



A Cloud-Based Training Management System

A Requirement for the
Advanced Software Engineering Course (SDEV 4304)

Submitted by:

Ghadeer Alhayek	(220191207)
Ayah Elshawwa	(220191521)
Raghad Alkhatib	(220190759)
Malak Tafesh	(220191354)

Submitted to:

Eng. Dr. Rebhi Baraka

Abstract

We developed a cloud-based training management web system that is designed to streamline and automate training processes in organizations. TMS offers a user-friendly interface that allows managers, advisors, and trainees to efficiently manage and track training activities. It facilitates creating training programs and meetings, tracking trainee's attendance and progress, and providing interactive learning materials. With features such as registration approval, resolving meeting conflicts, performance assessments, and reporting tools, TMS ensures that training initiatives are effectively implemented and monitored. Its cloud-based architecture allows for easy scalability and accessibility, making it ideal for organizations of all sizes.

1. Introduction

The cloud-based training management system is a web application designed to streamline and automate the management of training programs. It provides a centralized platform for trainees, managers, and advisors to collaborate and track the progress of training activities. TMS offers features such as trainee registration, course enrollment, progress tracking (through attendance forms), and communication tools (using emails).

We developed TMS by following the agile methodology, which emphasizes iterative development, frequent feedback, and collaboration. This approach allowed us to continuously improve and adapt to our changing comprehension of the requirements throughout the development lifecycle. The use of agile practices facilitated quick delivery of new features, enhanced responsiveness to the requirements, and efficiently managed our project timelines.

Overall, the cloud-based training management system provides a comprehensive solution for managing trainees, advisors and training programs efficiently and effectively. Its cloud-based architecture, and our approach with the agile development methodology, ensured flexibility, scalability, and responsiveness in meeting all of TMSs' requirements.

2. TMS's Requirements

TMS consists of 3 components, the manager, advisor and trainee. All these components fulfil the following requirement

1. Trainees:

- They must register by providing necessary information, uploading personal files and documents upon registration.
- They can apply for training programs, fill in attendance forms, and request meetings with their advisors.

2. Advisors:

- They must provide necessary information for registration.
- They are classified based on their discipline, manage their own trainees and keep up with their attendance

- They can approve or reject trainee meeting request.

3. Managers:

- They review training and registration requests by both advisors and trainees and decide whether to accept them or not.
- They generate unique trainee and advisor IDs, handle everyone's account management, manage training programs, and billing issues.
- They can update everyone's data in the cloud.
- They can send emails to advisors and trainees.

4. The System (TMS):

- TMS is responsible for generating highly secure and random passwords for users upon manager's approval of their registration request.
- It's also responsible for resolving meeting schedule conflicts.

- TMS's UML Diagram

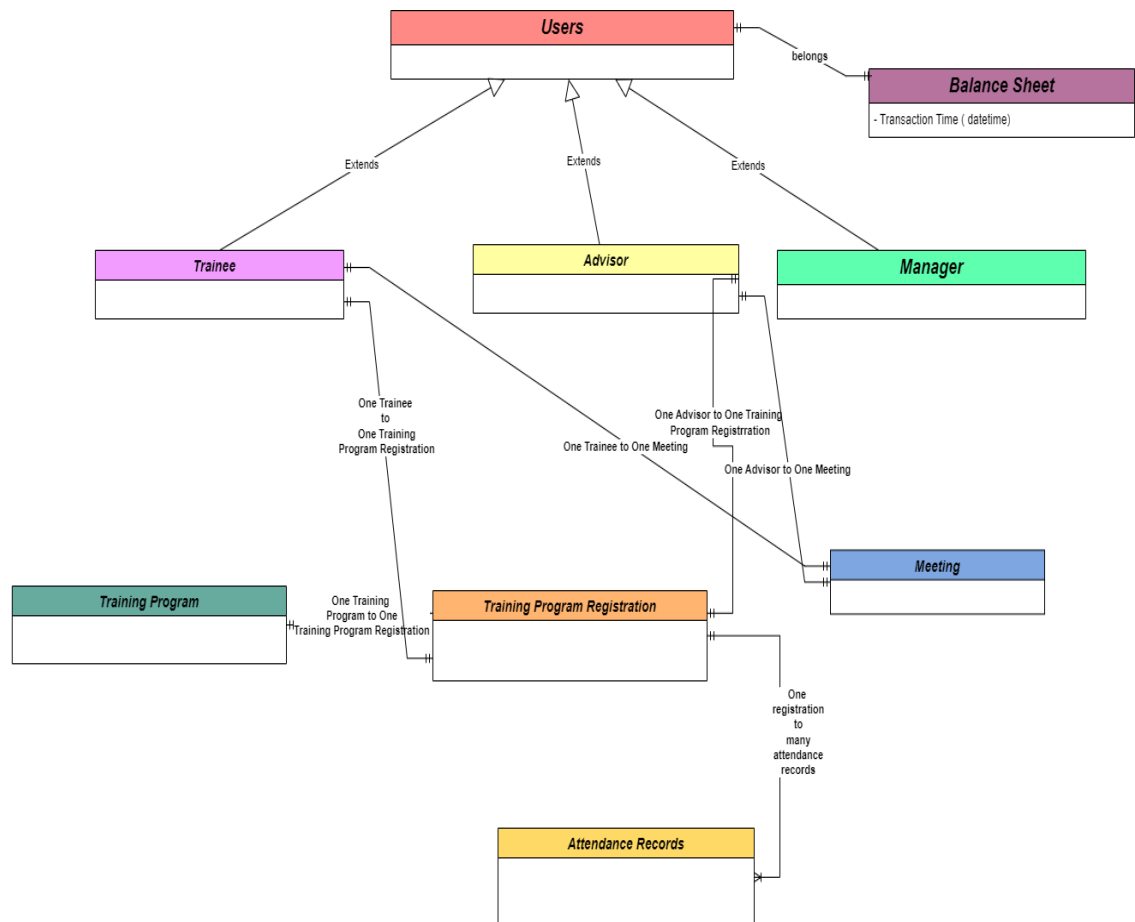


Figure 1- TMS's Componentr UML Diagram

- TMS's Use Case Scenario

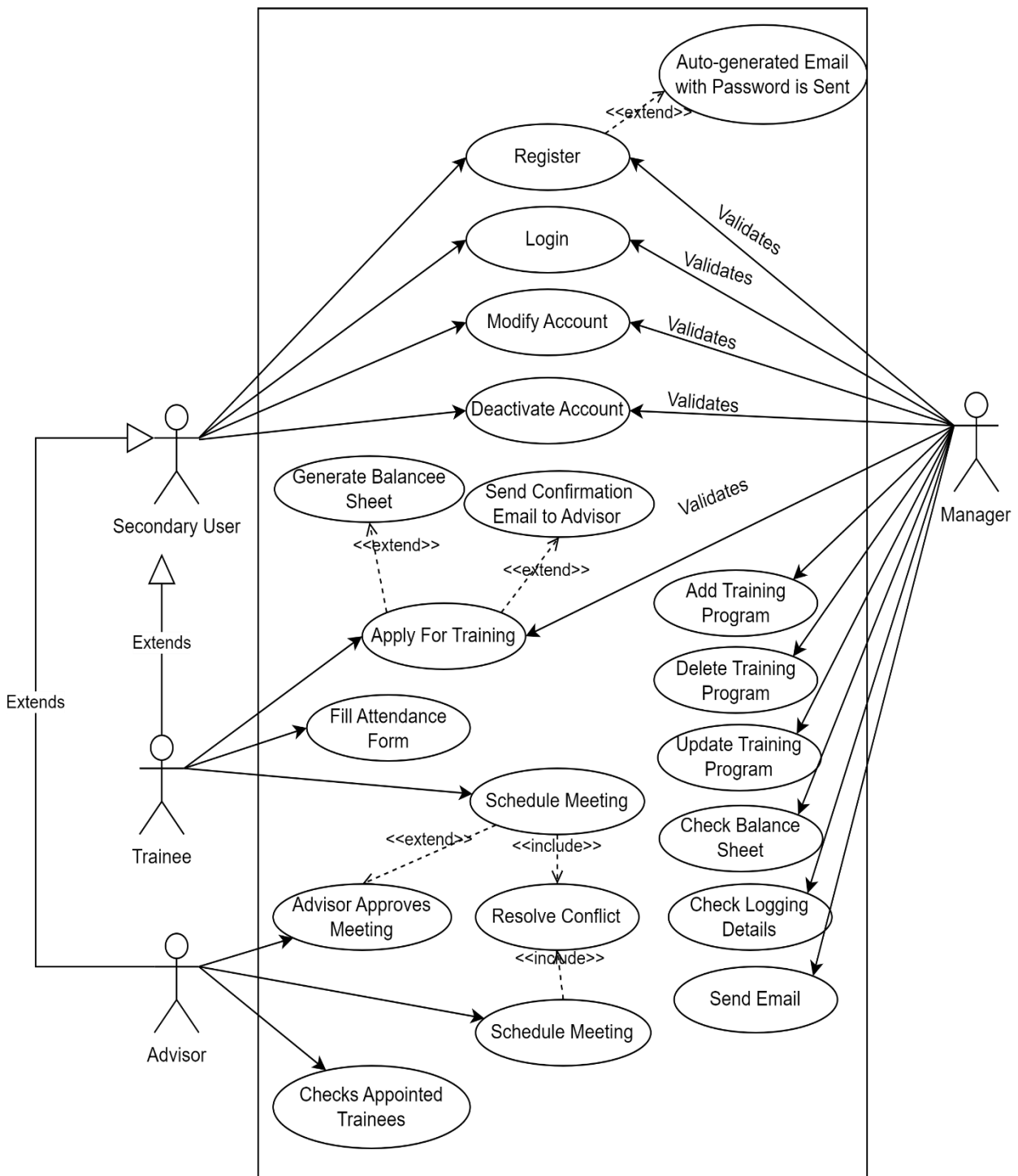


Figure 2 - TMS's Use Case Scenario

3. TMS's Architecture and Design

Based on the requirements of TMS, we decided that to build a three-tier architecture system. It consists of, the presentation tier, application tier, and data tier.

1. **Presentation Tier (Frontend):** also known as the frontend, is responsible for handling the user interface and displaying information to the users. It consists of an HTML frontend that enables users to interact with TMS.
2. **Application Tier (Backend):** also known as the backend, encompasses the business logic and application processing. It includes a Flask backend that processes requests received from the frontend, interacts with the MySQL database, and returns responses to the users.
3. **Data Tier (Database):** or the database tier, is responsible for storing and managing the data required by TMS. In our case, we hosted a MySQL database on Amazon RDS to store and retrieve data related to trainees, managers, advisors, and the overall system.

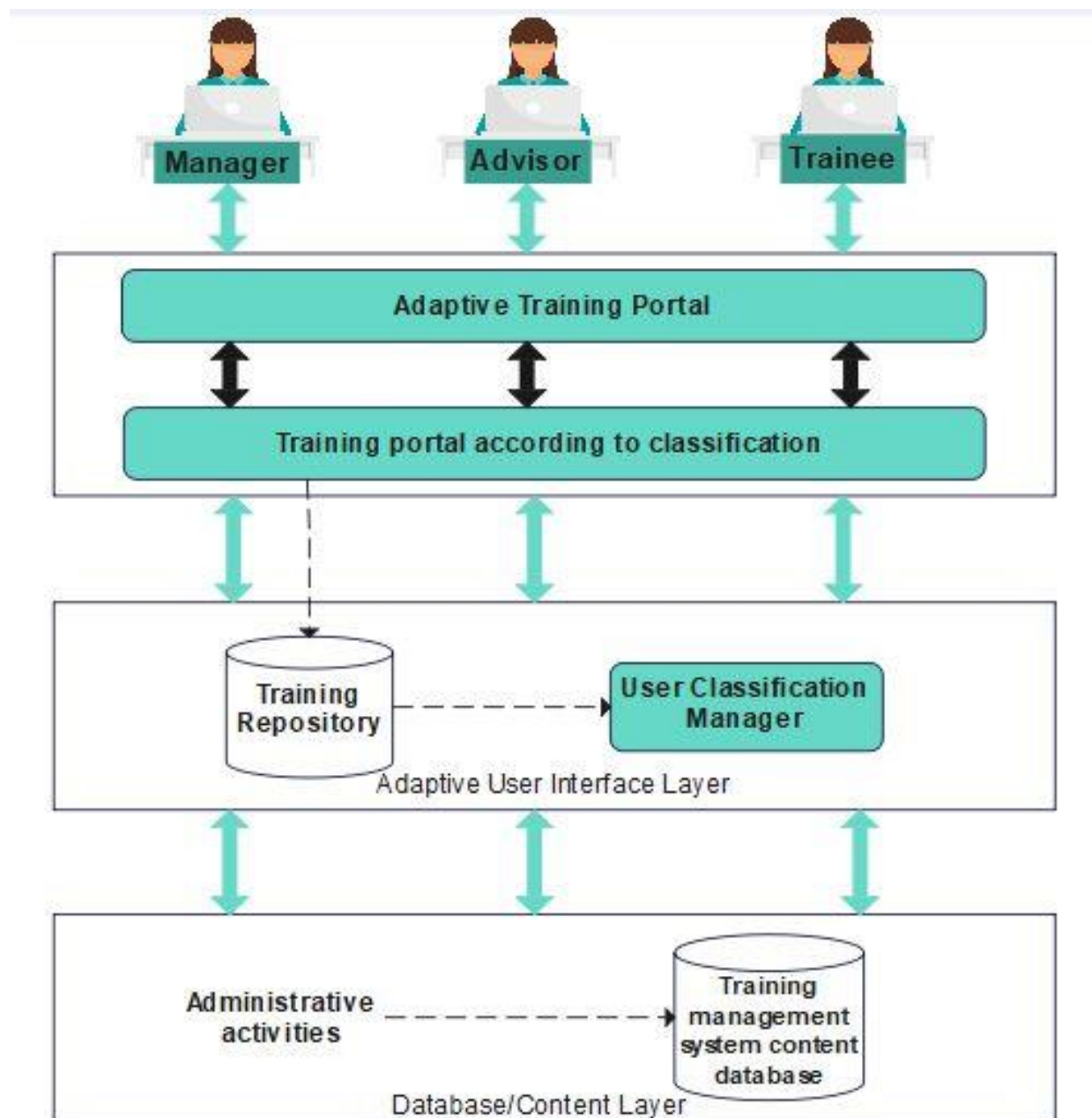


Figure 3 - Abstract View of TMS's Architecture

- **Functional components of TMS:**

- The provides user registration and login functionality to allow trainees, managers, and advisors to access their respective accounts.
- Trainees can upload their CVs upon registration, view their own progress, and communicate with their advisors via scheduling meetings.

- Managers have the ability to view and manage trainee profiles, assign advisors, and monitor overall training progress.
- TMS is integrated with external services for email notifications (Amazon SES) and user authentication (Amazon IAM).
- Advisors are able to review their trainees, provide feedback, and approve or schedule meetings with their trainees.
- TMS should support email functionality for important updates and actions, and resolve any scheduling conflict in meetings.

- **Data Management Requirements:**

- TMS stores and manages trainee, manager, and advisor profiles, including personal information, contact details, and role-specific data..

- **Security and Access Control Requirements:**

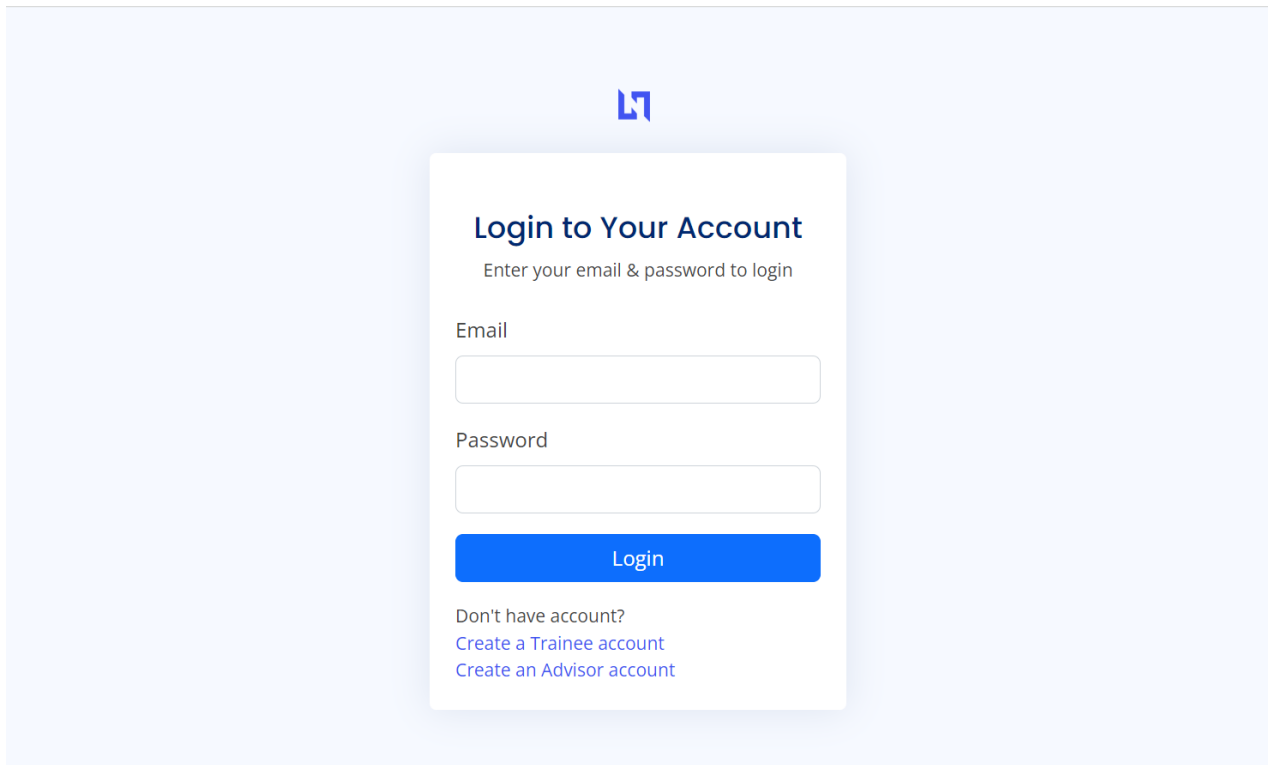
- TMS implements secure user authentication and authorization mechanisms to ensure only authorized individuals can access specific features and data.
- Trainee, manager, and advisor accounts are protected using strong password encryption.
- Role-based access control are enforced to restrict access to sensitive information and functionalities based on user roles.

- **Scalability and Performance Requirements:**

- TMS should be designed to handle a growing number of users, training documents, and concurrent interactions.
- Performance optimizations should be implemented to ensure fast response times and efficient data retrieval.

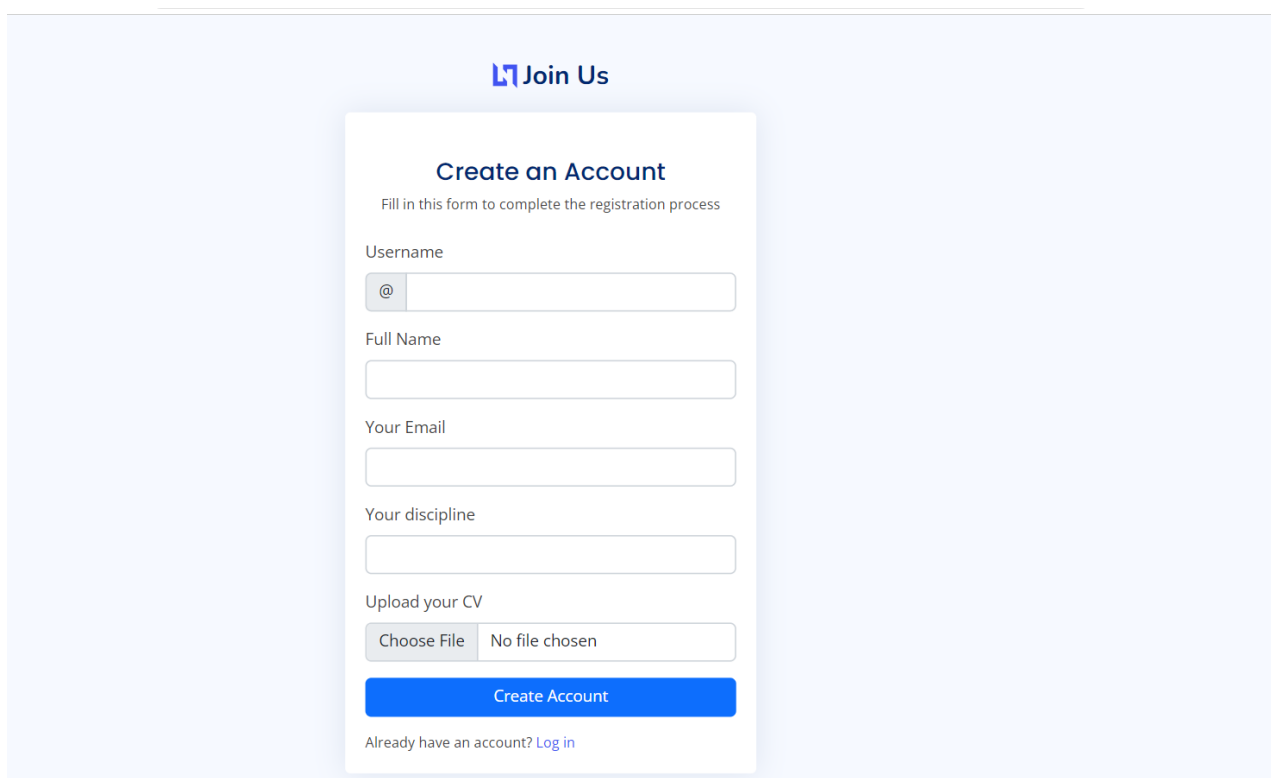
- **User Interfaces**

In this section, we will showcase some of TMS's interface for readability reasons. Please feel free to check out TMS and experience being an advisor or a trainee and see real time interfaces for yourself!



The image shows a login interface for a user. At the top center is a blue logo consisting of two interlocking squares. Below the logo is a white card with a blue border. The card has the title "Login to Your Account" in bold, followed by the instruction "Enter your email & password to login". There are two input fields: "Email" and "Password". Below the "Password" field is a blue button labeled "Login". At the bottom of the card, there is a link "Don't have account?" followed by two sub-links: "Create a Trainee account" and "Create an Advisor account".

The Login Interface for all User



The image shows a registration interface for a user. At the top center is a blue logo consisting of two interlocking squares, followed by the text "Join Us". Below the logo is a white card with a blue border. The card has the title "Create an Account" in bold, followed by the instruction "Fill in this form to complete the registration process". There are five input fields: "Username" (with an "@" icon), "Full Name", "Your Email", "Your discipline", and "Upload your CV" (with a "Choose File" button and "No file chosen" text). Below the "Upload your CV" field is a blue button labeled "Create Account". At the bottom of the card, there is a link "Already have an account? Log in".

Create Advisor Account

Join Us

Create an Account

Fill in this form to complete the registration process

Username

@

Full Name

Your Email

Desired Field

Area of Interest

Upload your CV

Choose File

No file chosen

Upload your motivation letter

Choose File

No file chosen

Other materials

Videos, Projects, Infographics, Certificates, ...

Choose File

No file chosen

Create Account

Already have an account? [Log in](#)


Create Trainee Account

TMS

Search

4

3

 mars2001

Dashboard

Trainees

Advisors

Training Programs

Balance Sheet

Send an Email


System Log

Manager

Home

mars2001

manager@gmail.com



Overview

About

mars is responsible for managing the system, trainees, and advisors. mars can review training requests, manage all registered trainees and advisors accounts.

Profile Details

Full Name

mars

Email

manager@gmail.com

Manager Dashboard

11

TMS

Search

ghadeerhayek

Dashboard

Trainees

Advisors

Training Programs

Balance Sheet

Send an Email

System Log

Trainee Registration Requests

Manager / Trainees / Trainees Registration Requests

10 entries per page

Search...

Id	Username	Email	Desired field	Area of Interest
No entries found				

All Trainee Registration Requests (No new Trainees at the moment 😞)

TMS

Search

mars2001

Dashboard

Trainees

Advisors

Training Programs

Balance Sheet

Send an Email

System Log

Training Programs

Manager / Training Programs

10 entries per page

Search...

Id	Name	Description	Area	Fees	Start Date	End Date		
1	Solar Panels	Very Common	Renewable Energy	50.0	2023-05-17 00:00:00	2023-05-30 00:00:00	Edit	Delete
2	Solar Panels	Descent	Renewable Energy	40.0	2023-05-29 00:00:00	2023-05-29 00:00:00	Edit	Delete

Showing 1 to 2 of 2 entries

All Programs

TMS

Search

mars2001

Dashboard

Trainees

Advisors

Training Programs

Balance Sheet

Send an Email

System Log

Add New Training Program

Manager / Training Programs / Create

Training Program Details

Program Name

Program Description

Program Area

Choose...

Program Fees

Program Start Date

dd/mm/yyyy

Program End Date

dd/mm/yyyy

Add

Reset

Add Programs

ririmajed

Dashboard

Apply for training program

My Training

Meetings

Trainee

Home

ririmajed

Machine Learning
Web Development

Information

Email

Trainee ID

raghadalkhattib3@gmail.com

18

Status

on_training

User Documents

Trainee Dashboard

ririmajed

Dashboard

Apply for training program

My Training

Meetings

My Training Program

Trainee / My Training Program

10 entries per page

Search...

Registration ID	Registration Time	Program ID	Advisor ID	Training Status	Attendance Form
12	2023-05-30 18:26:01	1	7	approved	Details

Showing 1 to 1 of 1 entries

Trainee's Training

ririmajed

Dashboard

Apply for training program

My Training

Meetings

Attendance Form

Trainee / My Training Program / Attendance Form / Records

10 entries per page

Search...

ID	Date	Start Time	End Time
3	2023-03-11	10:00:00	12:00:00

Showing 1 to 1 of 1 entries

Add new record

Trainee's Attendance Form

3

Search

4

3

ririmajed

Dashboard

Apply for training program

My Training

Meetings

Training Programs

Trainee / Training Programs

10

entries per page

Search...

Id	Name	Description	Area	Fees	Start Date	End Date	
1	Solar Panels	Mid	Renewable Energy	50.0	2023-05-23 00:00:00	2023-05-22 00:00:00	Apply
2	Solar Panels	Descent	Renewable Energy	40.0	2023-05-29 00:00:00	2023-05-29 00:00:00	Apply
3	Name	Description	Machine Learning	40.0	2023-05-30 00:00:00	2023-06-08 00:00:00	Apply

Showing 1 to 3 of 3 entries

Training Programs Trainees Can Apply To

Search

4

3

ririmajed

Dashboard

Apply for training program

My Training

Meetings

All Upcoming Meetings

Trainee / My Meetings

10

entries per page

Search...

Meeting ID	Details	Status	Link	Start Datetime	End Datetime
11		cancelled		2023-03-11 10:00:00	2023-03-11 12:00:00
12		approved		2023-05-16 21:35:00	2023-05-25 21:35:00
13		pending		2023-05-16 22:35:00	2023-05-25 22:35:00
14		approved		2023-05-16 21:35:00	2023-05-25 21:35:00

Showing 1 to 4 of 4 entries

Arrange New Meeting

Trainees Meetings

Search

ayah

Dashboard

My Trainees

Meetings

Profile

Home

Profile

ayah

ayahs19302@gmail.com

Overview

Edit Profile

Settings

About

Some description

Profile Details

Username

ayah

Full Name

ayah sh

Email

ayahs19302@gmail.com

Discipline

Web Development

Advisor Dashboard

Search

ayah

Dashboard

My Trainees

Meetings

Current Trainees

Home / My Trainees / Current Trainees

10 entries per page

Search...

ID	Username	Fullname	Desired field	Area of Interest	Status	Attendance Form
18	ririmajed	RAGHAD	Web Development	Machine Learning	on training	Check Ririmajed's Attendance Form

Showing 1 to 1 of 1 entries

Current Advisor's Trainees

Search

ayah

Dashboard

My Trainees

Meetings

All Upcoming Meetings

Advisor / My Meetings

10 entries per page

Search...

Meeting ID	Details	Status	Link	Start Datetime	End Datetime	Action
11		cancelled		2023-03-11 10:00:00	2023-03-11 12:00:00	Approve Cancel
12		approved		2023-05-16 21:35:00	2023-05-25 21:35:00	Approve Cancel
13		pending		2023-05-16 22:35:00	2023-05-25 22:35:00	Approve Cancel
14		approved		2023-05-16 21:35:00	2023-05-25 21:35:00	Approve Cancel

Showing 1 to 4 of 4 entries

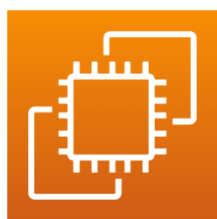
[Arrange New Meeting](#)

Upcoming Advisor Meetings

- **Real-world constraints of AWS**

In AWS, the EC2 instances run by default using the public IPv4 address that is generated by AWS. However, the public IPv4 DNS address provided by AWS doesn't run by default and needs to be configured from the EC2 console, which is rather complicated and risky as it might shut down the instance. This is because the server runs using nginx which is not as common as Apache for instance, so doing all that configuration is not very convenient as developers use SaaS providers so that they don't have to go through the configuration process. We think that configuring the DNS address by default from AWS would make it a lot easier for developers to test their instances using DNS address rather than IPv4 addresses.

4. Used AWS Services



AWS EC2 Instance



Amazon Simple Email Service



AWS Lambda Function



AWS IAM

We utilized a range of AWS services to support the implementation and deployment of our system. These services included Amazon Relational Database Service (RDS), Amazon Elastic Compute Cloud (EC2) instances, security groups, Amazon Simple Email Service (SES), AWS Identity and Access Management (IAM), and AWS Lambda Function.

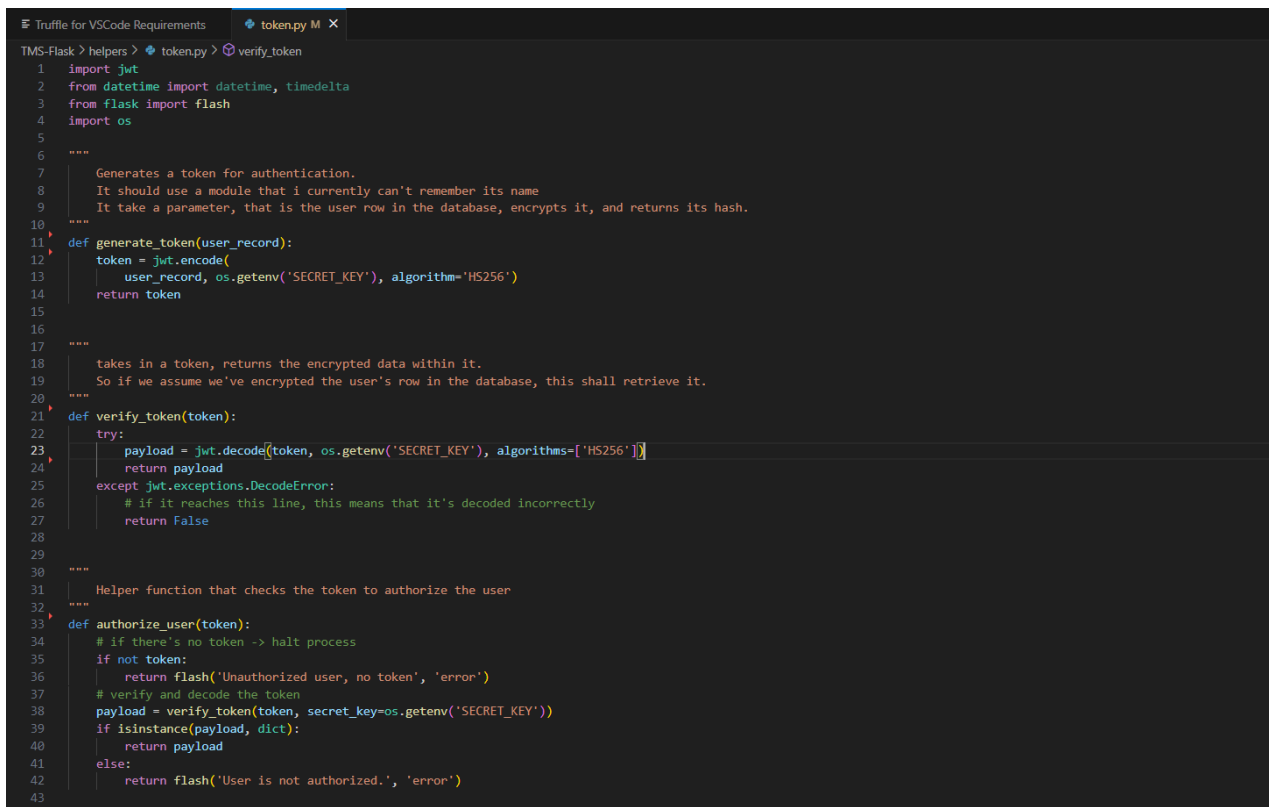
We leveraged the RDS service as our managed database solution, to enable us to store and access relational MYSQL data efficiently and securely. We also employed EC2 instances to host and run our application components, and provide scalable compute resources to handle varying workloads.

To ensure robust security measures, we utilized security groups to define network access rules and control inbound and outbound traffic.

IAM played a vital role in managing user access and permissions, allowing us to define fine-grained access controls and ensure secure authentication in the sending emails process. SES was employed for reliable and scalable email communication, enabling us to send emails to users for password authentication once their registration is approved by the manager. It provided an abstraction for the email delivery functionality using APIs that made the whole process easier. Lastly, we utilized AWS Lambda, a server less compute service, to invoke the send email functions in response to the manager's events. These AWS services collectively provided us with a scalable, secure, and reliable infrastructure foundation to build and deploy our application in the cloud environment.

5. Implementation

In this section, we will preview some of the most important code and the logic behind. Please feel free to check out all the source code of TMS through the provided GitHub link in the references.



```
Truffle for VSCode Requirements token.py M X
TMS-Flask > helpers > token.py > verify_token
1 import jwt
2 from datetime import datetime, timedelta
3 from flask import flash
4 import os
5
6
7     Generates a token for authentication.
8     It should use a module that i currently can't remember its name
9     It take a parameter, that is the user row in the database, encrypts it, and returns its hash.
10
11
12 def generate_token(user_record):
13     token = jwt.encode(
14         user_record, os.getenv('SECRET_KEY'), algorithm='HS256')
15     return token
16
17
18     takes in a token, returns the encrypted data within it.
19     So if we assume we've encrypted the user's row in the database, this shall retrieve it.
20
21
22 def verify_token(token):
23     try:
24         payload = jwt.decode(token, os.getenv('SECRET_KEY'), algorithms=['HS256'])
25         return payload
26     except jwt.exceptions.DecodeError:
27         # if it reaches this line, this means that it's decoded incorrectly
28         return False
29
30
31     Helper function that checks the token to authorize the user
32
33
34 def authorize_user(token):
35     # if there's no token -> halt process
36     if not token:
37         return flash('Unauthorized user, no token', 'error')
38     # verify and decode the token
39     payload = verify_token(token, secret_key=os.getenv('SECRET_KEY'))
40     if isinstance(payload, dict):
41         return payload
42     else:
43         return flash('User is not authorized.', 'error')
```

Token Generation, Verification & Authorization

```

TMS-Flask > controller > auth_controller.py > handle_login
12 def handle_login(request):
13     # get information from request
14     email = request.form['email']
15     password = request.form['password']
16     if not email or not password:
17         flash("Missing email or password", 'error')
18         return redirect(url_for('auth.login_view'))
19     # let's look whether we can find those credentials
20     query = text("SELECT * from users where email = :email and password = :password")
21     params = {"email": email, "password": password}
22     result = db.session().execute(query, params)
23     row = result.fetchone()
24     # return jsonify('querying trainee returned nothing')
25     # print(row)
26     if not row:
27         return jsonify("nothing");
28         flash('Email or password are incorrect, please try again', 'error')
29         return redirect(url_for('auth.login_view'))
30     # NOTE: We need to check the statuses of the trainee and the advisor before authorizing them to the system
31     # Which means we'll fetch the trainee and advisor records
32     # which also means deleting the trainee, advisor helpers :))))))
33     classification = row[3]
34     # print(classification)
35     if classification == "manager":
36         # get the row from the database
37         manager_record = db.session.execute(text(
38             "SELECT * from managers where userID = :userID",
39             {"userID": row[0]}
40         ).fetchone())
41         manager = {
42             "managerID": manager_record[0],
43             "username": manager_record[1],
44             "fullname": manager_record[2],
45             "email": manager_record[3],
46             "userID": manager_record[4],
47         }
48         # generate token
49         token = tokenHelper.generate_token(manager)
50         # print("inside manager")
51         response = redirect(url_for('manager.dashboard_view'))
52         response.set_cookie('token', token)
53         return response
54     elif classification == "advisor":
55         # get the row from the database
56         advisor_record = db.session.execute(
57             text("SELECT * from advisors where userID = :userID and status in ('active','training')"),
58             {"userID": row[0]}
59         ).fetchone()
60         if not advisor_record:
61             # it means the status match failed, so the user can not be authorized
62             flash("you are not authorized to the system.")
63             return redirect(url_for('auth.login_view'))
64         advisor = {
65             "advisorID": advisor_record[0],
66             "username": advisor_record[1],
67             "fullname": advisor_record[2],
68             "email": advisor_record[3],
69             "discipline": advisor_record[4],
70             "status": advisor_record[5],
71             "userID": advisor_record[6],
72         }
73         # generate token
74         token = tokenHelper.generate_token(advisor)
75         # print("inside manager")
76         response = redirect(url_for('advisor.dashboard_view'))
77         response.set_cookie('token', token)
78         return response
79     elif classification == "trainee":
80         # get the row from the database
81         trainee_record = db.session.execute(
82             text("SELECT * from trainees where userID = :userID and status in ('active','on_training')"),
83             {"userID": row[0]}
84         ).fetchone()
85         if not trainee_record:
86             # it means the status match failed, so the user can not be authorized
87             flash("you are not authorized to the system yet, wait for admin approval")
88             return redirect(url_for('auth.login_view'))
89         trainee = {
90             "traineeID": trainee_record[0],
91             "username": trainee_record[1],
92             "fullname": trainee_record[2],
93             "email": trainee_record[3],
94             "desired_field": trainee_record[4],
95             "area_of_training": trainee_record[5],
96             "status": trainee_record[6],
97             "balance": trainee_record[7],
98             "training_materials": trainee_record[8],
99             "userID": trainee_record[9],
100         }
101         # generate token
102         token = tokenHelper.generate_token(trainee)
103         # print("inside trainee")
104         response = redirect(url_for('trainee.dashboard_view'))
105         response.set_cookie('token', token)
106         return response
107     else:

```

The Login Implementation

```
Truffle for VSCode Requirements  logger.py X
TMS-Flask > helpers > logger.py > ...
1  import logging
2
3
4  class Logger:
5      _instance = None
6
7      def __new__(cls, *args, **kwargs):
8          if not cls._instance:
9              cls._instance = super(Logger, cls).__new__(cls, *args, **kwargs)
10             return cls._instance
11
12     def __init__(self):
13         self.logger = logging.getLogger('my_logger')
14         self.logger.setLevel(logging.DEBUG)
15
16         # Create a file handler and set the log level
17         file_handler = logging.FileHandler('app.log')
18         file_handler.setLevel(logging.DEBUG)
19
20         # Create a log formatter and add it to the file handler
21         formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
22         file_handler.setFormatter(formatter)
23
24         # Add the file handler to the logger
25         self.logger.addHandler(file_handler)
26
27     def log(self, message):
```

The Logger Class Used to Help Track User's Activities

```
controller > manager > manager_advisor_controller.py > ...
1  from flask import request, render_template, jsonify, flash, url_for, redirect
2  from sqlalchemy import text
3  from app import db
4  import secrets
5  import helpers.manager_helper as mghelper
6  import helpers.token as token_helper
7
8
9  """
10     This is the function that prepares data for the 'pending advisors' view, and returns the view with its data
11     """
12  def get_pending_advisors(request):
13      # token is the manager id or the manager record
14      token = request.cookies['token']
15      if not token:
16          flash('Token not found, invalid request', 'error')
17          return redirect(url_for('auth.login_view'))
18      manager = token_helper.verify_token(token)
19      # if the query returned nothing -> it actually means we can't render the dashboard
20      if not manager:
21          flash('Invalid token', 'error')
22          return redirect(url_for('auth.login_view'))
23      query = text("SELECT * from advisors where status = 'pending'")
24      result_cursor = db.session.execute(query)
25      rows = result_cursor.fetchall()
26      advisors = []
27      for row in rows:
28          advisors.append(row._data)
29      return render_template('manager/advisor/pending_advisors.html', manager=manager, advisors=advisors)
30
```

Get Pending Advisors

```

TMS-Flask > controller > manager > manager_trainee_controller.py > ...
33 """
34 This is the controller function that handles the approve button in the pending trainees view
35 """
36 def approve_trainee_registration(request):
37     token = request.cookies['token']
38     # make sure manager is authorized
39     if not token:
40         flash('Token not found, invalid request', 'error')
41         return redirect(url_for('auth.login_view'))
42     manager = token_helper.verify_token(token)
43     # if the query returned nothing -> it actually means we can't render the dashboard
44     if not manager:
45         flash('Invalid token', 'error')
46         return redirect(url_for('auth.login_view'))
47     # get hidden form data
48     traineeID = request.form['traineeID']
49     traineeEmail = request.form['traineeEmail']
50     pass_length = 7
51     # secret is python module that generates passwords according to ur preferred length
52     password = secrets.token_urlsafe(pass_length)
53     # insert trainee to user table first to link it to trainee table using userID(PK->FK relationship)
54     user_query = text("INSERT INTO users (password, email, classification) VALUES (:password, :email, 'trainee')")
55     user_cursor = db.session.execute(user_query, {'password': password, 'email': traineeEmail})
56     db.session.commit()
57     if not user_cursor:
58         return jsonify('the problem is in insert to users')
59     flash('Failed to approve trainee', 'error')
60     return redirect(url_for('manager.get_pending_trainees_view'))
61     # get userID from previous query
62     userID = user_cursor.lastrowid
63     # update trainee table with userID
64     trainee_query = text("UPDATE trainees SET status = 'active', userID = :userID WHERE traineeID = :traineeID")
65     trainee_cursor = db.session.execute(trainee_query, {'userID': userID, 'traineeID': traineeID})
66     trainee_rows = trainee_cursor.rowcount
67     db.session.commit()
68     if not trainee_rows:
69         # return jsonify('the problem is in update trainees', userID)
70         flash('Failed to approve trainee', 'error')
71         return redirect(url_for('manager.get_pending_trainees_view'))
72     flash('Trainee approved successfully', 'success')
73     # get the user so we can send him his data
74     user = db.session.execute(text("SELECT password, email from users where userID = :userID"),
75                               {'userID': userID}).fetchone()
76     # send credentials to the trainee
77     recipient = user[1]
78     sender = manager["email"]
79     message = """
80     Dear trainee,
81
82     Welcome! We're delighted to have you join our system.
83
84     As a valued member, we're here to support you every step of the way. If you have any questions or need assistance,
85     don't hesitate to reach out to our friendly support team.
86
87     Let's embark on this exciting journey together!
88     Here are your login credentials, Don't share them with anyone
89     Email: {0}
90     Password: {1}
91     Note: you can change your information anytime.
92
93
94     Best regards,
95     {2} from TMS.
96     """.format(user[1], user[0], manager["fullname"])
97     subject = "Welcome to TMS!"
98     response = helper.send_email(recipient=recipient, sender=sender, message=message, subject=subject)
99

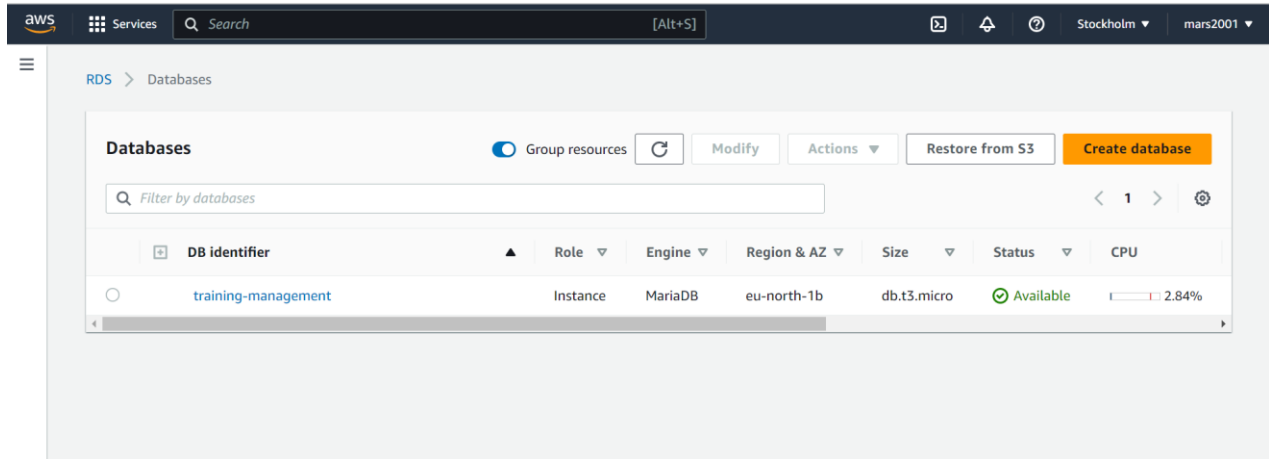
```

Approving Trainees Registration Request

(The Same Logic is Applied To Advisor's Registration Request)

6. Data

The data model of our system is implemented using MySQL, which is hosted on Amazon RDS (Relational Database Service).



The Database Instance in RDS

In RDS, the MySQL database is stored and its various entities and their related information are managed. This includes data related to trainees, managers, advisors, and other system-specific entities. The data model consists of table, as shown in the previous figures, that represent different entities, and the relationships between these entities define the structure and organization of the data.

As shown in the previous figures, the data itself is stored within the MySQL database instance provided by Amazon RDS, which is a fully managed database service that simplifies the deployment, management, and scaling of databases in the cloud. It offers features such as automated backups, high availability, and monitoring capabilities.

By utilizing MySQL with Amazon RDS, TMS benefits from a reliable and scalable database solution that ensures data integrity, availability, and performance. The data model is designed to meet the requirements of TMS, allowing efficient storage, retrieval, and manipulation of data to support the functionality and operations of the training management system.

```
aws Services Search [Alt+S] Stockholm mars2001

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

Last login: Tue May 30 10:08:25 2023 from 213.6.13.161
ubuntu@ip-172-31-39-17:~$ mysql -u admin -p -h training-management.cn5tkelt6vnd.eu-north-1.rds.amazonaws.com
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7157
Server version: 5.5.5-10.6.10-MariaDB-log managed by https://aws.amazon.com/rds/

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

i-0ac2efbe2b3b2d324 (TMS-Flask-WebServer)
PublicIPs: 13.51.85.106 PrivateIPs: 172.31.39.17

Executing and Connecting RDS using the Connection Endpoint in EC2 Console

```
aws Services Search [Alt+S] Stockholm mars2001

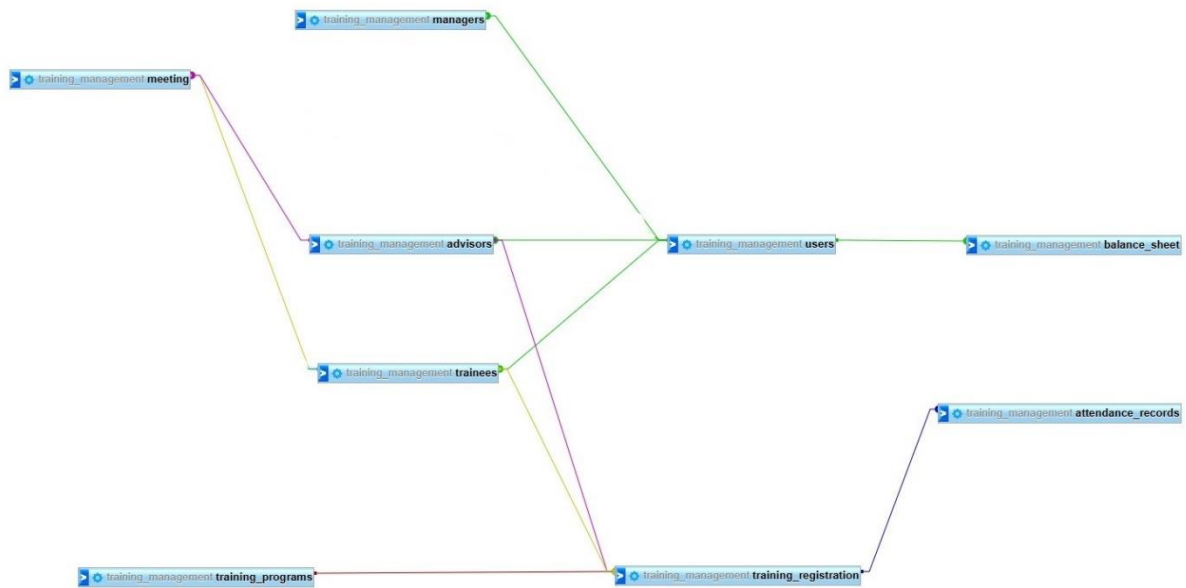
mysql> use training_management;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_training_management |
+-----+
| advisors                      |
| attendance_records            |
| balance_sheet                 |
| emails                        |
| managers                      |
| meetings                      |
| notifications                  |
| trainees                      |
| training_programs             |
| training_registration         |
| users                         |
+-----+
```

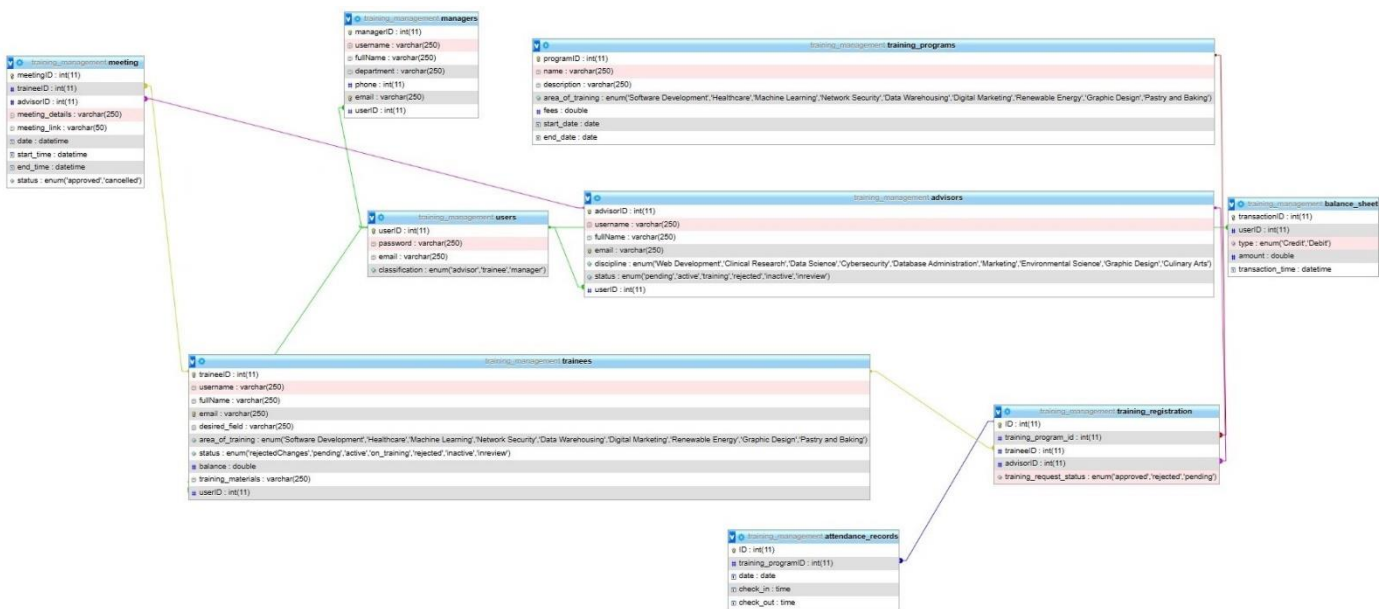
i-0ac2efbe2b3b2d324 (TMS-Flask-WebServer)
PublicIPs: 13.51.85.106 PrivateIPs: 172.31.39.17

All Tables in the Database

- The Database Schema



Database Schema Pt.1



Database Schema Pt.2 (Detailed)

7. The Architecture of AWS

Amazon Web Services (AWS) is the world's most widely adopted and comprehensive cloud, offering more than 200 end-to-end data center services globally. Millions of customers, including the fastest growing startups, largest companies, and leading government agencies, are using AWS to lower costs, increase agility, and innovate faster.

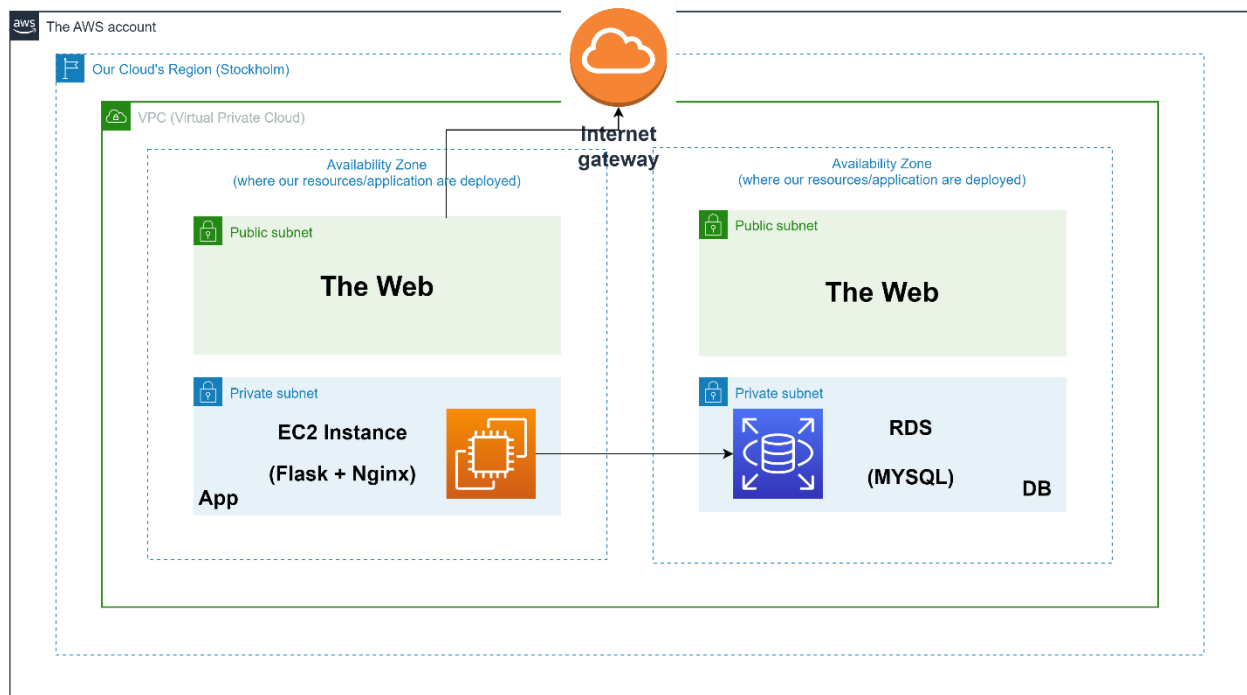


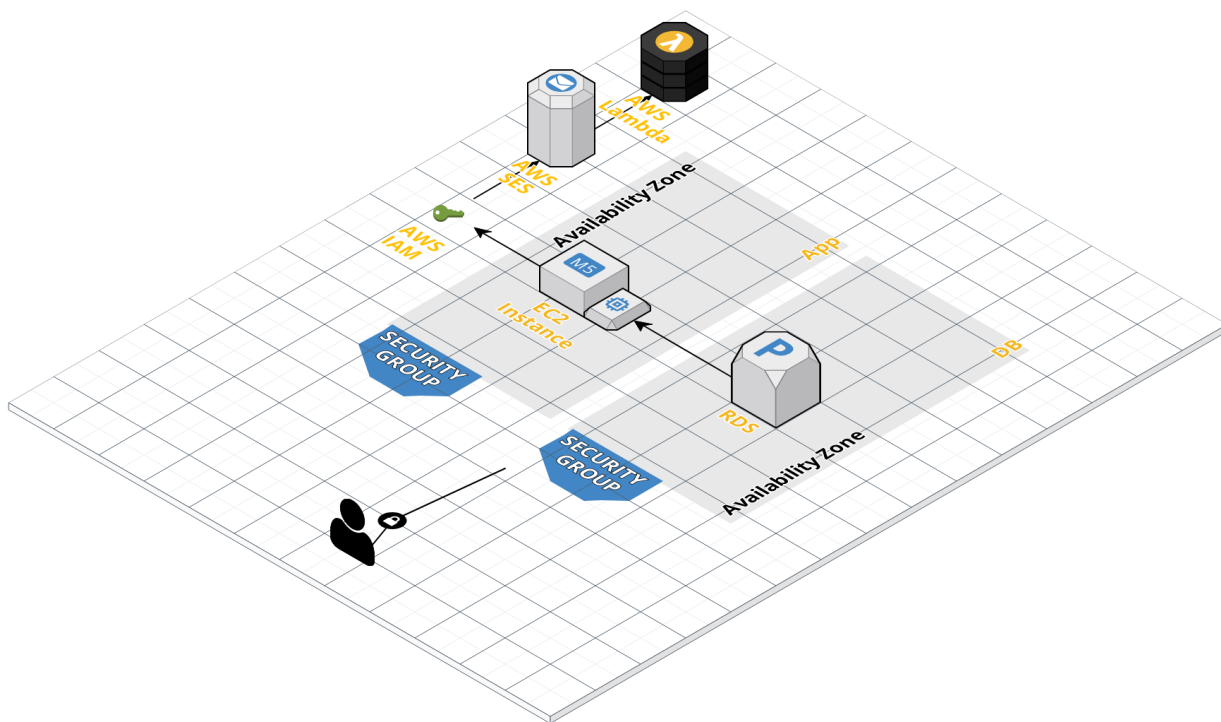
Figure4 - The Architecture of TMS on AWS

AWS is designed to be the most flexible and secure cloud computing environment available today. Our core infrastructure was built to meet the security requirements of the military, international banks, and other organizations that must adhere to strict confidentiality requirements. It is backed by a comprehensive set of cloud security tools, with more than 300 security, compliance, and governance services and features. AWS supports 98 security standards and compliance certifications, and all 117 AWS services that store customer data offer the ability to encrypt that data.

Having said that, the AWS architecture for TMS is designed as follows:

1. **EC2 Instances:** TMS requires one EC2 instances to host all its needed components. This instance is be deployed in a sperate availability zone than RDS for high availability and fault tolerance.
2. **RDS (Relational Database Service):** TMS uses RDS to store and manage the MySQL database. The RDS instance is configured with appropriate storage capacity, instance type, and backups to ensure data persistence and reliability.
3. **VPC (Virtual Private Cloud):** TMS is deployed within a VPC, which provides network isolation and security. It allowed us to define subnets, route tables, and network access control lists (ACLs) to control inbound and outbound traffic.
4. **Subnets:** TMS utilizes both public and private subnets. Public subnets are associated with internet gateways, allowing external access to TMS's components. Private subnets, on the other hand, are not directly accessible from the internet. Thus, we stored our RDS database and EC2 instance to provide an additional layer of security
5. **Security Groups:** we used security groups to control inbound and outbound traffic for the EC2 instances and RDS. They act as virtual firewalls, which allowed us to specify allowed protocols, ports, and source IP ranges.
6. **IAM (Identity and Access Management):** TMS uses IAM to manage user access and permissions to AWS resources. It allowed us to verify users and define fine-grained access controls for all our user types, the managers, advisors, and trainees.
7. **SES (Simple Email Service):** SES is utilized for sending email notifications to users, such as trainees and advisors. It provided a reliable and scalable email delivery service.
8. **Lambda:** AWS Lambda functions were used to perform serverless computing tasks within TMS. We used it to trigger automated email sending actions.

These are some of the key components and services that are a part of the AWS architecture for TMS. The figure below showcases how components are related to each other.



A 3D and More Detailed Representation of TMS's Cloud Architecture

8. Deployment on the Platform

Here are the steps in which we deployed TMS on AWS:

1. Launched the Instance: In the AWS Management Console, we selected EC2 and chose "Launch Instance."
2. Chose Instance Type: we chose the instance type that met our requirements in terms of CPU, memory, storage, and networking capacity.
3. Configured the Instance: we set up additional configuration options, such as the number of instances, network settings, security groups, and storage options.
4. Added Storage: we specified the size and type of storage for our instance.
5. Configured Security Groups: we defined the inbound and outbound rules for the security group associated with our instance. This controlled the network traffic that could reach our instance.

6. **Review and Launch:** Double-check your instance configuration and settings, and then launch the instance.
7. **Key Pair Selection:** we created a key pair to securely connect to your instance from our local terminal.
8. **Launch Status:** Once our instance was launched, we received a confirmation where we were able to view the status and details of our instance.

9. User Support

User service in TMS is more than just the time in which trainee spends interacting with the provider. The end-to-end user experience includes many touchpoints and interactions between the trainee and the TMS application. Each of these interactions provide an opportunity for the TMS application to influence the user's perception about the quality that they are receiving.

- 1- Trainees will have the comfort in using this application: Trainees can apply for training and can select one or more training programs based on their desired field and area of training, they can fill-in training attendance forms. Also request for a meeting with his advisor. In requesting a meeting, TMS must resolve any conflicts.
- 2- Manager is the person who has a training background: They can review training requests and decide whether to accept them or not based on a given criteria, the manager registers the trainee as an authorized user and sends a unique trainee ID for future login, Whenever the trainee wants to login again, the manager first verifies the entered trainee ID.
- 3- Easy way to update the data in cloud: The trainee or the advisor cannot update or perform any modification on the data stored in the cloud. Only the manager has the rights to update the cloud's data to ensure data security.
- 4- No need to worry about the data management: Cloud application stores trainees' records, performs computation and any needed functionality such as

trainee-advisor appointment management, notifications, emailing, searching, advertising such as pop-up general medical advices.

- 5- Advisor has the right to choose: An advisor manages his own trainees, e.g., accepts and schedules meetings with his trainees, follows up his trainees. He also can see a list of his trainees such as new trainees, on training trainees, etc. Also, he can send notifications or emails to his trainees as needed.

10. Conclusion

TMS is a comprehensive platform for trainees, advisors, and managers to efficiently manage training programs. With features such as registration, document upload, training selection, meeting scheduling, and emailing, the system offers a seamless user experience. Leveraging AWS services like EC2 instances, RDS, IAM, and SES, the system ensures scalability, security, and reliable data storage. Its three-tier architecture enables efficient data management, analytics, and tracking of user activities. Overall, TMS streamlines training processes, enhances collaboration, and delivers a user-friendly interface for effective training management.

References/Links

[The GitHub Repo for TMS](#)

<http://www.tmstraininghub.click/login>