Easy A
CS 422 Project 1 Team Git Good

Raj Gill, Giovanni Mendoza Celestino, Jason Webster, Karma Woeser

https://github.com/kwoeser/Project-1-EasyA

Table of Contents

4. Engineering Diagram

4.1 Data Upload Process
4.2 User Search & Filtering
4.3 Data Visualization Flow
4.4 Admin Data Management
4.5 System Maintenance

---

5. Engineering Flow Guide

5.1 Backend Database Management (MongoDB & Flask)
5.2 Data Processing (DataLoader Class & Faculty Merging)
5.3 Web Scraper (Faculty Data Collection)
5.4 Flask Web Routes (User & Admin Pages)
5.5 Data Visualization (Grade Distribution Graphs)
5.6 Testing & Quality Assurance (Unit Tests with `unittest`)
5.7 Deployment (Docker Setup & Configuration)

---

6. Engineering Flow Visual

1. Project Proposal

a. Member Contact & Team Name

i. Rajveer Gill, rajveerg@uoregon.edu, (925) 997-8995
ii. Karma Woeser, karmawoeser1@gmail.com
iii. Jason Webster, jasonfwebster69@gmail.com
iv. Giovanni Mendoza, giomendoza5202003@gmail.com

b. Initial Description

This is the initial project plan that describes the overall strategy for the project, the roles and responsibilities of team members, the technology to be used, the timeline, and the resources needed to complete the project.

We have organized the team by breaking us up into 2 smaller teams. The teams for this project will be Jason and Giovanni on the frontend and Raj and Karma on backend and database integration. Each week a new member will take the team leader position and check in with all other members and do a status check of where each member is currently at in their progression (this will give each member a feel for team leader and responsibility). The current acting team leader will be the one in charge of the project and will have the final decision in cases where consensus cannot be reached. The team will meet in-person or zoom for the weekly meetings and will be led by the current week's project team leader and the progress will be monitored frequently via google docs, team chat, and face to face meetings.

To achieve the goals and objectives of the project, the project will be broken down into a number of tasks and these will be arranged in a structured work breakdown schedule. The milestones are; understanding and completing the requirements for the project, creating the GitHub repository, pre-processing of gradedata.js to populate MongoDB, setting up the Flask project structure, creating API endpoints, creating UI mockup and forms using WTForms, developing data visualization with Chart.js, integrating the front end with the back end, developing tools for admin to update data, testing and debugging, and deploying the application. Testing will be conducted during each stage and will include both unit and integration tests. Documentation will be written during development and finalized in the last week

In the first week we will focus on documentation and preparing the initial project plan. We will set up a GitHub repository and establish branch workflow. Preprocess gradedata.js and populate MongoDB. Set up Flask project structure and implement basic API endpoints. Week two milestones are as follows: Create initial UI mockups and forms using WTForms. Develop data visualization using Chart.js. Integrate frontend with backend. Develop admin tools for data updates. Week three list is to conduct testing and debugging. Deploy the application. Finalize documentation and submission materials. Prepare and rehearse the project presentation.

The progress will be monitored through a shared Google Sheet where tasks, hours worked and completion status will be entered. Each week the team will meet to check on the milestones and talk about any problems that have arisen; also, peer reviews will be employed to ensure that the work that is being done is of high quality and is consistent.

c. Initial Project Plan

1. Management Plan

Team Organization:

- **Jason**: Frontend Developer
- **Giovanni**: Frontend Developer
- **Raj**: Backend Developer
- **Karma**: Backend Developer
- **Shared Responsibilities**:
    - All team members will contribute to project management and documentation tasks.
    - Everyone will participate in regular team meetings and reviews to ensure progress.

Work Division:

- **Frontend**:
    - Design and implement UI components (Jason and Giovanni).
    - Develop WTForms and integrate graphs (Jason and Giovanni).
- **Backend**:
    - Set up API endpoints and database integration (Raj and Karma).
    - Implement data processing and admin tools (Raj and Karma).

Decision-Making:

- Decisions will be made collaboratively during team meetings. If consensus cannot be reached, the project leader (rotating weekly) will make the final decision.

Communication:

- **Tools**: Group chat for daily communication, Zoom or in-person for weekly meetings.
- **Frequency**: Updates via chat as necessary and weekly meetings for progress review.

2. Work Breakdown Schedule and Project Schedule

Milestones:

1. Finalize project requirements and SRS documentation.
2. Preprocess gradedata.js and populate MongoDB.
3. Set up Flask project structure and implement basic API endpoints.
4. Create initial UI mockups and forms using WTForms.
5. Develop data visualization using Chart.js.
6. Integrate frontend with backend.
7. Develop admin tools for data updates.
8. Conduct testing and debugging.
9. Deploy application
10. Finalize Documentation and submission materials
11. Prepare project presentation.

Project Schedule:

- **Week 1**: Milestones 1-3
- **Week 2**: Milestones 4-8
- **Week 3**: Milestones 8-11

3. Monitoring and Reporting

- **Tracking Progress**:
  - Use a shared Google Sheet to log tasks, hours worked, and completion status.
  - Each member updates the sheet daily.
- **Progress Reviews**:
  - Weekly team meetings to review milestones and resolve issues.
  - Peer reviews of each other's work.

4. Build Plan

Sequence of Steps:

1. Finalize requirements and set up the environment.
2. Develop individual components (API endpoints, forms, data processing).
3. Integrate components and ensure smooth functionality.
4. Test and debug the application.
5. Deploy and prepare for submission.

Rationale:

- Breaking the system into frontend and backend components allows parallel development, maximizing efficiency.
- Early focus on backend ensures data availability for frontend integration.

5. Risks and Mitigation:

- **Risk**: Delays in backend development.
    - **Mitigation**: Start with basic API endpoints and iterate.
- **Risk**: Integration issues.
    - **Mitigation**: Conduct regular integration tests during development.

6. Software Design Specification (SDS)

i. Product Description

The system will allow students to compare grade distributions of instructors and classes at the University of Oregon. It includes:

- A web interface for browsing data.
- Interactive graphs for visualizing grade distributions.
- Admin tools for updating datasets.

ii. Overall Design Description

System Components:

1. **Frontend**:
    - User Interface: Flask templates, WTForms, and graph rendering.
    - Interaction with backend via APIs.
2. **Backend**:
    - Flask application with routes for data retrieval and processing.
    - MongoDB for data storage.
3. **Database**:
    - MongoDB stores instructor and grade data.
4. **Admin Tools**:
    - CLI or web-based tools for data updates.

System Structure:

- **Frontend**: Communicates with the backend to fetch and display data.
- **Backend**: Processes data requests and interacts with the database.
- **Database**: Stores processed grade and instructor data.

iii. Subsystem Models

Static Model:

- **Database Schema**:
    - Collections: instructors, classes, grades, department
- **APIs**:
    - Endpoints: /get-teacher, /get-grades, /filter-grades, /update-data.

Dynamic Model:

- **User Flow**:
    - User selects filters on the frontend.
    - Filters sent to the backend via API.
    - Backend retrieves and processes data from MongoDB.
    - Processed data returned to frontend for display as graphs.

iv. Design Rationale

- **Flask**: Lightweight and suitable for rapid development.
- **MongoDB**: Flexible schema for handling structured grade data.
- **WTForms**: Simplifies form validation and handling.
- **Chart.js**: Enables clear and interactive data visualization.

7. List of Requirements to Complete the Project

Technologies & Libraries

1. Python 3.x
2. Flask (Web framework)
3. MongoDB (Database)
4. WTForms (Form handling)
5. Chart.js (Data visualization)
6. Flask-PyMongo (MongoDB integration)
7. HTML, CSS, JavaScript (Frontend development)
8. Bootstrap (Optional for responsive UI design)
9. AJAX for seamless page integration

Development Tools

1. PyCharm, VS Code, or another IDE
2. Git/GitHub (Version control)
3. Postman or curl (API testing)
4. Python virtual environment

Data Files

1. gradedata.js
2. Faculty Data (Wayback Machine)

System Setup

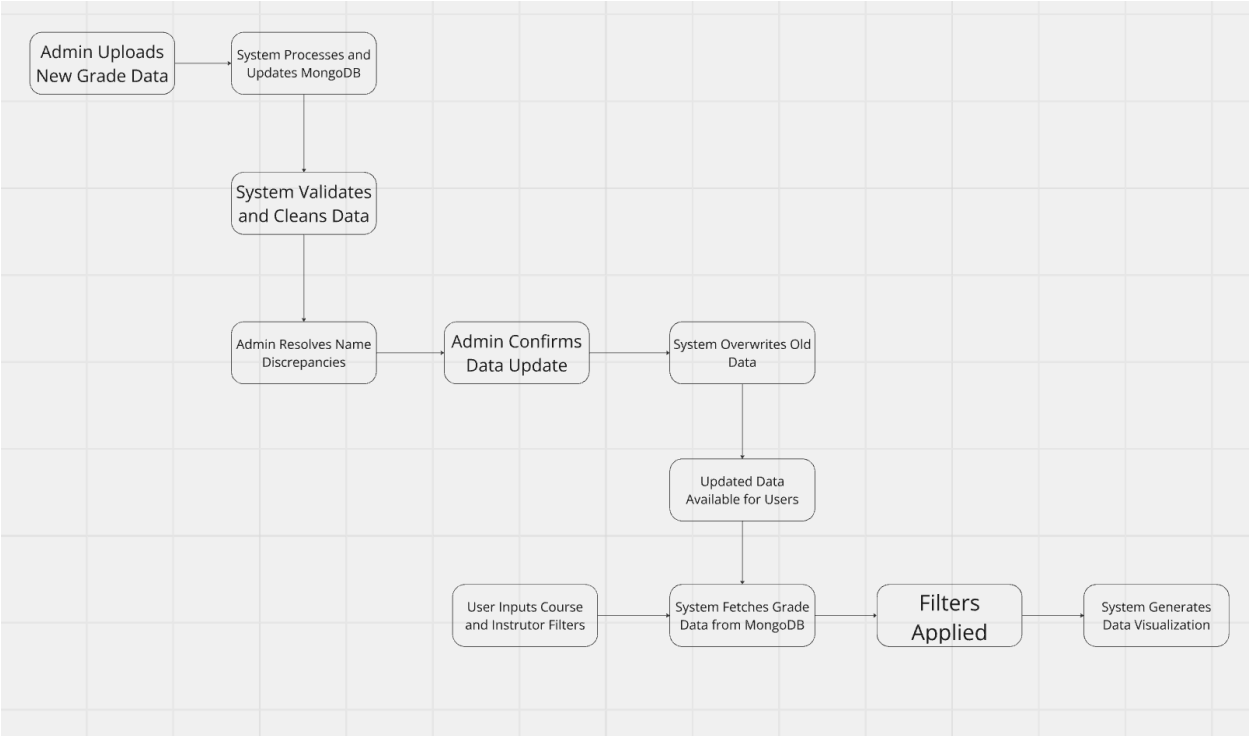
1. Local Flask development server
2. MongoDB instance
3. Deployed server environment (e.g., ix.cs.uoregon.edu)

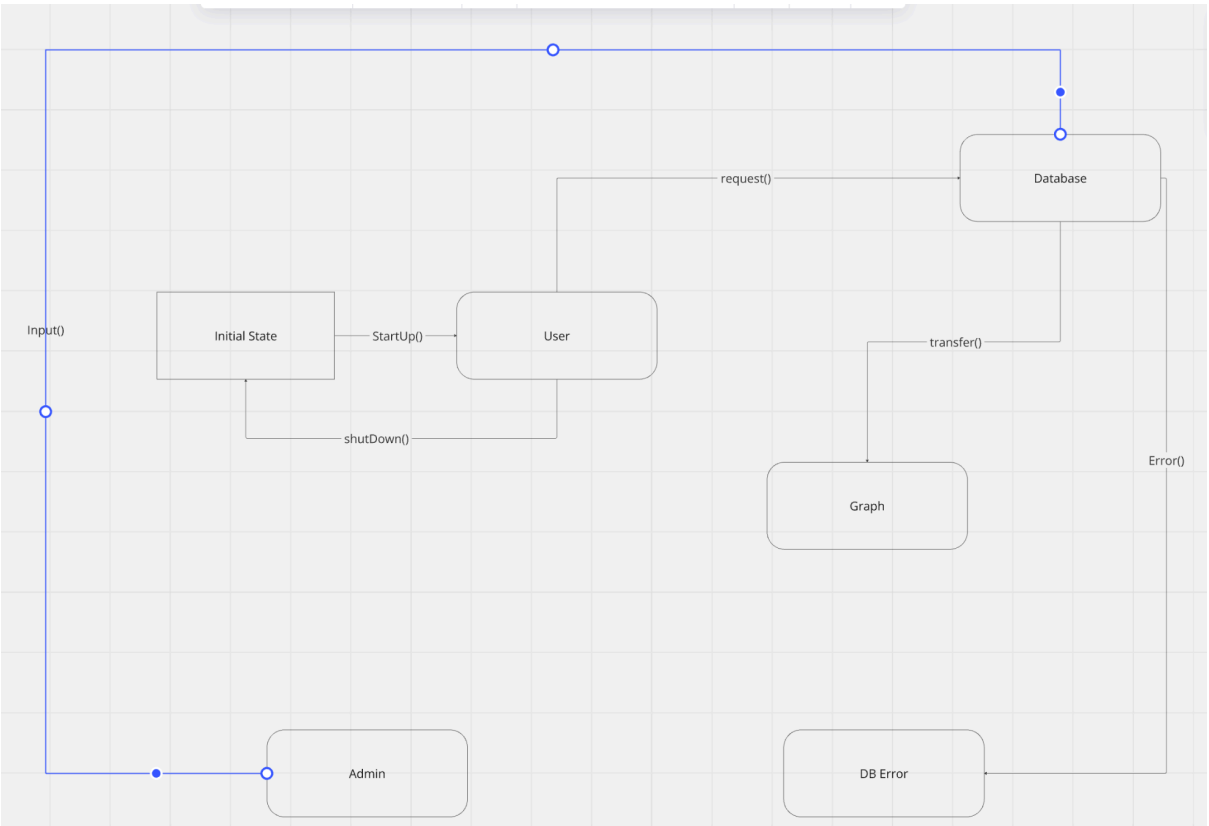
2. CheckList

EasyA_Project_Checklist

Category	Task	Status
Project Management	Assign team roles	Done
Project Management	Set up communication tools	Done
Project Setup	Finalize project requirements	Done
Project Setup	Create SRS documentation	Done
Project Setup	Set up GitHub repository	Done
Project Setup	Preprocess gradedata.js and populate MongoDB	Done
Project Setup	Set up Flask project structure	Done
Project Setup	Implement basic API endpoints	Done
Development	Create UI mockups and forms using WTForms	Done
Development	Develop data visualization	Done
Development	Integrate frontend with backend	Done
Development	Develop admin tools	Done
Testing & Deployment	Conduct unit and integration testing	Done
Testing & Deployment	Debug issues	Done
Testing & Deployment	Finalize project documentation	Done
Testing & Deployment	Prepare project presentation	Done
Frontend	Develop UI with WTForms	Done
Frontend	Implement interactive graphs	Done
Frontend	Ensure seamless API integration	Done
Backend	Set up Flask application	Done
Backend	Implement API endpoints	Done
Backend	Connect Flask backend to MongoDB	Done
Backend	Develop API endpoints	Done
Backend	Test API endpoints	Done
Database	Design MongoDB schema	Done
Database	Store instructor and grade data	Done
Admin Tools	Implement admin tools for data updates	Done
Technical Setup	Set up Flask	Done
Technical Setup	Configure MongoDB	Done
Technical Setup	Install required libraries	Done
Development Tools	Set up Git/GitHub	Done
Development Tools	Set up Docker	Done
System Setup	Set up local Flask development server	Done
System Setup	Connect to MongoDB	Done

3. Application Diagrams





4. Engineering Diagram

1. Data Upload

- **Event:** Admin uploads `gradedata.js` file.
- **Process:**
  - Parse `gradedata.js` to extract relevant data.
  - Transform and clean data to match database schema.
  - Insert processed data into MongoDB collections.
- **Outcome:**
  - Data successfully stored in the database.
  - System ready to serve user queries.

2. User Search

- **Event:** User inputs search criteria (e.g., course name, instructor).
- **Process:**
  - Receive and validate user input.
  - Query MongoDB to retrieve matching records.
  - Aggregate data to calculate statistics (e.g., average grades, distribution).
- **Outcome:**
  - Display aggregated results to the user.
  - Provide visualizations (e.g., charts, graphs) for better understanding.

3. Data Visualization

- **Event:** User requests detailed view of search results.
- **Process:**
  - Generate visual representations of data (e.g., bar charts, pie charts).
  - Highlight key metrics (e.g., grade distributions, instructor comparisons).
- **Outcome:**

    - Enhanced user insight into course and instructor grading patterns.
    - Facilitated decision-making for course selection.

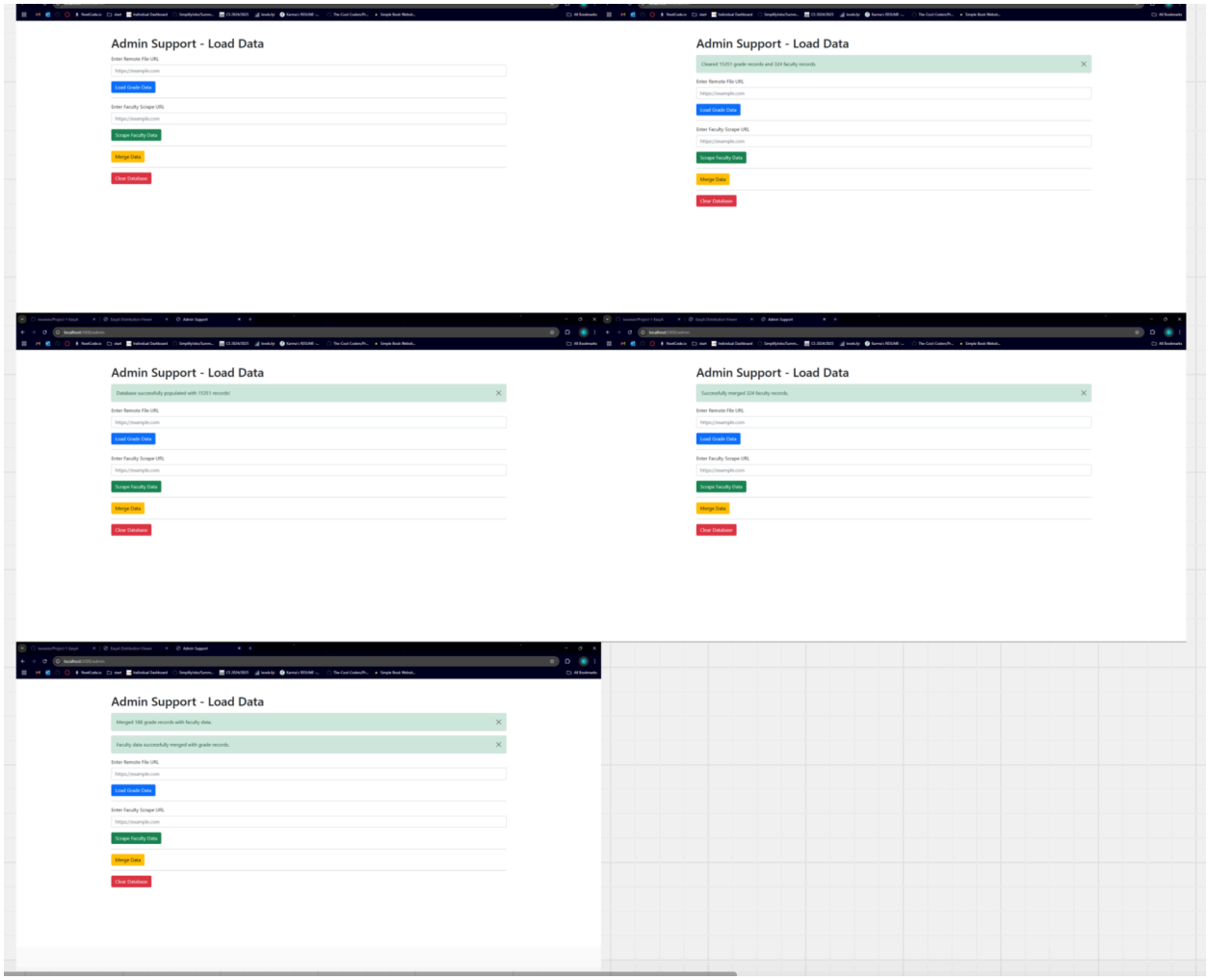
4. Admin Data Management

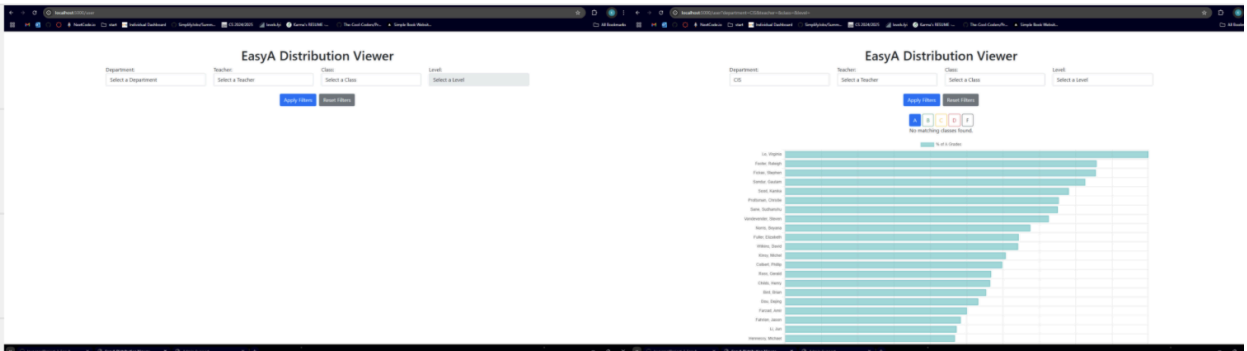
- **Event:** Admin identifies discrepancies or outdated data.
- **Process:**
    - Access admin panel to review current data entries.
    - Edit, update, or delete records as necessary.
    - Validate changes to ensure data integrity.
- **Outcome:**
    - Database reflects accurate and up-to-date information.
    - Users receive reliable data for their queries.

5. System Maintenance

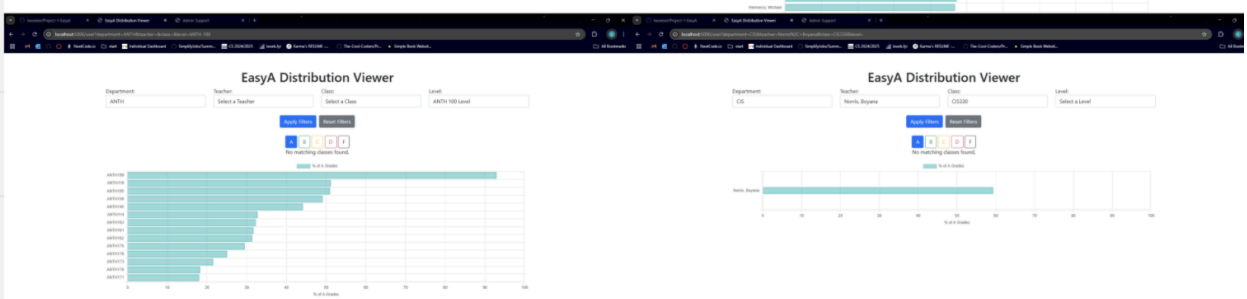
- **Event:** Scheduled maintenance or unexpected issues arise.
- **Process:**
    - Backup current database state.
    - Deploy updates or fixes to the application.
    - Monitor system performance post-deployment.
- **Outcome:**
    - System operates smoothly with minimal downtime.
    - Users experience uninterrupted access to features.

Engineering flow Visual

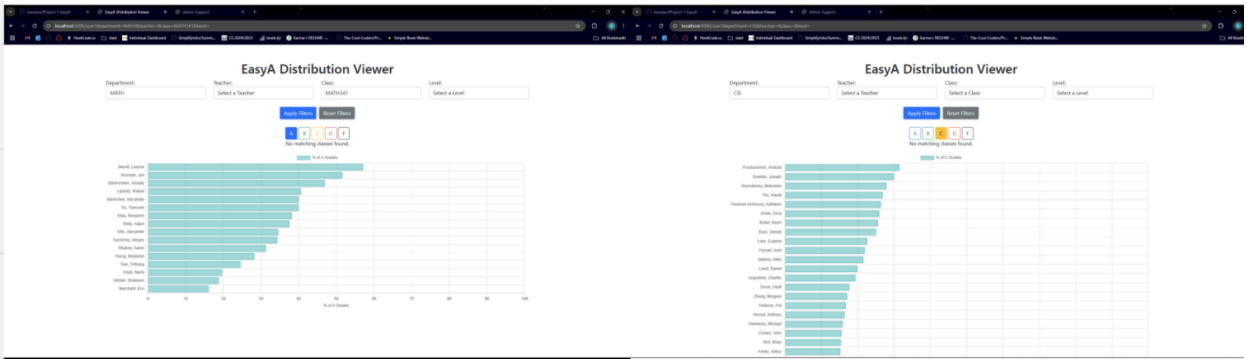



EasyA Distribution Viewer
Department:
Select a Department
Teacher:
Select a Teacher
Class:
Select a Class
Level:
Select a Level
Apply Filters
Reset Filters

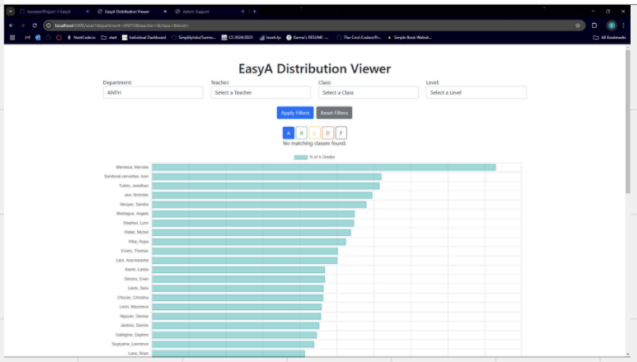
EasyA Distribution Viewer
Department:
CIS
Teacher:
Select a Teacher
Class:
Select a Class
Level:
Select a Level
Apply Filters
Reset Filters
A B C D F
No matching classes found.


EasyA Distribution Viewer
Department:
ANTH
Teacher:
Select a Teacher
Class:
Select a Class
Level:
ANTH 100 Level
Apply Filters
Reset Filters
A B C D F
No matching classes found.

EasyA Distribution Viewer
Department:
CIS
Teacher:
Norris, Boyana
Class:
CIS100
Level:
Select a Level
Apply Filters
Reset Filters
A B C D F
No matching classes found.


EasyA Distribution Viewer
Department:
MATH
Teacher:
Select a Teacher
Class:
MATH141
Level:
Select a Level
Apply Filters
Reset Filters
A B C D F
No matching classes found.

EasyA Distribution Viewer
Department:
CIS
Teacher:
Select a Teacher
Class:
Select a Class
Level:
Select a Level
Apply Filters
Reset Filters
A B C D F
No matching classes found.


EasyA Distribution Viewer
Department:
ANTH
Teacher:
Select a Teacher
Class:
Select a Class
Level:
Select a Level
Apply Filters
Reset Filters
A B C D F
No matching classes found.

5. Engineering Flow Guide

1. Backend Database Management - MongoDB

Initializing MongoDB with Flask

To manage user data and grade distributions, EasyA uses MongoDB through Flask.

Setup MongoDB Connection in Flask

```
from flask import Flask

from flask_pymongo import PyMongo

from config import Config


app = Flask(__name__)

app.config.from_object(Config)

mongo = PyMongo(app)
```

Database Structure

- `grades`: Stores grade distributions.
- `faculty`: Stores faculty information.

Creating Indexes for Faster Queries

```
@app.before_request

def create_indexes():

    if not hasattr(app, 'indexes_created'):
```

```
        mongo.db.grades.create_index("course")
        mongo.db.grades.create_index("instructor")
        mongo.db.grades.create_index([("course", 1),
("instructor", 1)])
        mongo.db.faculty.create_index("department")
        mongo.db.faculty.create_index("instructor")
        app.indexes_created = True
```

---

2. Data Processing - DataLoader Class

The `DataLoader` class is responsible for processing course and faculty data.

Loading Faculty Data into MongoDB

```
from pymongo import UpdateOne


class DataLoader:
    def __init__(self, db, departments):
        self.db = db
        self.departments = departments
```

```
    def insert_faculty_data(self, faculty_data):
        operations = [
            UpdateOne(
                {"name": faculty["name"]},
                {"$set": faculty},
                upsert=True
            ) for faculty in faculty_data
        ]
        self.db.faculty.bulk_write(operations)
```

Merging Faculty with Grades

```
def merge_faculty_with_grades(self):
    faculty_data = {doc["name"]: doc["department"] for
doc in self.db.faculty.find({})}
    updates = []

    for grade in self.db.grades.find({}):
        instructor = grade.get("instructor", "")
        department = faculty_data.get(instructor,
"Unknown")
```

```
        updates.append(UpdateOne({"_id": grade["_id"]},
{"$set": {"department": department}}))


    if updates:

        self.db.grades.bulk_write(updates)
```

---

3. Web Scraper - Faculty Data Collection

EasyA scrapes faculty data from archived university pages.

Scraping Faculty Names

```
import requests

from bs4 import BeautifulSoup


def get_faculty(url):

    response = requests.get(url)

    soup = BeautifulSoup(response.content,
"html.parser")

    faculty_list = [name.text.strip() for name in
soup.find_all("span", class_="faculty-name")]

    return faculty_list
```

Storing Scraped Data

```
import json


faculty_data =
get_faculty("https://web.archive.org/.../faculty-list")

with open("faculty_data.json", "w") as f:

    json.dump(faculty_data, f)
```

---

4. Flask Web Routes - User and Admin Pages

User Page - Filtering and Graphs

The `/user` page allows students to filter grade distributions.

Route Handling User Filters

```
@app.route("/user")

def user_page():

    department = request.args.get("department", "")

    single_class = request.args.get("class", "")

    instructor = request.args.get("teacher", "")


    query = {}
```

```
    if department:
        query["course"] = {"$regex": f"^{department}"}
    if single_class:
        query["course"] = single_class
    if instructor:
        query["instructor"] = instructor

    results = list(mongo.db.grades.find(query))
    return render_template("user_page.html",
results=results)
```

Admin Page - Data Management

The `/admin` page allows admins to load and merge data.

Load Data from JSON

```
@app.route("/load_data", methods=["POST"])
def load_data():
    file = request.files["file"]
    data = json.load(file)
```

```
    mongo.db.grades.insert_many(data)
    flash("Data successfully uploaded.", "success")
    return redirect(url_for("admin_page"))
```

Merge Faculty Data

```
@app.route("/merge_data", methods=["POST"])
def merge_data():
    try:
        data_processor.merge_faculty_with_grades()
        flash("Data successfully merged.", "success")
    except Exception as e:
        flash(f"Error: {e}", "danger")
    return redirect(url_for("admin_page"))
```

---

5. Data Visualization - Grade Distribution Graphs

EasyA visualizes grade distributions using **Chart.js**.

Frontend Graph Rendering (`user_page.html`)

```
<canvas id="gradeChart"></canvas>
<script>
```

```
    const ctx =
document.getElementById('gradeChart').getContext('2d');
    const chartData = {{ graph_data | tojson }};
    new Chart(ctx, {
        type: 'bar',
        data: {
            labels: ['A', 'B', 'C', 'D', 'F'],
            datasets: [{
                label: 'Grade Distribution',
                data: chartData,
                backgroundColor: ['green', 'blue',
'yellow', 'orange', 'red']
            }]
        }
    });
</script>
```

---

6. Testing - Unit Tests with `unittest`

Testing Database Insertion

```
import unittest

from unittest.mock import patch

import mongomock

from data_loader import DataLoader


class TestDatabase(unittest.TestCase):

    def setUp(self):

        self.mock_db = mongomock.MongoClient().db

        self.data_loader = DataLoader(self.mock_db, {})


    def test_insert_faculty(self):

        faculty_data = [{"name": "John Doe",
"department": "CIS"}]


self.data_loader.insert_faculty_data(faculty_data)


self.assertEqual(self.mock_db.faculty.count_documents({
}), 1)
```

Testing Graph Data Retrieval

```
class TestGraphAPI(unittest.TestCase):
```

```
    @patch("app.mongo.db")
    def test_get_graph_data(self, mock_db):
        mock_db.grades.find.return_value = [{"aprec": 50, "bprec": 30}]
        response = self.client.get("/get_graph_data?course=CIS101")
        self.assertEqual(response.status_code, 200)
```

---

7. Deployment - Docker Setup

EasyA is deployed using **Docker**.

Docker Compose Setup (`docker-compose.yml`)

```
version: "3.8"
services:
  app:
    build: .
    ports:
      - "5000:5000"
    environment:
      - MONGO_URI=mongodb://mongo:27017/easya
```

```
    depends_on:
      - mongo

  mongo:
    image: mongo:5.0
    container_name: mongodb
    ports:
      - "27017:27017"
```