

## **Erg-lytics**

ECE 49595SD-046 Fall Semester

### Team Members:

Noah Wisniewski - [nwisnie@purdue.edu](mailto:nwisnie@purdue.edu)

Kassandra Bankson - [bankson@purdue.edu](mailto:bankson@purdue.edu)

Aidan Mahaffey - [mahaffea@purdue.edu](mailto:mahaffea@purdue.edu)

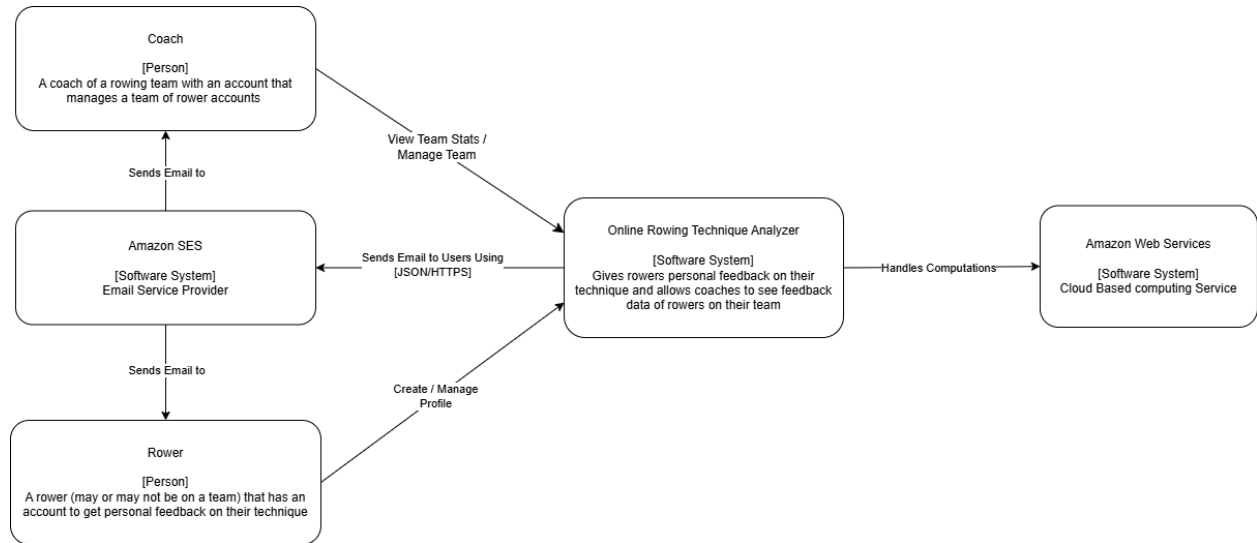
Evan Osborne - [osbor105@purdue.edu](mailto:osbor105@purdue.edu)

## Design Layout:

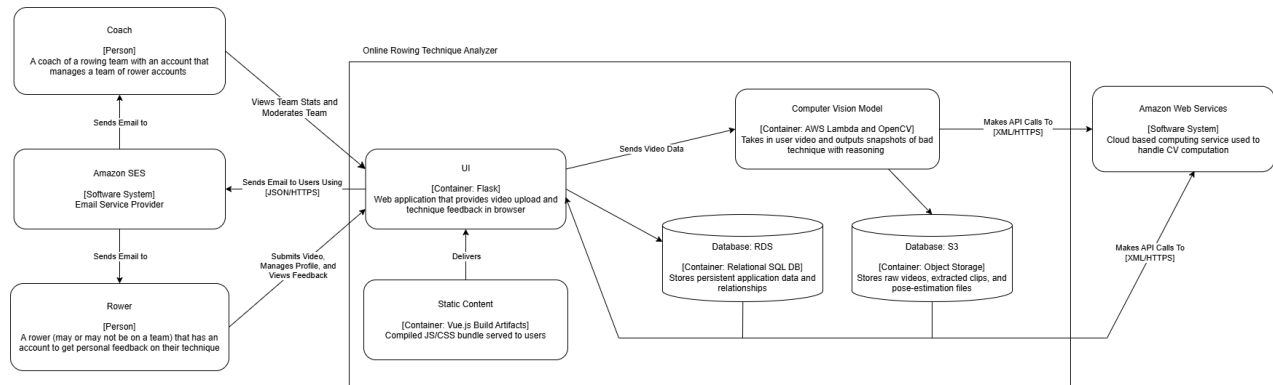
### Technology Stack:

| Category                  | Tech                | Reason  |
|---------------------------|---------------------|---|
| Tool                      | Swagger             | Swagger will help to simplify API design, documentation, and testing.   |
| Tool                      | AWS SES             | SES will help to manage secure and reliable email verification.   |
| Frameworks                | Flask               | Flask will power our API endpoints and connections to AWS services.   |
| Frameworks                | Vue.js              | Vue will be used to build our interactive web interface.  |
| Libraries                 | OpenCV              | OpenCV will help to process and analyze rowing videos for computer vision tasks.  |
| Languages                 | C++ & Python        | C++ will be used when working with OpenCV, and Python will be used to work with Flask.  |
| Cloud or Server Utilities | AWS Lambda          | AWS Lambda will host our web app without requiring a physical server. AWS RDS will store user data such as accounts and team associations in a relational database. |
| Cloud or Server Utilities | AWS RDS             | AWS RDS will store user data such as accounts and team associations in a relational database.   |
| Cloud or Server Utilities | AWS S3              | AWS S3 will handle raw video storage, short error-clip storage, and pose-estimation data.   |
| Cloud or Server Utilities | AWS Cloud Formation | Allows easier building / management of AWS environment  |
| Cloud or Server Utilities | AWS CloudWatch      | CloudWatch will be used to monitor application performance and track errors, in real time.  |

## System Context Diagram:



## System Container Diagram:



**Figma:** [Link](https://www.figma.com/file/Rowlytics/doc/links) is in Rowlytics/doc/links

## **Design Trade:**

***Subsystem:*** *Web Client and API Interface*

### **Alternatives**

#### **D1: Full Flask Integration:**

This approach uses Flask as a lightweight Python web framework to handle routing, template rendering (Jinja), session management, and communication with AWS services via RESTful APIs. It was considered because it aligns well with the Python-based backend and minimizes added complexity.

#### **D2: Vue.js Frontend with Flask API Backend:**

This design separates concerns by using Vue.js for frontend rendering and state management while Flask serves only as an API backend. It was considered for its strong scalability and interactivity but requires significant JavaScript expertise and setup effort.

#### **D3: Full Django Integration:**

This option replaces Flask with Django, leveraging its full-stack capabilities such as built-in authentication, ORM, and admin tools. It was considered because it could reduce the need for external libraries but introduces additional complexity and development overhead.

### **Evaluation Table**

| Criterion                   | Weight | D1  | <b>D2</b> | D3  |
|-----------------------------|--------|-----|-----------|-----|
| Ease of Development         | 25%    | 2   | 9         | 4   |
| Performance                 | 20%    | 4   | 6         | 4   |
| Scalability                 | 10%    | 9   | 4         | 6   |
| Security                    | 20%    | 6   | 6         | 6   |
| Ease of Backend Integration | 25%    | 6   | 9         | 4   |
| Weighted Total              | 100%   | 4.9 | 7.3       | 4.6 |

### **Reasoning for Selection**

We chose a Vue.js frontend with a Flask API backend because it gives us a good balance between interactivity, scalability, and development effort. Vue lets us build a more responsive interface than server-rendered templates, which is important for live feedback and clean data visualization, while still being lighter and easier to learn than something like React. Using Flask strictly as an API backend keeps our Python-based computer vision and AWS integrations simple and consistent with the rest of the system. The main trade-off is added complexity compared to a fully server-rendered frontend, but we felt the improved UI responsiveness and clearer separation of concerns were worth it for this project. Overall, this architecture supports our real-time requirements without overengineering the system.

***Subsystem: Account Verification***

**Alternatives:**

Bcrypt:

This hashing algorithm uses 64-bit blocks to expand symmetric keys. It was considered because it is a strong hash with a slow hashing speed that guards against brute force attacks

Argon2:

This hashing algorithm works by starting with a 64-bit hash before compressing each lane before XORing the final block to achieve the final hash. This hash is naturally memory-intensive and guards well against modern attacks, which is why it is considered.

PBKDF2:

This hashing algorithm works by generating functions for each block before concatenating each of them together. This was considered because of the fast hashing speed and lack of memory intensity.

**Evaluation Table**

| Criterion   | Weight | Bcrypt | Argon2 | PBKDF2 |
|-------------|--------|--------|--------|--------|
| Performance | 75%    | 4      | 2      | 6      |
| Security    | 20%    | 8      | 10     | 6      |
| Memory      | 5%     | 7      | 3      | 8      |
| Total       | 100%   | 5.15   | 4.05   | 6.1    |

**Reasoning for Selection**

We choose PBKDF2 because it is a light-weight solution perfect for our product. It has strong performance compared to other alternatives without the worry of compromising security.

### ***Subsystem: Automated Emailing***

#### **Alternatives:**

##### Amazon Simple Email Service (SES):

A cloud-based email-sending service provided by AWS that supports automated sending through verified domains and templates. It aligns closely with the system's AWS-based architecture and emphasizes reliability and low cost.

##### Mailchimp:

An external email platform primarily designed for marketing emails, offering an API, visual template editor, and delivery analytics dashboard. While reliable and user-friendly, it introduces significantly higher costs and requires integration with non-AWS services.

##### Mailgun:

An email delivery service that focuses on high deliverability and tracking through APIs. It provides strong control over email content and metrics with a simpler setup than SES sandbox mode, but still adds an external dependency and less visual template management.

#### **Evaluation Table:**

| Criterion            | Weight | SES | Mailchimp | Mailgun |
|----------------------|--------|-----|-----------|---------|
| Delivery Reliability | 30%    | 9   | 9         | 9       |
| Scheduling/Templates | 25%    | 8   | 8         | 8       |
| Content Accuracy     | 25%    | 8   | 9         | 8       |
| Low Cost             | 20%    | 10  | 6         | 8       |
| Total                | 100%   | 8.7 | 8.15      | 8       |

#### **Reasoning for Selection:**

Amazon SES was selected as the best option for the Automated Emailing Subsystem because it achieved the highest overall score in the trade study. It gives a balanced level for cost, reliability, scheduling configuration, and accuracy. On top of this, it allows for simple integration with our current planned AWS infrastructure. SES leverages the same environment already used in Lambda, RDS, EventBridge, and S3, which simplifies our system management and allows for consistent performance monitoring. All of these things in combination are why Amazon SES is the best alternative for our system and the Automated Emailing Subsystem.

### ***Subsystem: Computer Vision Pose Estimator***

#### **Alternatives:**

##### **YOLOv8 Pose:**

A model that predicts up to 17 body landmarks, many of which will be useful for our project (knees, elbows, hips), but some of which won't be required such as eyes and ears. Different versions exist which balance speed and accuracy. Can be easily implemented using Python.

##### **MediaPipe Pose:**

A model used by Google developers for low latency pose estimation. Can place up to 33 body landmarks. It can also estimate depth, despite not being a 3D pose estimation model. Easily implemented using Python.

##### **AlphaPose:**

A model capable of delivering a high degree of accuracy at the cost of a higher computational strain. Tracks 17 body landmarks. Typically more difficult to deploy than most other pose estimation models.

#### **Evaluation Table:**

| Criterion         | Weight | YOLOv8 Pose | MediaPipe Pose | AlphaPose |
|-------------------|--------|-------------|----------------|-----------|
| Subject Tracking  | 40%    | 8           | 8              | 9         |
| Movement Tracking | 40%    | 7           | 8              | 7         |
| Latency           | 20%    | 7           | 9              | 4         |
| Total             | 100%   | 7.4         | 8.2            | 7.2       |

#### **Reasoning for Selection:**

MediaPipe Pose was chosen to be our pose estimation model because it offers strong overall tracking accuracy as well as low latency. MediaPipe outperformed YOLOv8 Pose during movement tracking tests as MediaPipe was better able to track fast movements while also analyzing the testing video in a shorter period of time. MediaPipe offered comparable accuracy when compared to AlphaPose for our chosen use, however it was able to complete video analysis in a shorter period of time.