

ADAPT Community Network Systems Design Document

August 27, 2025

Project Developers:

Darien Miller - Web Developer

Wilma Rodriguez - Project proposal & UI/UX developer

Revision History

Date	Version	Author	Changes

Contents

SECTION 1

Project Overview.....	3
1.1 Project name and Description.....	4
1.2 Stakeholders.....	??
1.3 Assumptions.....	??
1.4 Core-features.....	??

SECTION 2

Functional Requirements.....	3
2.1 Performance.....	4
2.2 Scalability.....	??
2.3 Security.....	??
2.4 Reliability.....	??
2.5 Maintainability.....	??

SECTION 3

System Architecture.....	3
3.1 High-Level Diagram.....	4
3.2 Technology Stack.....	??
3.3 System Components.....	??

SECTION 4

User interfaces.....	3
4.1	

SECTION 5

Module Design.....	3
4.1 Authentication & Authorization Module.....	??
4.2 Class Management Module.....	4
4.3 Reporting Module.....	??
4.4 Attendance Recording Module.....	??

SECTION 6

Database Design.....	3
5.1 ER Diagram.....	4
5.2 Schema Design.....	??
5.3 Indexes.....	??

SECTION 7

API Design.....	3
6.1 Endpoints (HTTP Method, Request/Response Format, etc).....	4
6.2 Authentication.....	??
6.3 Authorization (roles, resources, permissions).....	??
6.4 Rate Limiting.....	??
6.5 Error Handling.....	??

SECTION 8

Security Design.....	3
7.1 Authentication/Authorization.....	4
7.2 Data Encryption.....	??
7.3 Security Auditing.....	??
7.4 Vulnerabilities.....	??
7.5 Security Stack.....	??

SECTION 9

Deployment Architecture.....	3
8.1 Environment Setup (Development, Staging, Production).....	??
8.2 Scaling Strategy.....	??
8.3 Monitoring Stack.....	??

SECTION 10

Testing Strategy.....3

9.1 Unit Testing.....??

9.2 Integration Testing.....??

9.3 Acceptance Testing.....??

9.4 Performance Testing.....??

9.5 Security Testing.....??

9.6 Automated Testing.....??

SECTION 11

Maintenance and Monitoring.....3

10.1 Logging.....??

10.2 Alerting.....??

10.3 System Health Montioring.....??

10.4 Error Tracking.....?

SECTION 12

Backup and Recovery.....3

11.1 Backup Strategy.....??

11.2 Disaster Recovery.....??

SECTION 13

Risks and Mitigation.....3

12.1 Technical Risk.....??

12.2 Mitigation Strategies.....??

SECTION 14

Future Enhancements.....3

13.1 Roadmap.....??

13.2 Scalability Considerations.....??

1 Project Overview

1.1 Project Name:

This project will be called the ADAPT Community “Student Database”.

1.2 Project Description:

The ADAPT Community Networks student Database will track student course registration, attendance, and community participation. Student profiles will provide essential information, allowing for the compliance of transfer of care and community engagement protocols. Here is a summary of the feature this project will support:

1. Taking daily attendance for our classes
2. Adding new students to our classes
3. Creating and removing classes from our schedules
4. Submitting weekly schedules
5. Translating class and student data to pre-styled

As of now, all of the responsibilities above must be done manually, which is a common source of stress for the specialists. At its completion, the project will allow for the automation for every single responsibility in one place.

1.3 Stakeholders:

The student database project was introduced on November 11, 2022, by Education Specialist Wilma Rodriguez. The project received is supported by supervising management team: Peter Cobb, Shaniece Frank and Chevonne Brown. In 2025 the project evolve into a collaboration with computer specialist Darien Miller.

1.4 Assumptions:

Initially, this project was started in May, with a release sometime in September. Unfortunately, due to hardware failure, and a priority on another ADAPT project for jeopardy score keeping, the completion date for the first version had to be pushed back. With a new laptop, and the scorekeeping project completed, there can now be clear timelines for each milestone. Moving forward, I will proceed with the following assumptions:

Resource Assumptions:

- Serverside/Backend Development will be handled primarily by Darien Miller, with UI/UX and general aesthetics handled Wilma Rodriguez.
- Specialists will create test classes alongside test students for validation.
- Specialists will work within a stable and easy to use environment.

Technology Assumptions:

- For the back-end, all data processing will be done with ASP.NET, with views being handled By HTMX and Alpine.js for interactivity.
- Class and student data will be exported to the official specialist, pre-styled excel sheet.
- All class and student data will be managed by a relational database, most likely PostgreSQL.

Scope Assumptions:

- Version 1.0 will include the ability to add classes, students, and export to excel sheets.
- Additional features like the admin dashboard will be released with later versions of the app.

Risk Assumptions:

- No other projects will interfere with the development timeline of this project
- Staff will be able to start using it as soon as the app is released.

Dependency Assumptions:

- Specialists will access the application through modern browsers (Chrome, edge, Safari, etc).
- Application will be hosted on existing IT infrastructure.

Milestones:

- Version 1.0 release date - **10/25**
 - - Features: Add classes, Add students, export them to excel
- Version 1.1 release date - **11/1/25**
 - - Features: Admin page for supervisors to view specialists classes.

1.5 Core-features:

This project as mentioned before will consolidate every single responsibility into a single app to reduce complexity and improve efficiency. Here is a summary of the features:

- Add classes
- Add Students
- Export classes and files to excel
- Admin page for supervisors to view and edit classes/schedules
- Create weekly schedules

2 Functional Requirements

2.1 Performance:

Response Times

- The ideal response times for adding new students, adding new classes, and adding students to classes should ideally fall within 1-2 seconds assuming a solid internet connection.
- Generating the excel spreadsheet that contains all of the specialists classes, and students who attended for a specific month should ideally take no longer than 5 seconds.

Throughput/Concurrency

- This application should be able to very easily handle the load of all of the specialists using it at the same time, as there are only around 25 of us with classes and students to add.
- Max usage for the app should be at the end of each day, and near to the deadline for the monthly attendance sheet submission.

Transaction Rates

- During peak periods when the app is being used the most, a good benchmark for the amount of transactions per minute is 100, combining classes and students being added.

Client-Side Performance

- For solid internet connections, classes and students for each class should all load in at most two seconds, ideally under one second.
- Front-end interactions should be instant, and not really too much on re-loading the page to fetch new information.

2.2 Scalability:

Due to the internal nature of this application, and the limited number of ADAPT specialists using the application at any given time, **vertical scaling** will be the preferred scaling method. Horizontal scaling would require deploying the source code on multiple machines, and I currently do not have access to such resources, but even if I did, it would be overkill for this project. With vertical scaling, I can deploy a single instance to a cloud provider, and focus entirely on expanding it should the need ever come up for my team.

2.3 Security:

For security, I plan to implement all of the best practices for protecting log in, as well as other methods to satisfy HIPAA requirements and protect information about people-supported. Methods include:

- Anti-CSRF token for log in.

- Passwords will be hashed (scrambled) before being stored in the database.
- Session usage on the backend to maintain login/logout flow.
- Refresh tokens to prevent users from getting signed out too quickly.
- Sign up restricted to users with @adaptn emails.
- HTTPS for all data transmission.
- 2FA (2-factor authorization) using the authenticator app on mobile.
- Encryption for people-supported names for database.

2.4 Reliability:

This app should be as reliable as possible, and in order to meet that threshold, I will guarantee the app will meet several benchmarks set by me.

Expected uptime:

Uptime is defined as the amount of time the website will actually be up and running, in contrast to downtime which is how long websites stay down. With this project being hosted on the free tier on the cloud service Azure, there will some down time as the website will “sleep” when not using , but it will only affect the first initial request, with each subsequent one operating normally.

Data Durability:

Database will be backup every other day, with timestamps marked to date each backup

Disaster Recovery:

- Disaster Recovery is defined as the ability for an application to recover valuable data should there ever be a breach, or some server related crisis that causes the application to shut down while in use. In such a scenario, there are some tools I have available to address them now, and better tools in the future to more effectively deal with it.
- As mentioned before, my cloud SQL database (Azure storage) will be backed up daily, with rollback to earlier versions possible should anything go wrong. Excel Attendance exports for each specialist can also function as back up data since it contains the classes and attendance marks for each student.
- In the case where the project has to be redeployed, moved to another cloud service, or even run from the source (this laptop) process will be made much simpler due to the application being containerized using docker.

Failover Systems:

- A failover describes the process of the primary server automatically switching to another backup server should the primary server fail. Due to this project being hosted on a single cloud service with a single instance, the system must be manually restarted through the cloud service provider.
- As the project grows and becomes more widely adopted, load balancing will be adopted to distribute client requests to different servers and instances rather than just one as I am currently doing now.

2.5 Maintainability:

Here is a list of active practices that will be done to keep this project as maintainable as possible:

Codebase Standards:

- Every single file in this project will follow C# naming and styling conventions for clarity and consistency.
- Razor pages will be organized according to standard project hierarchy.
- For Class names, PascalCase will be used, and for variables, camelCase will be the default

Dependency Management:

Every single fil- TBC

Documentation:

TBC

Testing:

For the purposes of testing, unit testing and integration testing will both be used to ensure every aspect of the application has accounted for every single possible edge case.

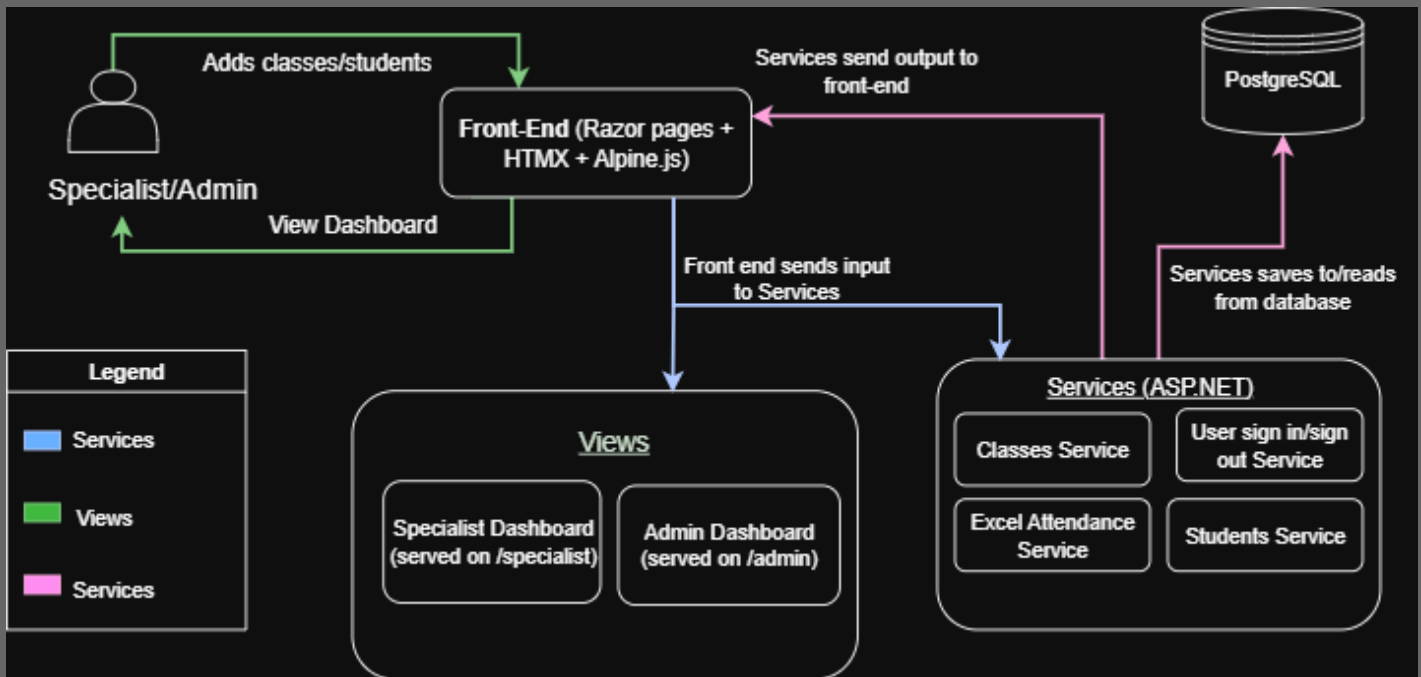
- **Unit Testing** is a type of test that checks each individual method, function, class, etc in isolation to ensure the expected outcome is returned.
- **Integration Testing** is a type of test that checks to see if different layers of the program interact with each other as intended, and also to ensure database connectivity is functioning. This type of test would test to see if passwords, usernames, and classes for example are all actually being saved to the database.

Logging:

3 System Architecture

3.1 High-Level Diagram:

Here is the current system architecture diagram for the Attendance Tracker:



3.2 Technology Stack:

For my Attendance Tracker, I decided to forego a full fledged JS Framework in favor of a more simple, server rendered front-end due to my previous project getting bloated with Vue.js. Though my experience was certainly not negative, I would like to have a smaller, simpler to maintain codebase for this project, which works well considering this project is smaller in scope than my previous one.

Front-end:

- [HTMX](#). This is an extension
- Alpine.js
- CSS
- Razor Pages

Back-end:

- C#
- ASP.NET

Database:

- PostgreSQL

Infrastructure:

- Docker
- Azure

3.3 System Components:

4 User Interfaces

Homepage

The homepage will require the creation of a username and password. Two-factor authentication (2FA) should also be enabled for added security.



INSTRUCTOR PROFILES



**WILMA
RODRIGUEZ WILMA**
EDUCATION SPECIALIST

Wrodriguez@ucpnyc.org



TRISCA LAGHARI
ASST. MANAGER

hello@reallygreatsite.com



CHUN HEI KIM
SENIOR SPECIALIST

hello@reallygreatsite.com



ESTELLE DARCY
SENIOR SPECIALIST

hello@reallygreatsite.com



JIARA MARTINS
JUNIOR SPECIALIST

hello@reallygreatsite.com







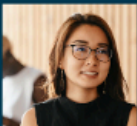

MARGARITA PEREZ
INTERN

hello@reallygreatsite.com

Instructor Profile – User Instructions

- Each instructor profile displays the following information:
 - Profile Photo
 - Full Name
 - Email Address
- To view an instructor's personal profile:
 1. The user locates the instructor of interest.
 2. The user clicks on the instructor's profile photo.
 3. The system opens the instructor's detailed profile page, where the user can find course information, instructor schedule, upcoming community trips, student caseload (1:1's and students that the instructor works closely with).
 4. Attendance record,
 5. Student progress records
 6. Teaching resources

STUDENT PROFILES

 <p>JOHN S. SMITH LOCATION: FLUSHING ADAPT E-mail</p>	 <p>TRISCA LAGHARI ASST. MANAGER hello@reallygreatsite.com</p>
 <p>AMY PARIS LOCATION: FLASHING ADAPT Email</p>	 <p>ESTELLE DARCY SENIOR SPECIALIST hello@reallygreatsite.com</p>
 <p>JIARA MARTINS JUNIOR SPECIALIST hello@reallygreatsite.com</p>	 <p>MARGARITA PEREZ INTERN hello@reallygreatsite.com</p>

- Each student profile displays the following information:
 - Profile Photo
 - Full Name
 - Email Address





John S Smith
PROGRAM: FLUSHING

John is an active participant in the Flushing program. He enjoys interacting with peers and supporting staff. Currently, John is pursuing his GED. He also enjoys participating in art and educational activities and has a deep interest in music.

Evaluation

Community Information

Weekly schedule

Attendance



Person Supported bio

The student profile tabs:

Evaluation

Community details

Current courses

STUDENT PROFILES



JOHN S. SMITH

LOCATION: FLUSHING
ADAPT E-mail



TRISCA LAGHARI

ASST. MANAGER
hello@reallygreatsite.com



AMY PARIS

LOCATION: FLASHING
ADAPT Email



ESTELLE DARCY

SENIOR SPECIALIST
hello@reallygreatsite.com



JARA MARTINS

JUNIOR SPECIALIST
hello@reallygreatsite.com



MARGARITA PEREZ

INTERN
hello@reallygreatsite.com

On route “/student-profiles”



Wilma C. Rodriguez

Education Specialist

Instructor Bio Courses Weekly Schedule
Attendance

Locations

Flushing >

Lawrence >

Elmwood >

About Me

 Edit

 Share

 Print PDF

Wilma Rodriguez is an Education Specialist at Adapt Community Network. In the Summer of 2019, during her internship with the Ford Foundation, she worked alongside the team of the Office of the President. In this role, she supported in developing disability awareness events and promoted disability inclusion in the grant-making process. As part of the Adapt family, she uses personal, academic, and work experiences to design innovative disability studies courses and assist students in preparing for the TASC (GED) examination.

Weekly Schedule

Monday


Tuesday

Wednesday

Thursday

Friday

On route “/instructor-profile”



Evaluation Community Info Weekly Schedule



John S Smith

PROGRAM: FLUSHING

John is an active participant in the Flushing program. He enjoys interacting with peers and supporting staff. Currently, John is pursuing his GED. He also enjoys participating in art and educational activities and has a deep interest in music.

Attendance



Person Supported Bio

This section can be used to provide more detailed information, notes, or updates regarding the person supported. It offers a flexible space for staff to document progress, observations, and other relevant biographical details that might be helpful for providing the best support.

For example, you could include:

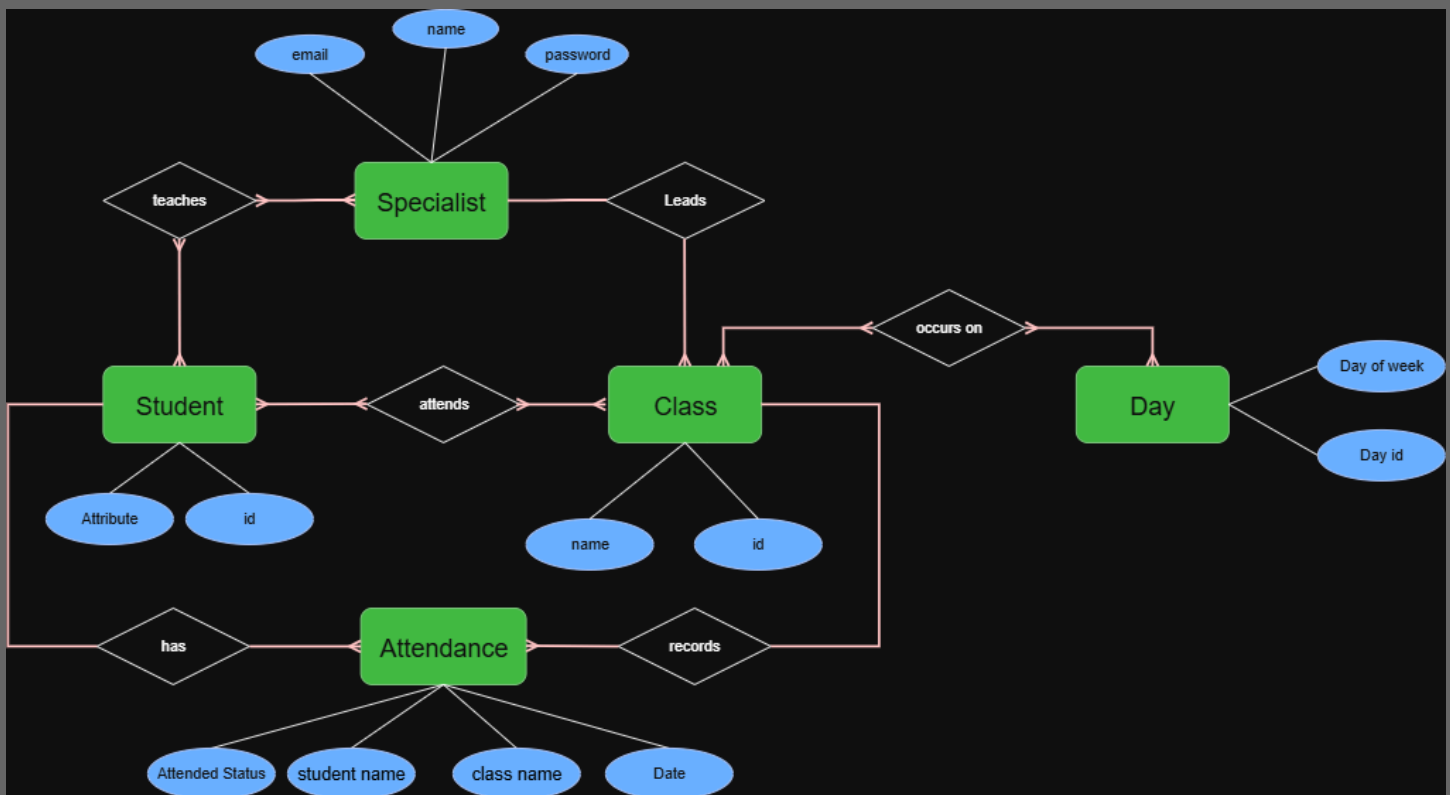
- Recent achievements and milestones.
- Specific support strategies that have been effective.
- Updates on personal goals and aspirations.
- Notes from recent check-ins or meetings.

On route “/person-supported”

6 Database Design

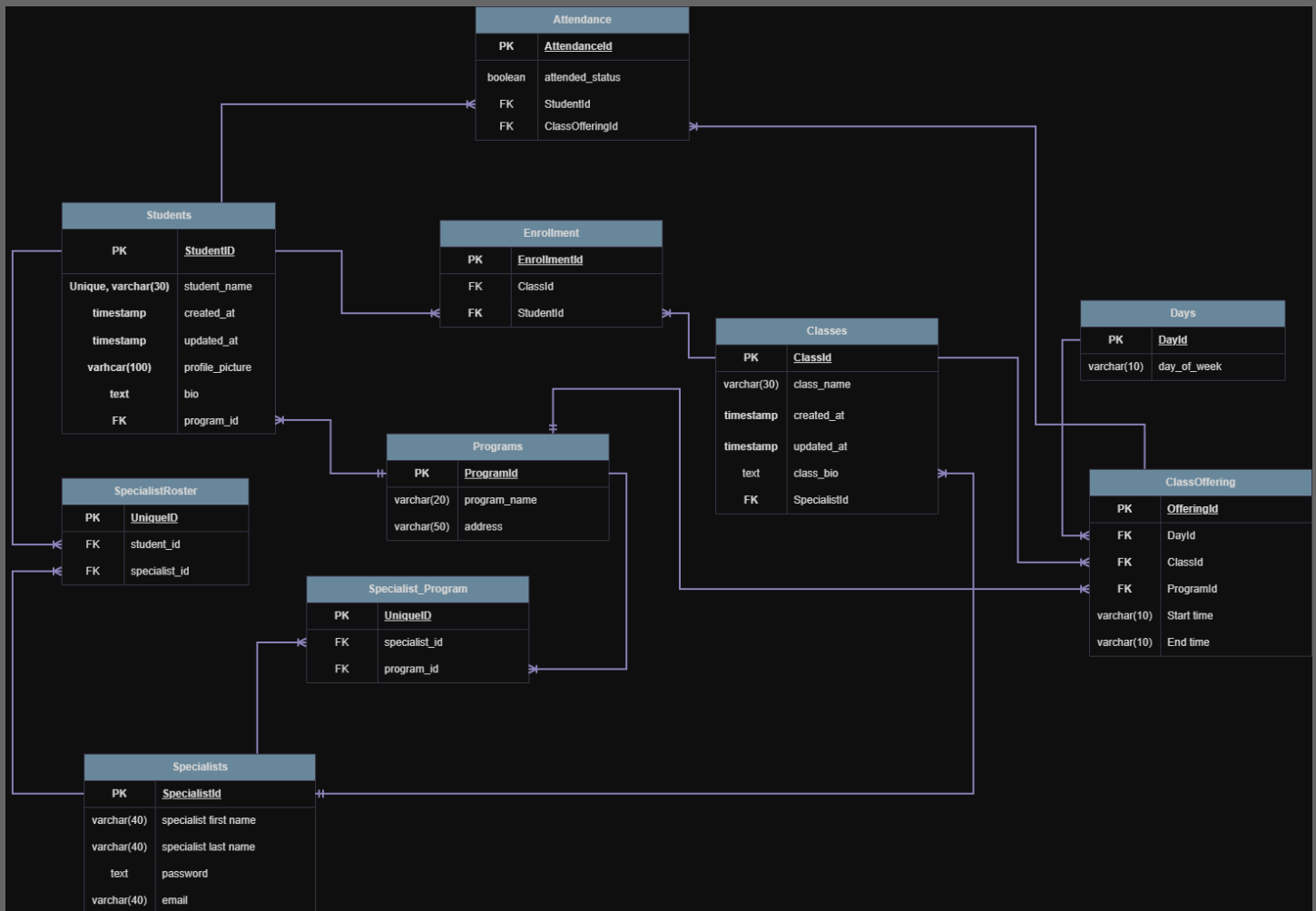
6.1 ER Diagram:

Here is the ER diagram for the database to be used for this project. For the initial release, there will be 5 tables: One for Specialists, one for students, one for classes, one for days of the week, and one for attendance. Students and classes will have a **many-to-many** with each other, classes will have a **one-to-many** with specialists, attendance will have a **one-to-many** relationship with students and Classes, and finally, classes and days will have a **many-to-many** relationship with each other.



6.2 Schema Diagram:

Here is the schema diagram for the actual database tables. There are 10 tables, with many also serving as junction tables for two other tables.



The following is a description of the database tables, and the relationships they have with each other:

Students: Students will hold all of the necessary data of all the students inputted by every single specialist using this application. Students will have a name, a bio, a profile picture, and their day program. Four other tables will interact with the students table: **Enrollment**, **Class Roster**, **Program**, and **Attendance**.

Specialists: Specialists will contain all of the information that specialists will need to use this app. Specialists will have first and last name, an email, and a password to log in with. The Specialists table will interact with three other tables: **Class Roster**, **Specialist program** and **Classes**.

Enrollment: Enrollment will hold the unique id of both the Student table and Class table. It will allow users to know which classes people-supported are enrolled in. This table will interact with Two other tables: **Classes** and **Students**.

Attendance: Attendance will allow specialists to record attendance for each of their students for every single class, for all program sites. Attendance will have a flag for 'P' or 'A', the unique ids of the student, class and day program, and the timestamp for when the attendance was taken. This table will interact with three other tables: **Students**, **Classes**, and **Programs**.

Specialist Roster: Specialist Roster will hold the roster of people-supported that each specialist has, and which specialists each person-supported attends. Class Roster will hold the unique id of both the student and the specialist. This table interacts with two other tables: **Specialists** and **Students**.

Class Schedule: Class Schedule will contain all of the instances of each class that each specialist will have. Since specialists can have the same class at different times and different sites, this table will hold all copies of a single "base" class. Class Schedule will have the start and end times of the class, the id of the program where the class is being taught, the id of the base class, which day of the week it is. Three other tables reference this table, them being **Class**, **Program** and **Days**.

Classes: Classes will hold all of the information for the classes we specialists have to offer. Classes will have the class name, and the unique id of the specialist teaching the class. Four other tables reference this table, them being **Attendance**, **Enrollment**, **Specialist** and **Class Schedule**.

Programs: Programs will contain the names of all of the day hubs, or program sites for our people-supported, which is where our classes are hosted. Programs will have a name, and an address. Four other tables reference this table, them being **Attendance**, **Students**, **Class Schedule**, and **Specialist Program**.

Specialist Program: Specialist Program will tie together programs and specialists, determining each program every specialist goes to, and all of the specialists that work at a particular program. This table will have the unique id of each specialist, and each program. As mentioned before, two tables reference this table, them being **Specialists** and **Programs**.

Days: Days will simply hold the 5 days of the week when classes are held. Days will just hold on field, that being the name of the day. It will be referenced only by Class Schedule, so each class will know what day it's held on.

6.3 Indexes:

In database programming, an "Index" is a trade off that allows data to be retrieved significantly faster than normal, but it makes sending data to the database slightly slower than normal. Indexes should ideally be used when dealing tables that has generally met two conditions:

1. It has an enormous number of rows, in the tens of thousands or more.
2. The data from the table is retrieved extremely frequently across many users.

When looking at these two conditions, the table that would definitely benefit from be indexed would be the Attendance table, as it would accrue dozens of attendance per day, for each specialist, for each person-supported. Every other table would only grow up to a certain point, since we only have a finite number of specialists, people-supported, classes and day programs.

6.4 Transactions:

When dealing with SQL databases, inevitably “transactions” will need to be done depending on how much data there is to handle, and how many tables the database will contain. A transaction will allow multiple database operations to be done in one clean action, rather than one by one. For example, if a student needs to be added to a new class, that represents two different operations: Adding a new student to the student table, and then connecting that student to a pre-existing class. Instead of doing them separately, transactions allow both actions to be combined into one larger one! Here are cases in which transactions would help maintain database integrity:

Matching a student to a Specialist:

- When adding a new student, a new relationship between that student is created with the specialist that added that student. Both the Students and SpecialistsRosters tables are updated.

Taking Attendance for a student:

- Taking attendance for a student involves creating a new relationship between the student table, the class table, and the program table. This of course requires multiple steps.

Class creation and scheduling:

- Creating a new class, and variations of that class for different sites and times involves creating a new relationship between the Class table, Days Table, and Program table

As you can see, with actions like these, transactions allows multiple procedures to be packaged into one larger one, which can then be called as one unit.

7 API Design

7.1 Endpoints:

Due to the Attendance tracker using razor pages rather than an API, most of the following endpoints will be returning pages rather than JSON. Alongside the project however, I will design a lightweight API to parallel the razor page web endpoints.

The endpoints will follow the typical REST conventions:

GET - Retrieve student/specialist/class data

POST - Create new student/specialist/class data

PUT - Update student/specialist/class data

DELETE - Remove student/specialist/class data

As mentioned before, each endpoint will return a razor page, protected by session data and other means of protection.

Student endpoints

Endpoint	HTTP Method	Description
/api/v1/students	GET	Retrieves all students
/api/v1/students/{id}	GET	Retrieves a single student with a specific id.
/api/v1/students	POST	Adds a new student to the database
/api/v1/students/{id}	DELETE	Deletes a student with a certain id.
/api/v1/students/{id}	PUT	Updates a student trait (profile picture, bio, name, etc)

Class endpoints

Endpoint	HTTP Method	Description
/api/v1/class	GET	Retrieves all classes
/api/v1/class/{id}	GET	Retrieves a single class with a specific id.
/api/v1/class	POST	Adds a new class to the database
/api/v1/class/{id}	DELETE	Deletes a class with a certain id. Will also delete all instances of that class for all sites.
/api/v1/class/{id}	PUT	Updates a class trait (Class name, bio, etc)

Class Offering endpoints

Endpoint	HTTP Method	Description
/api/v1/class-offering/{classid}	POST	Adds a new class instance/offering for a main class with selected id.
/api/v1/class-offering/{id}	GET	Retrieve a specific class offering.
/api/v1/class-offering	GET	Retrieve all class offerings.
/api/v1/class-offering/{id}	DELETE	Deletes a single class offering (math class on tue/thur at 3PM, etc)
/api/v1/class-offering/{id}	PUT	Updates a class offering trait (Class name, bio, etc)

Specialist endpoints

Endpoint	HTTP Method	Description
/api/v1/specialist	GET	Retrieves all specialists
/api/v1/specialist/{id}	GET	Retrieves a single specialist with a specific id.
/api/v1/specialist	POST	Adds a new student to the database
/api/v1/specialist/{id}	DELETE	Deletes a specialist with a certain id.

/api/v1/specialist/{id}	PUT	Updates a specialist trait (profile picture, bio, name, etc)
-------------------------	-----	--

Attendance endpoints

Endpoint	HTTP Method	Description
/api/v1/attendance?limit=x&offset=y	GET	Retrieves all attendance up to {limit} and skips the first {offset} rows
/api/v1/attendance/{id}	GET	Retrieves a single attendance record with a specific id.
/api/v1/attendance	POST	Adds a new attendance record to the database
/api/v1/attendance/{id}	PUT	Changes attendance record from present to absent and vice versa

Program endpoints

Endpoint	HTTP Method	Description
/api/v1/program-sites	GET	Retrieves all program sites
/api/v1/program-sites/{id}	GET	Retrieves a program site with a specific id.
/api/v1/program-sites	POST	Adds a new program site.

7.2 URL Schemas & Naming Conventions:

- For my endpoints, they will all be lowercase, with kebab-case.
- Resources are associated with the api suffix.
 - /api/classes/{classId}/students -> returns students for a registered class
 - /api/specialists/{specialistId}/classes -> returns classes that a specialist teaches

GET /api/students/123
 GET /api/classes/12/students
 POST /api/attendance

7.3 Request/Response Structure:

Here are some examples of the JSON format I will be using for requests and responses that the future API will use. Note however, that due to the server-driven model I am using, data will be sent directly from services to razor pages without an API layer.

**Example: Add new Person-Supported
POST /api/v1/students**

Request Body:

```
{  
  "name": "John Doe",  
  "bio": "Loves computer skills and art",  
  "profilePicture": "https://example.com/images/john.jpg",  
  "programId": 3  
}
```

This is what will be returned.

Response:

```
{  
  "studentId": 201,  
  "message": "Student created successfully"  
}
```

Example: Submit Attendance

POST /api/v1/attendance

Request Body:

```
{  
  "studentid": 201,  
  "classOfferingId": 42,  
  "attended_status": true,  
}
```

Response Body

```
{  
  "attendancelid": 1000,  
  "message": "Attendance recorded"  
}
```

7.4 Authentication and Authorization:

Authentication: As mentioned before, the app will be secured with anti-CSRF and sessions, while the API when it's fully implemented will be secured with JWT (JSON Web Token). When a user has been properly authenticated through 2FA (phone, email, microsoft authenticator), they will be allowed to have access to their dashboard.

Authorization: People-supported names will be encrypted, and only accessible to specialists and admins. Future enhancements will include role-based access control (RBAC) to distinguish between admin and regular specialists.

7.5 Rate Limiting:

In software engineering, rate limiting is a technique to reduce the amount of requests that are sent to and requested from the server. This is VERY important to preventing DOS/DDOS, which locks out the intended users from accessing their data. All routes will be rate limited, with POST/DELETE/PUT routes being more aggressively limited to 10 requests a minute, and GET routes being more lenient, and dependent on how much data is being returned per GET route. GET routes returning more data will be more aggressively rate limited, and GET routes returning less will be more lenient.

7.6 Error Handling:

This is a typical response from the server to display an error message. The structure is of a typical JSON object.

```
{  
  "status": 400,  
  "error": "InvalidRequest",  
  "message": "Student ID not found."  
}
```

For the server driven razor pages, errors will be sent as HTML fragments to be displayed directly onto the page.

Here are the common HTTP codes I will use for responses to the client.

Code	Meaning
200	Success
201	Resource created
400	Bad request
401	Unauthorized
404	Resource not found
500	Internal Server Error

8 Security Design

8.1 Authentication and Authorization:

This is a typical response from the server to display an error message. The structure is of a typical JSON object.

Authentication Method:

For the Attendance Tracker, it will run a server driven model where users are authenticated using sessions, which will be combined with other best practices to secure user data.

- Users log in with credentials validated against the database.
- Upon success, a user session is created on the backend, which is secured with anti-CSRF tokens to prevent unauthorized users from making requests on behalf of a valid user.
- Every time a request is made from a logged in user, their session id and anti-CSRF token is passed to the server to validate requests to and from the server.

Authorization Model:

For admins, this project will include Roles-based Access control (RBAC), so they are able to exert a greater level of control over specialists.

These roles will include:

- **Admin:** Full control over users, programs, attendance records.
- **Specialist:** Manage attendance for assigned classes.
- **Student/Person-Supported:** Can view personal attendance records and the classes Specialists teach.

Role permissions are enforced at the API route and database query level.

OAuth Integration (optional):

In the future, the web app will allow users to use google accounts to log in.

8.2 Authentication and Authorization:

Data in Transit: All requests to the server will be made over **HTTPS (TLS 1.3)** instead of plain HTTP to prevent interception or man-in-the-middle attacks. The former does not do this, which means data is entirely readable to these kinds of attacks.

Data at Rest: For the application, sensitive fields (people-supported names) are encrypted using **AES-256**, with passwords in the database being hashed using **bcrypt (library to scramble passwords)**. Database credentials and environment variables are stored securely in the cloud using .env files and server-level secrets.

8.3 Security Auditing:

Event Logging: All authentication events, data modifications done by every specialists (Creating, Reading, Updating and Deleting resources), and failed login attempts are logged, which allows server admins to monitor website traffic to seek out anomalous or suspicious activity.

Audit Trails: Each attendance record that is created or modified will provide the name of the responsible user and the timestamp of when it happened.

Monitoring: This app will include Integration with monitoring tools such as LogRocket, AWS CloudWatch, or ELK stack for real-time anomaly detection and general user activity.

8.4 Vulnerabilities and Mitigations:

Potential Vulnerabilities	Description	Mitigation Strategy
SQL Injection	Insertion or “injection” of a SQL query via the input data from the client to the application.	Use SQL parameters queries or a SQL ORM. Sanitize user input data on backend.
XSS (Cross-site Scripting)	Hackers inject client-side scripts into web pages viewed by other users.	Sanitize all user input on front-end and back-end, escape data, and use security headers (CSP).
Cross-site Request Forgery (CSRF)	A valid user is tricked by an attacker into submitting a web request that they did not intend.	Use CSRF tokens for forms and SameSite cookies (Strict or Lax).
Broken Authentication	Attackers exploit weak infrastructure to impersonate legitimate users.	Rotate/invalidate session IDs, Enforce strong passwords, 2FA, hash passwords
Data Exposure	Accidently allowing sensitive information to appear in logs or API responses.	Mask or exclude confidential data from logs and responses.

8.5 Security Stack and Defense Layers:

Here is a description of every line of defense this application will use to protect user data.

First Line of Defense:

Client-side and Server-side Input validation of user information, authentication, HTTPS, CORS restrictions to prevent communication from unknown servers, rate limiting to mitigate DOS and DDOS attacks, and 2FA for user log in.

Second Line of Defense:

Authorization checks (requesting names of people-supported), role validation on back-end using Roles Based Access Control (RBAC), API gateway filtering, Session id validation.

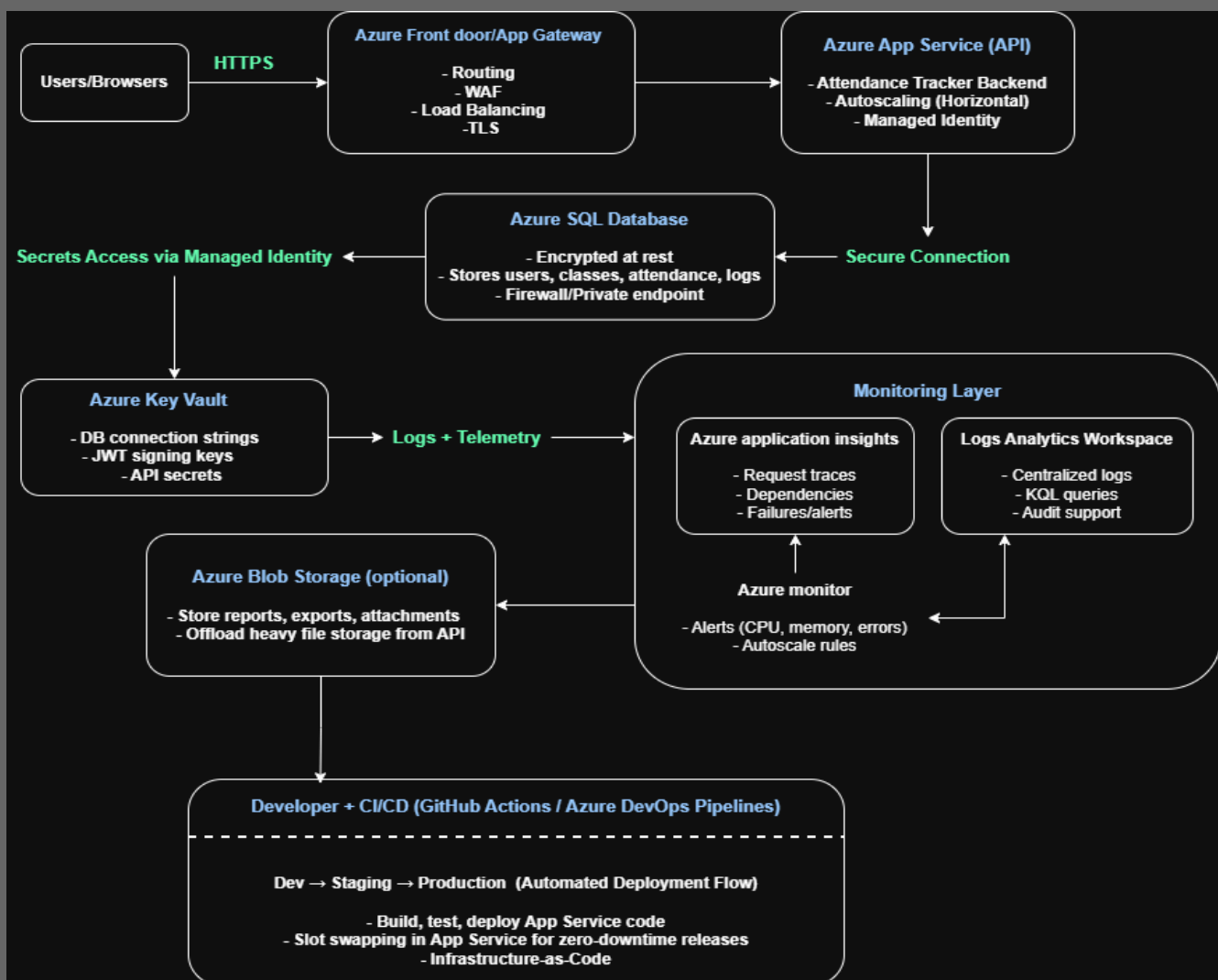
Last Line of Defense:

Database encryption of student names in database, audit trails, web-application Firewalls (WAF), Human Firewall, Regular database backup, regular security patching.

9 Deployment Architecture

9.1 Security Stack and Defense Layers:

The following diagram below will represent the architecture of the Azure cloud instance this project will be deployed to. Responsibilities are separated into different Azure architecture groupings, with the api service running under Azure App Service, Database storage and retrieval handled under Azure SQL Database, secrets being secured under Azure Key Vault, and logs being handled and monitored under Azure Monitor and Log Analytics Workspace.



9.2 Environment Setup:

My local environment setup will look as follows:

- Local PostgreSQL Server
- Local environment variables located in .env file
- Postman for API testing
- Hot-reload enabled backend via dotnet command line tool.
- Docker for containerized local testing and universal deployment
- Unit tests + integration tests using local mocks

When being staged for deployment, it will look as follows:

- Hosted in Azure with the same configuration as production
- Uses smaller SKUs to reduce cost
- CI/CD pipeline deploys automatically when project is pushed to github repo
- Full integration testing before production release
- Feature flags enabled for test features
- Connected to test database (Azure SQL - staging instance)

Finally, when fully deployed, it will use the following Azure services:

- Azure App Service Plan (Standard/Premium tier)
- Production Azure SQL Database
- Azure Front Door or Application Gateway for global routing + WAF
- CI/CD pipeline deploys on approval from staging
- Autoscaling rules enabled for horizontal scaling
- Continuous monitoring and alerting configured for security

9.3 Scaling Strategy:

For scaling purposes, I can choose between **Horizontal Scaling** and **Vertical Scaling**. The former adds more instances, and the latter expands the size and capacity of one instance. For the sake of this project, and how robust I would like it to be, I will choose horizontal scaling as this allows for load balancing, which ensures that if one instance fails, another one is automatically used to keep service up. In the cloud, App Service autoscaling will be enabled based on the following:

- CPU usage
- Memory consumption
- Request queue length
- Custom metrics from Application Insights

Multiple instances behind Azure Load Balancer or Azure Front Door will be used, guaranteeing a failsafe should one instance fail. This system is perfectly suitable for stateless API services, which will aid in the development of an API in the future.

9.3 Monitoring Stack:

Here are the following tools I will use to monitor logs and other related data:

Azure Monitor

- Real-time metrics for CPU, memory, response times
- Alerts for downtime, throttling, or scaling issues

Application Insights

- Request tracing
- Query performance
- Error logs
- User behavior analytics
- Custom events (attendance updates, login failures)