement saving the model to disk or another storage
};

Const retrain = async (labeleddata) => {
  // implement retraining logic using the existing model architecture and new
  labeled data
  const updatedmodel = trainmodel(labeleddata);
  return updatedmodel;
};

Module.exports = {
  trainmodel,
  savemodel,
  retrain,
  // other model functions...
};

```

## Model output integration

The model's output can be seamlessly integrated into a digital ai-driven product, offering valuable insights to end-users. For example:

- **Diagnostic reports**
  - The model's probability scores and classification results can be translated into user-friendly diagnostic reports for healthcare professionals, facilitating quick decision-making.
- **Patient dashboards**
  - In a telemedicine platform, patients could access a secure dashboard displaying the model's findings, promoting transparency and understanding of their health status.

## Challenges and considerations (general)

- **Ethical considerations**
  - Ensuring patient privacy and ethical use of medical data is paramount. Strict adherence to data protection regulations and obtaining informed consent are crucial elements of the implementation.
- **Continuous model improvement**
  - Regular updates to the model using feedback from healthcare professionals contribute to ongoing improvements and ensure the model remains effective in evolving scenarios.

## Challenges (while making this project)

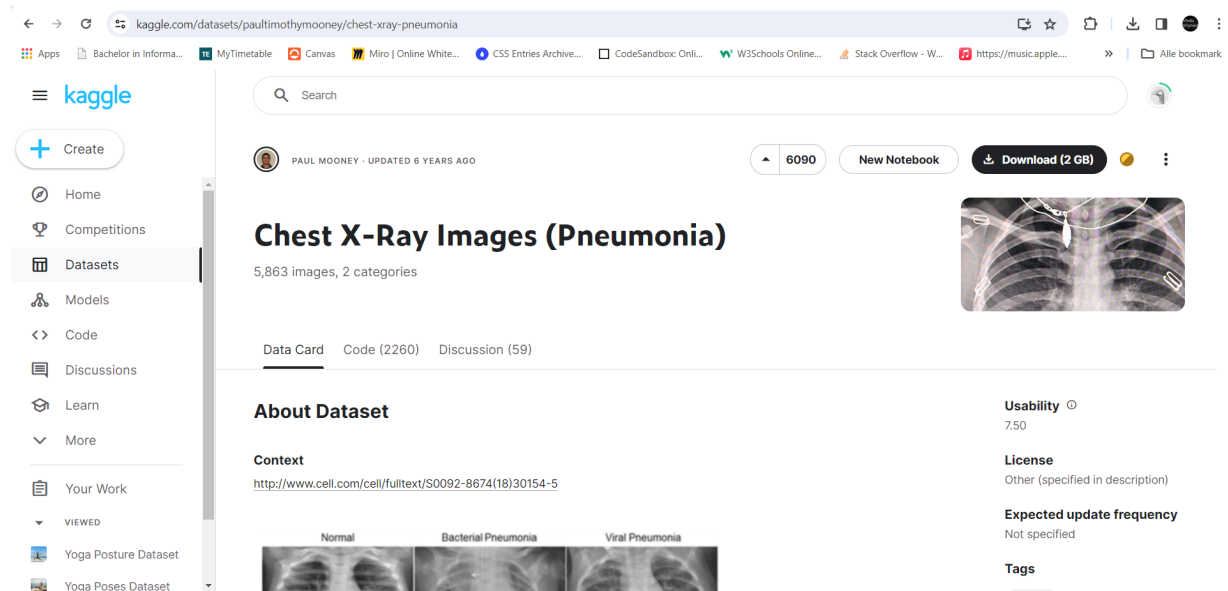
- **Continuous model improvement**
  - I tried to do this, but it was too complicated to do it automatically. What I can do, is save the images and the feedback in a database and retrain the model manually from time to time. (I should also setup something to upload the picture too, now it saves the URL but not the image. I could do this with an online cloud service.).
- **CORS**
  - While making the database with the API, it kept getting a CORS error. That is why mode: "no-cors" is enabled and my database does not get the right data from my request. I could not resolve this, even when putting corsOptions in my back-end code



# Overview of project development process

## 1. Dataset exploration

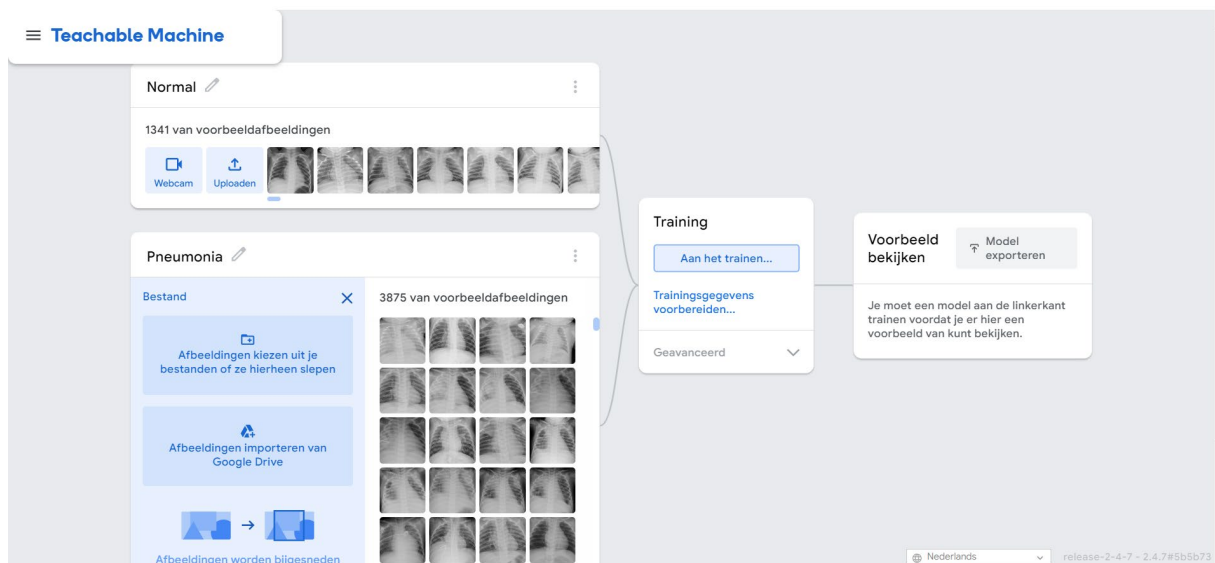
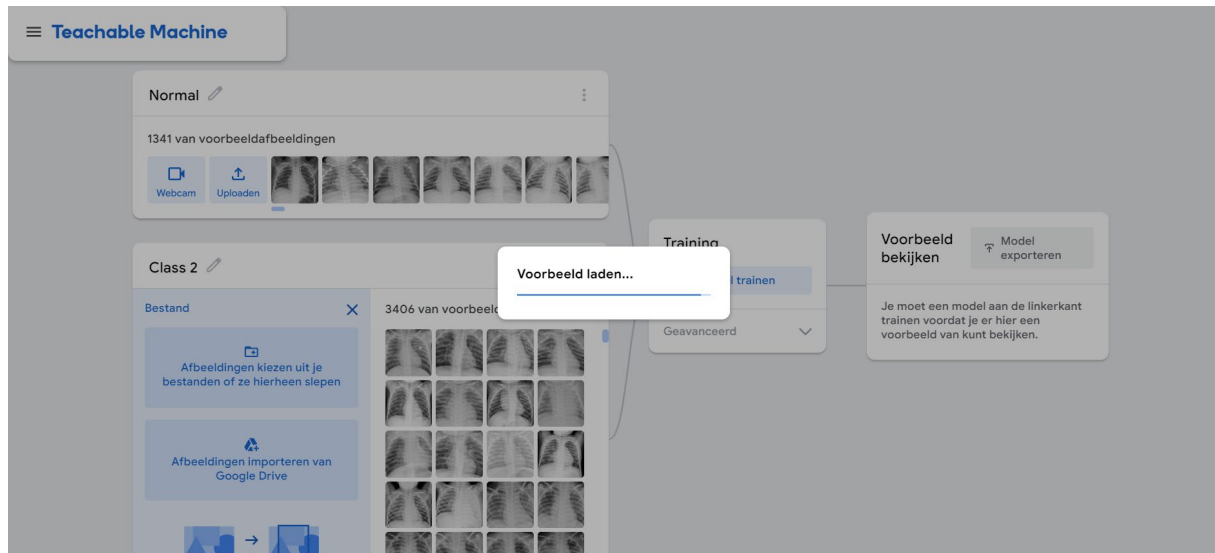
The project initiation involved an extensive search for a relevant and comprehensive dataset. After careful consideration, i selected the chest x-ray images (pneumonia) dataset from kaggle as the foundation for training the model.



The screenshot displays the Kaggle dataset page for 'Chest X-Ray Images (Pneumonia)' by Paul Mooney, updated 6 years ago. The page features a sidebar with navigation options like Home, Competitions, Datasets, Models, Code, Discussions, Learn, and More. The main content area shows the dataset title, a search bar, and a 'Download (2 GB)' button. Below the title, it states '5,863 images, 2 categories'. The 'About Dataset' section includes a 'Context' link to a research paper and three sample images labeled 'Normal', 'Bacterial Pneumonia', and 'Viral Pneumonia'. On the right, there are sections for 'Usability' (7.50), 'License' (Other), 'Expected update frequency' (Not specified), and 'Tags'.

## 2. Model training with teachable machine

The selected dataset was then fed into teachable machine, a user-friendly platform for training machine learning models. Through a series of iterations and adjustments, the model learned to identify patterns indicative of pneumonia in chest x-ray images.



### 3. Exporting the trained model

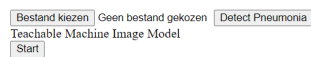
Once satisfied with the model's performance, i exported it in the form of a json file. This file encapsulated the learning and insights gained during the training process.



### 4. Html testing page

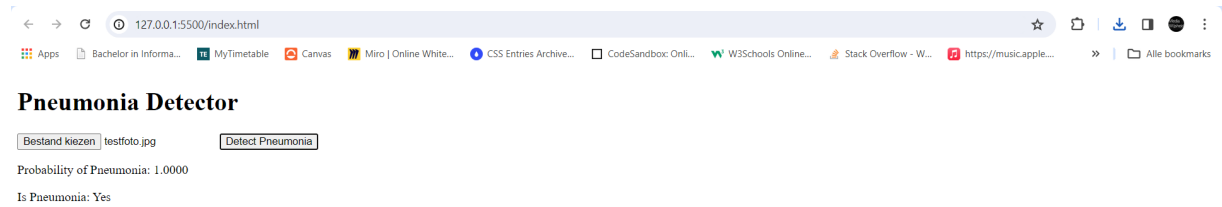
To test the trained model's functionality, i crafted a simple html page that interacted with the model using javascript. This allowed me to validate the accuracy and responsiveness of the model in a controlled environment.

#### Pneumonia detector



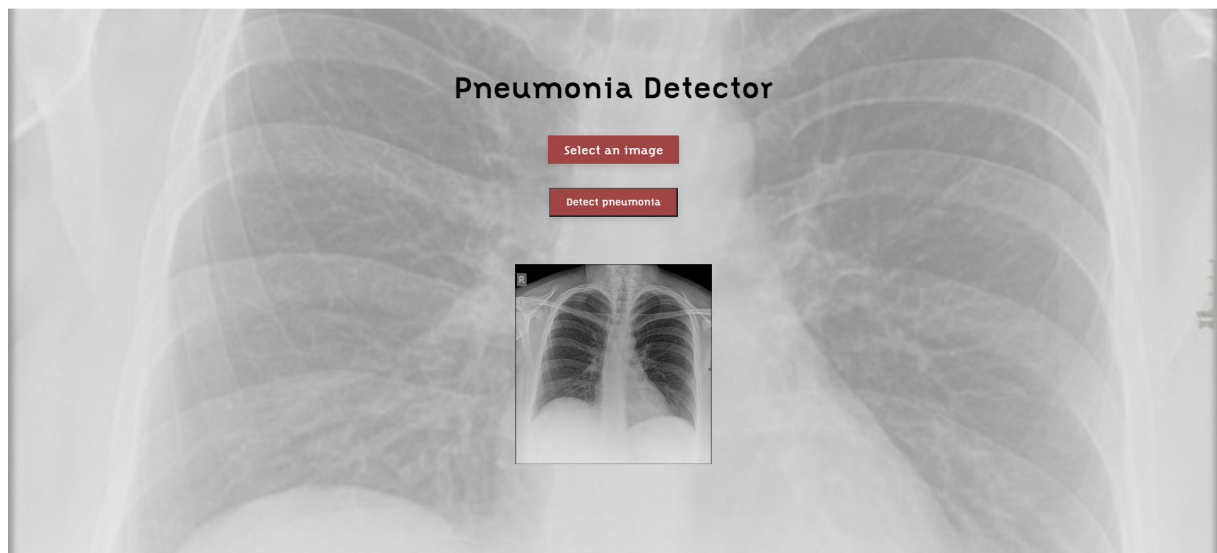
## 5. Image uploading implementation

Acknowledging the need for practicality and user-friendliness, i shifted the focus from camera-based input to image uploading. This modification opened avenues for real-world applications, making it easier for users to analyze chest x-ray images of their choice.



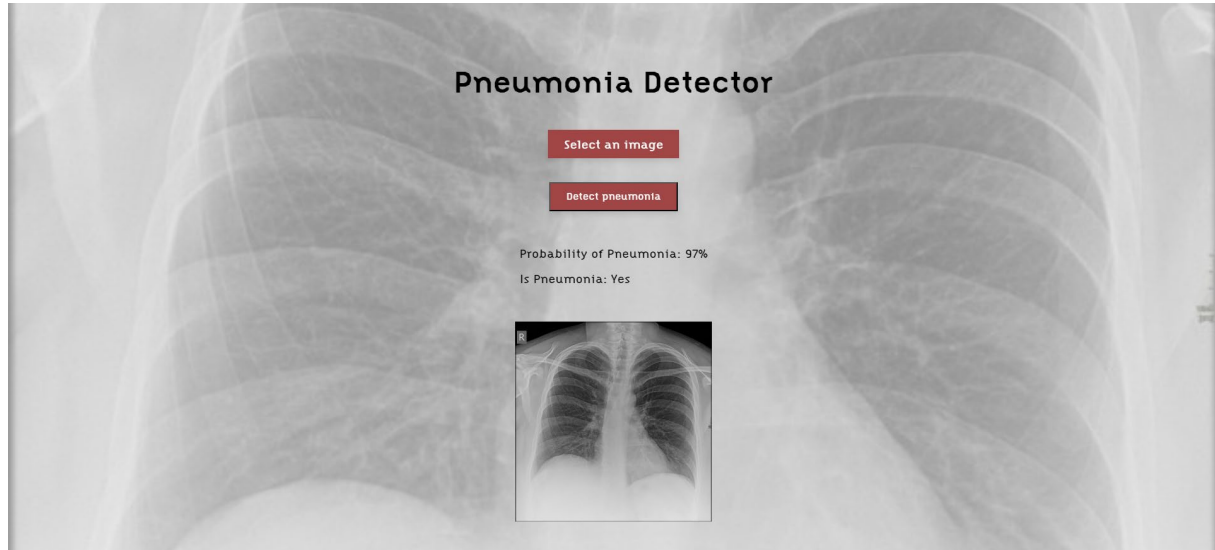
## 6. Image display enhancement

Building on the image uploading feature, i implemented a mechanism to display the uploaded image on the interface. This addition facilitated a more intuitive and informative user experience during the detection process.



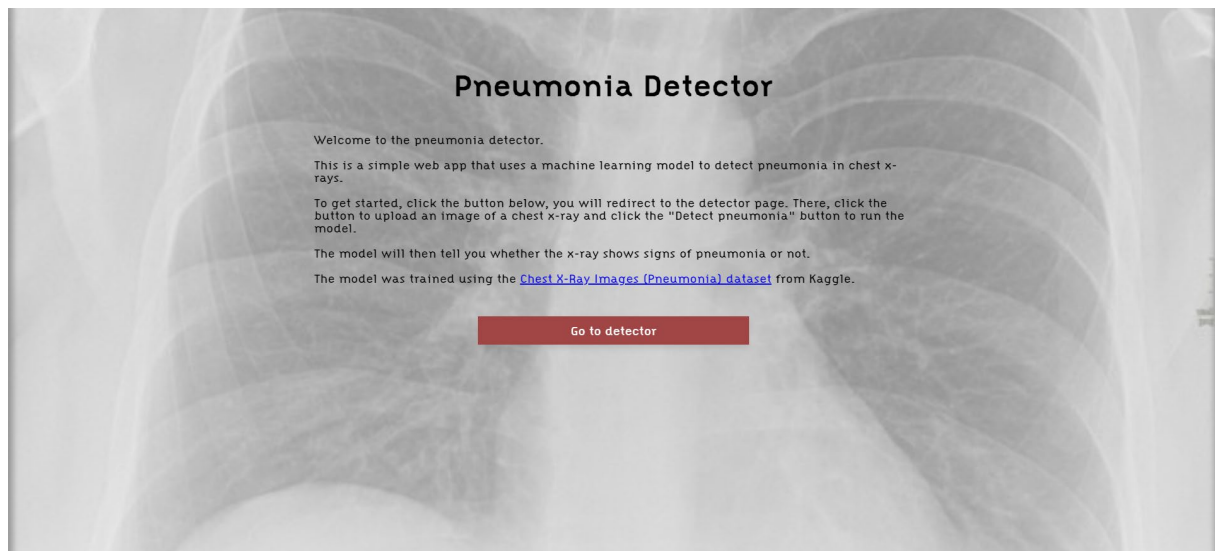
## 7. Styling for user interface

Recognizing the importance of aesthetics and user engagement, I dedicated time to refining the visual elements of the interface. Styling improvements, including fonts, colors, and layout, were incorporated to create a cohesive and appealing design.



## 8. Introduction landing page

To provide context and guidance to users, I developed a landing page with explanatory content. This page introduces the pneumonia detector, its purpose, and offers a seamless transition to the detection interface.

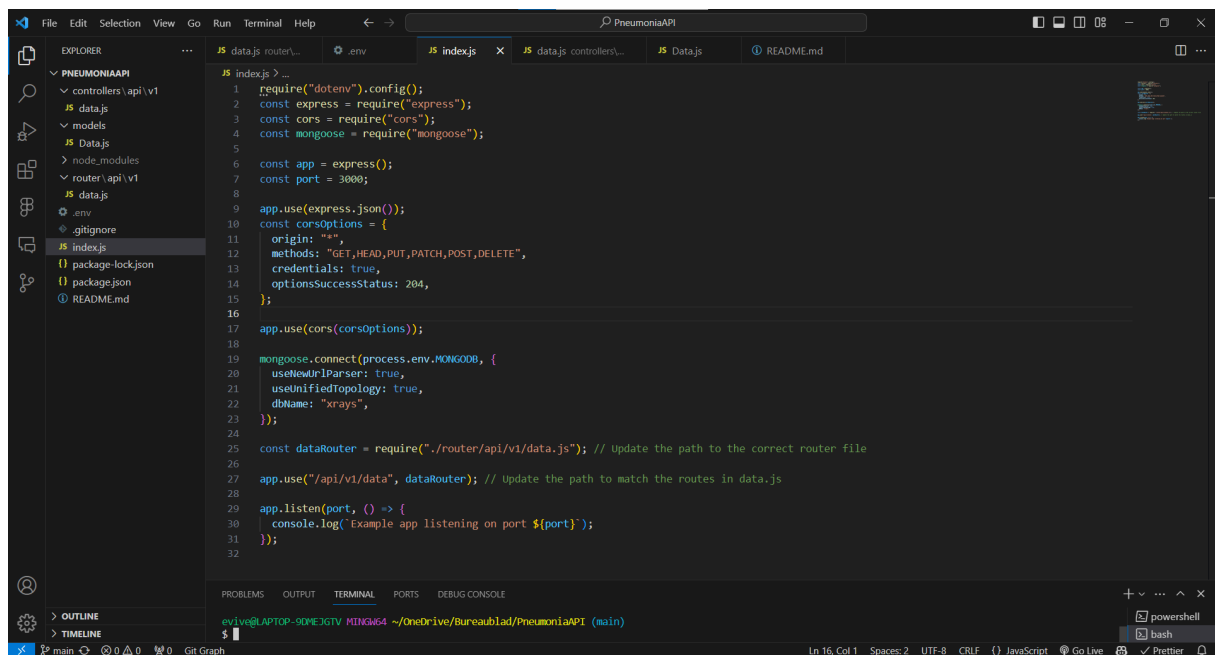


## 9. Add Node.js back-end with API endpoint

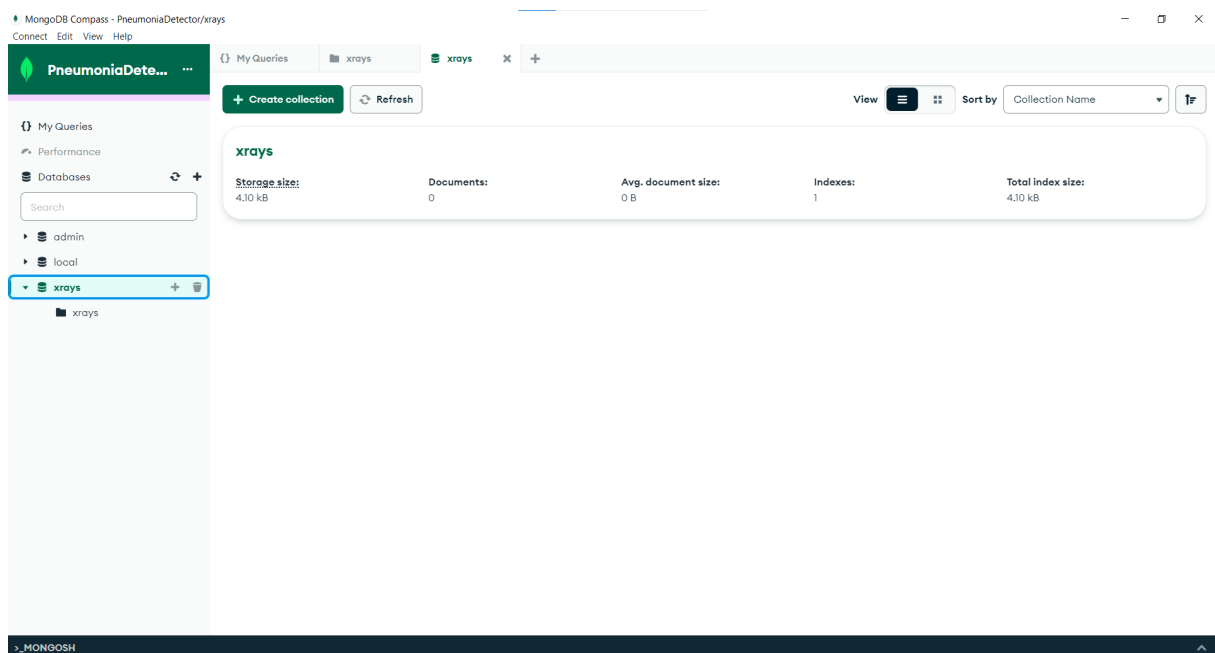
To facilitate the automatic updating of my trained model with new data, including uploaded images and feedback input, I've established a Node.js backend with API endpoints. This infrastructure incorporates controllers, models, routers, and an index.js file to efficiently manage the flow of information.

The setup allows for seamless interaction with the API. Specifically, a POST-request can be made when a user submits an image for pneumonia detection and provides feedback on the AI's accuracy. This data is then stored in a database, forming a repository that becomes instrumental for ongoing model improvement through retraining.

Additionally, a GET API is implemented to retrieve all relevant data from the database.



```
1 require("dotenv").config();
2 const express = require("express");
3 const cors = require("cors");
4 const mongoose = require("mongoose");
5
6 const app = express();
7 const port = 3000;
8
9 app.use(express.json());
10 const corsOptions = {
11   origin: "*",
12   methods: "GET,HEAD,PUT,PATCH,POST,DELETE",
13   credentials: true,
14   optionsSuccessStatus: 204,
15 };
16
17 app.use(cors(corsOptions));
18
19 mongoose.connect(process.env.MONGODB, {
20   useNewUrlParser: true,
21   useUnifiedTopology: true,
22   dbName: "xrays",
23 });
24
25 const dataRouter = require("../router/api/v1/data.js"); // Update the path to the correct router file
26
27 app.use("/api/v1/data", dataRouter); // Update the path to match the routes in data.js
28
29 app.listen(port, () => {
30   console.log(`Example app listening on port ${port}`);
31 });
32
```



## 10. Handle image upload and feedback input and sending it to the database to collect more data to retrain the model automatically

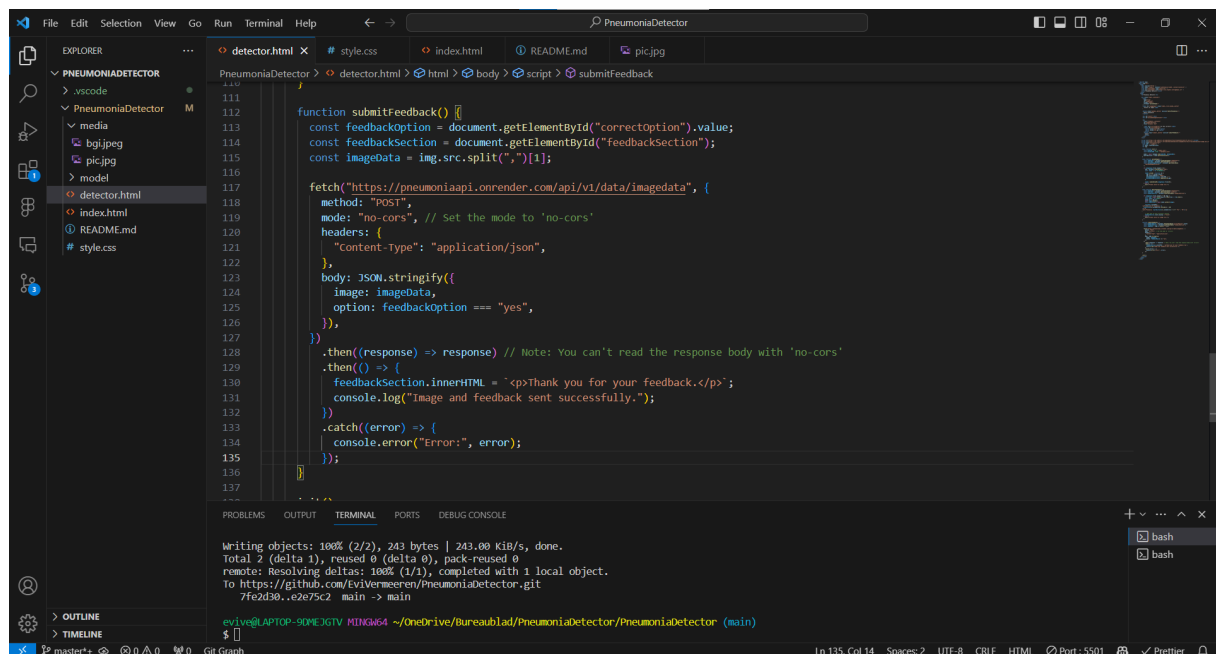
In the frontend implementation, I developed a function triggered upon pressing the submit button. This function utilizes a fetch request to execute a POST-request through my API, collecting data from both the upload and the feedback form, and subsequently storing it in the database. While the current setup functions effectively, there are aspects I would enhance given more time.

Ideally, I intended to modify the image upload process to utilize a Cloud Storage solution instead of solely obtaining the image URL. Unfortunately, time constraints limited the implementation of this improvement in the current version.

During the testing phase, I verified the functionality of my APIs using Postman, where they demonstrated successful operations. However, upon integration into the application, I encountered a CORS (Cross-Origin Resource Sharing) problem. Despite attempting to address this issue through CORS headers, the problem persisted, and the reason remains unclear.

As a temporary workaround, I employed the 'no-cors' mode. Although this mitigated the CORS problem, it led to an unintended consequence—data not being stored correctly in the database. Notably, the console logs indicated successful transmission of data to the database.

Given additional time, my priority would be to resolve the CORS issue comprehensively, ensuring secure and efficient communication between the frontend and backend. This would contribute to a more robust and seamless operation of the application.



```
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137

function submitFeedback() {
  const feedbackOption = document.getElementById("correctoption").value;
  const feedbackSection = document.getElementById("feedbackSection");
  const imageData = img.src.split(",")[1];

  fetch("https://pneumoniaapi.onrender.com/api/v1/data/imagedata", {
    method: "POST",
    mode: "no-cors", // Set the mode to 'no-cors'
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({
      image: imageData,
      option: feedbackOption === "yes",
    }),
  })
    .then((response) => response) // Note: You can't read the response body with 'no-cors'
    .then(() => {
      feedbackSection.innerHTML = `<p>Thank you for your feedback.</p>`;
      console.log("Image and feedback sent successfully.");
    })
    .catch((error) => {
      console.error("Error:", error);
    });
}
```

```
Writing objects: 100% (2/2), 243 bytes | 243.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/EviVermeeren/pneumoniadetector.git
7fe2d30..e2e75c2 main -> main
evive@LAPTOP-S0NEJGTV MINGW64 ~/OneDrive/Bureau/ablad/PneumoniaDetector/PneumoniaDetector (main)
$
```

Postman interface showing a REST client request to `https://pneumoniaapi.onrender.com/api/v1/data/imagedata` via POST. The request body is a JSON object:

```
1 {
2   "image": "base64_encoded_image_data",
3   "option": "yes"
4 }
5
```

The response status is 200 OK. The response body is a JSON object:

```
1 {
2   "status": "success",
3   "message": "POSTING new image and boolean data",
4   "data": {
5     "savedData": {
6       "image": "base64_encoded_image_data",
7       "option": "yes",
8       "id": "6595b6d8f46f4e18dbc1839d",
9       "v": 0
10    }
11  }
12 }
```

Web application interface titled "Pneumonia Detector". It features a "Select an image" button and a "Detect pneumonia" button. Below the buttons, it displays "Probability of Pneumonia: 100%" and "Is Pneumonia: Yes". A small image of a chest X-ray is shown. At the bottom, it says "Thank you for your feedback."

The browser's console shows the message: "Image and feedback sent successfully." with a link to `detector.html:131`.



MongoDB Compass - PneumoniaDetector/xrays

ConnectEditViewCollectionHelp

PneumoniaDete...

My Queries

Performance

Databases

Search

admin

local

xrays

xrays

xrays

xrays

xrays.xrays

01DOCUMENTSINDEXES

DocumentsAggregationsSchemaIndexesValidation

Filter

Type a query: { field: 'value' } or [Generate query](#)

ExplainResetFindOptions

ADD DATA

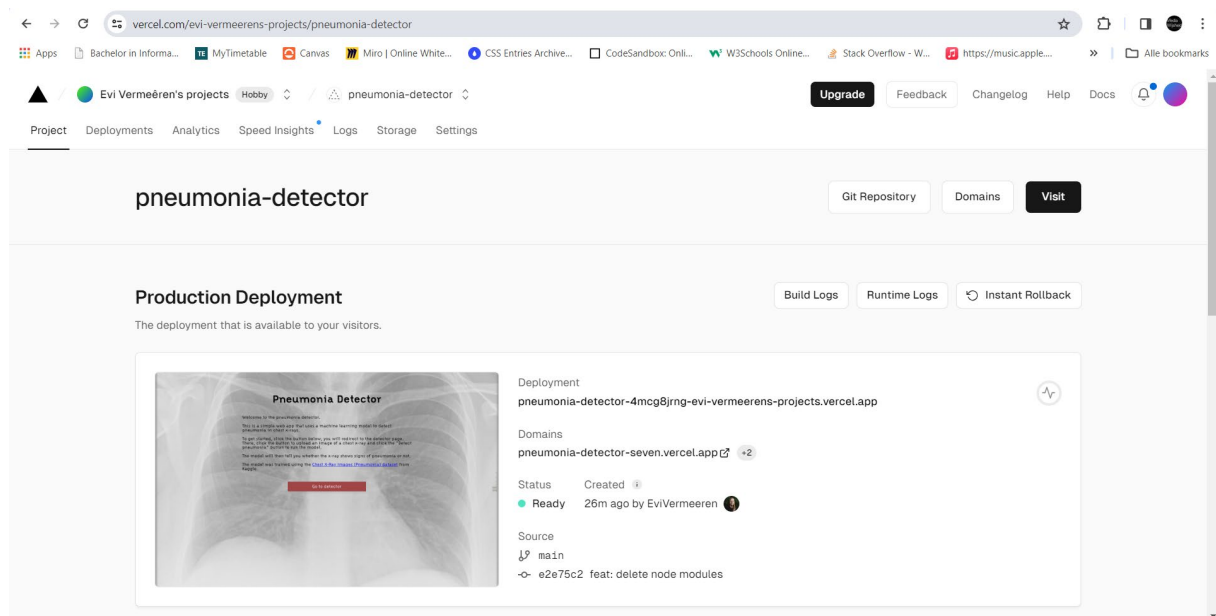
EXPORT DATA

1 - 1 of 1

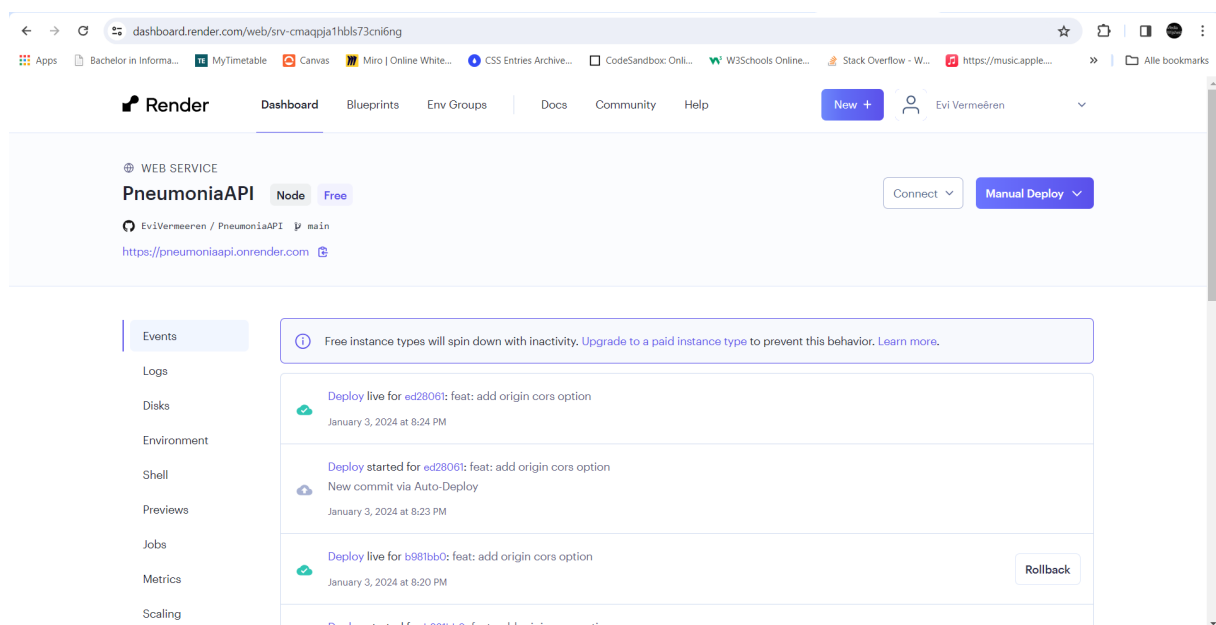
```
{ "_id": ObjectId("6595b6d8f46f4e18dbc1830d"), "image": "base64_encoded_image_data", "option": "yes", "_v": 0 }
```

## 11. Getting everything online

For your convenience in testing, I've deployed the entire project online. This provides a more authentic demonstration, allowing users to actively test and engage with the application.



The screenshot shows the Vercel deployment interface for the 'pneumonia-detector' project. The page title is 'pneumonia-detector'. Below the title, there are buttons for 'Git Repository', 'Domains', and 'Visit'. The main section is titled 'Production Deployment' and includes buttons for 'Build Logs', 'Runtime Logs', and 'Instant Rollback'. A deployment card is displayed, showing a preview of the application (a chest X-ray with text overlay) and details: Deployment 'pneumonia-detector-4mcg8jrnq-evi-vermeeren-projects.vercel.app', Domains 'pneumonia-detector-seven.vercel.app', Status 'Ready', Created '26m ago by EviVermeeren', Source 'main', and Commit 'e2e75c2 feat: delete node modules'.



The screenshot shows the Render deployment interface for the 'PneumoniaAPI' project. The page title is 'PneumoniaAPI' and it includes buttons for 'Connect' and 'Manual Deploy'. The page is divided into a sidebar with navigation links (Events, Logs, Disks, Environment, Shell, Previews, Jobs, Metrics, Scaling) and a main content area. The main content area displays a list of deployment events, including 'Deploy live for ed28061: feat: add origin cors option' and 'Deploy started for ed28061: feat: add origin cors option'. A 'Rollback' button is visible next to the 'Deploy live' event.

## If I had more time and knowledge...

Given more time, I envision several enhancements to my project. First and foremost, I would focus on refining the user interface by transitioning from an online platform to a dedicated software application, ensuring a more seamless and user-friendly experience.

In terms of data handling, I would implement a more sophisticated image storage solution by utilizing cloud storage instead of merely storing image URLs. This modification aims to enhance data integrity and accessibility.

Addressing the CORS issue is a priority. With additional time, I would invest effort in thoroughly resolving this challenge, enabling secure and efficient communication between the frontend and backend components of the application.

A significant improvement would involve establishing a robust mechanism for updating the trained AI model dynamically. This would entail implementing a functional code that triggers model retraining every time a new image is uploaded with feedback. Consequently, the frontend would seamlessly integrate the updated model, ensuring the AI's continuous improvement.

Furthermore, I would introduce a reporting feature within the application. When a button with patient information is pressed, the application would generate a comprehensive report containing details such as patient information, the uploaded image, and additional insights about the identified pneumonia. This feature aims to provide a more comprehensive diagnostic overview.

Expanding on this idea, I would develop a patient dashboard to consolidate information for healthcare professionals. This dashboard would serve as a centralized platform, presenting data collected by the AI, facilitating informed decision-making during virtual consultations.

In alignment with data privacy regulations such as GDPR, I would also consider implementing features to obtain explicit patient consent. This might include a functionality where patients can grant permission for image and feedback storage, ensuring compliance with ethical considerations and privacy standards.

These proposed enhancements collectively contribute to a more refined, secure, and comprehensive AI-driven application, catering to both user experience and ethical considerations.

## Conclusion

In summary, I believe I've demonstrated my proficiency in working with Google's Teachable Machine, showcasing my ability to develop a machine learning model and integrate it into a practical application.

The pneumonia detector, in particular, holds substantial real-world potential and could prove highly valuable in practical applications.

I'm optimistic that this effort meets the expectations for a top-notch evaluation, aiming to conclude my IMD journey on a high note. 🤓👍