# INTRODUCTION

## What Is Nadyne V1

A software to profile customers and store their account data in a database. That data base will be compiled and sent as an attachment to email recipients. After comparison. These customers have to be validated against a list before they can be added to the email list.

# ROADMAP SUMMARY

1. Define project scope and goals: Determine the target audience, features to be included, and project timeline.
2. Develop user personas and user stories: Identify the different types of users that will be using the web application and their corresponding goals and tasks.
3. Design the database schema: Determine the structure of the database, including tables, columns, and relationships between them.
4. Set up the development environment: Install necessary software and tools, including Flask, Python, and any required dependencies.
5. Develop the front-end: Design and implement the user interface using HTML, CSS, and JavaScript, with the help of a front-end framework such as Bootstrap.
6. Develop the back-end: Build the server-side logic using Python and Flask, including user authentication and authorization, database integration, and API integration.
7. Test the application: Conduct thorough testing of the application to ensure that it meets the functional and non-functional requirements specified in the user stories.
8. Deploy the application to a web server: Use AWS Elastic Beanstalk or another cloud hosting platform to deploy the web application.
9. Monitor and maintain the application: Continuously monitor the application for bugs and performance issues and make any necessary updates or improvements.
10. Gather user feedback: Collect feedback from users and use it to improve the web application, adding new features or making changes based on user needs and preferences.

# ROADMAP DETAILED

## Project scope and goals

The Goal is to develop a web application called Nadyne that will register customers, store their account data in a database, and compile and send the data as an attachment to email recipients. The application will validate customer data against a list before they can be added to the email list.

The scope of the project includes the development of the front-end and back-end of the web application, the implementation of a database to store customer data, the integration of email APIs to send email attachments, and the integration of data validation APIs to compare customer data against a pre-existing list.

Non-technical goals: (user experience, scalability, and security) The web application should be user-friendly, scalable to accommodate future growth, and secure to protect customer data from unauthorized access or breaches.

## Target audience

The target audience/ users of Nadyne are people who are interested in a course advertised and proceeds to click on the shared app URL to begin their journey.

### Features to be included.

1. Customer Registration: Allow new customers to register by filling out a form that includes their personal information such as first name, other names, email address, phone number, country, city, date of birth, job type, course of interest, and start date.
2. Customer Validation: Validate new customer data against an existing email list to ensure that the customer is not already on the list. If the customer's data matches an entry on the list, mark them as validated.
3. Email List Compilation: Compile a list of validated customers' personal information and store it in a database for future reference.
4. Email List Comparison: Compare the compiled email list against a master list to ensure that there are no duplicates.
5. Email List Export: Export the compiled email list as an attachment in CSV or Excel format to be sent to recipients.
6. Admin Dashboard: Create an admin dashboard for authorized users to manage customer data, view compiled email lists, and manage validation status.
7. Data Analytics: Provide data analytics for customer data such as demographics, course preferences, and registration trends.
8. Security: Implement proper security measures to ensure the protection of customer data and prevent unauthorized access.
9. Responsive Design: Ensure that the web application has a responsive design that can be accessed on different devices such as desktop, tablet, and mobile.
10. User-Friendly Interface: Create a user-friendly interface that is easy to navigate and understand and includes clear instructions and feedback for users.

### Project timelines

1. Planning and Requirements Gathering: 2-4 weeks During this phase, the project team will work with stakeholders to identify the project's scope, goals, and requirements. This may involve conducting user research, defining user personas, and developing user stories.
2. Database Design and Setup: 2-4 weeks In this phase, the team will design the database schema, set up the database environment, and integrate it with the web application.
3. Front-end Development: 4-6 weeks During this phase, the team will work on developing the user interface and front-end functionalities such as customer registration, data validation, and data export.
4. Back-end Development: 4-6 weeks In this phase, the team will work on the back-end functionalities such as data storage, email list comparison, and admin dashboard.
5. Quality Assurance and Testing: 2-4 weeks This phase involves testing the web application for bugs and ensuring that it meets the project's quality standards.
6. Deployment: 2-4 weeks During this phase, the team will deploy the web application on the chosen hosting platform, configure APIs, and ensure that it is fully functional.

Overall, the estimated timeline for the development of the Nadyne web application is between 14-24 weeks, depending on the complexity of the project and available resources. It is important to note that these timelines are flexible and may change based on the project's progress and changes in requirements.

## User personas, use cases and user stories.

### User personas

1. New Customers: These are users who are not registered with Nadyne yet. They are looking to sign up for an account and provide their customer data.
2. Existing Customers: These are users who have already signed up for an account with Nadyne. They may need to update their account information, add new customer data, or manage their email preferences.
3. Admin Users: These are users with administrative privileges who can manage user accounts, view customer data, and export email lists.
4. Email Recipients: These are users who receive the email lists generated by Nadyne. They may need to validate the email addresses on the list against their own list of valid email addresses.
5. Guest Users: These are users who may not be registered with Nadyne but can access some limited functionality within the web application, such as browsing customer data or viewing the homepage.

### Use cases

1. User registration: Allow users to create a new account on the web application by providing their name, email address, and password.
2. User login: Allow users to log in to the web application using their email address and password.
3. Account management: Allow users to update their account information, such as their name, email address, and password.
4. Customer data collection: Allow users to input the following customer data: first name, other names, email address, phone number, country, city, DOB, job type, course of interest, Start date.
5. Customer data storage: Store customer data in a database on Amazon S3 and associate it with the user's account.
6. Email list compilation: Compile a list of email addresses from the customer data stored in the database.
7. Email list validation: Validate the email addresses in the compiled list against a pre-defined list of valid email addresses.
8. Email list export: Export the validated email list as a file attachment and send it to designate email recipients.
9. User management: Allow an admin user to manage user accounts, such as creating new accounts, updating account information, and deleting accounts.
10. User roles and permissions: Define user roles and permissions, such as admin, regular user, and guest, to control access to certain features and functionality within the web application

### User stories

1. As a new customer, I want to sign up for an account so that I can provide my customer data and receive emails from Nadyne.
2. As an existing customer, I want to update my account information, such as my name, email address, and password, so that my information is accurate and up-to-date.
3. As an existing customer, I want to add new customer data, such as the names and email addresses of my colleagues, so that I can compile an email list.
4. As an existing customer, I want to manage my email preferences, such as opting out of certain types of emails, so that I only receive emails that are relevant to me.
5. As an admin user, I want to view all user accounts and customer data, so that I can manage the web application and ensure that everything is working properly.
6. As an admin user, I want to create a new user account and assign the appropriate roles and permissions, so that I can give other users access to the web application.
7. As an admin user, I want to export an email list as a file attachment and send it to designated email recipients, so that the recipients can validate the email addresses and add them to their own email lists.
8. As an email recipient, I want to validate the email addresses in the list provided by Nadyne against my own list of valid email addresses, so that I can ensure that the emails I send are received by the intended recipients. (Validation done externally)
9. As a guest user, I want to browse customer data and learn more about the services offered by Nadyne, so that I can decide whether to sign up for an account.

### User task and goals

1. Customer:

* Goal: To register their details for a course of interest.
* Tasks: Enter their personal information, select the course they are interested in, and submit their registration.

1. Admin:

* Goal: To manage customer registrations and export customer data.
* Tasks: View customer registrations, validate customer data against a list, export customer data, and manage course information.

1. Manager:

* Goal: To view and analyze customer data.
* Tasks: View customer data in the dashboard, generate reports, and analyze customer data to make informed decisions.

1. IT Support:

* Goal: To manage the technical aspects of the application and ensure its proper functioning.
* Tasks: Monitor application performance, troubleshoot technical issues, and ensure that the application meets security and data protection standards.

## Database schema

The following database schema that will be used for the Nadyne web application:

## List of tables

1. Customers: stores the data for each customer, such as first name, last name, email address, phone number, country, city, date of birth, job type, and start date.
2. Validation\_list: stores the data for each validation list entry, such as first name, last name, email address, and phone number.
3. Courses: stores the data for each course, such as course name, course type, and course duration.
4. Course\_schedule - stores the data for each course schedule, such as start date, end date, and location.
5. Orders: stores the data for each order, such as order date, order amount, and payment status.

### Database structure and table columns

Table name: customers

| **Column name** | **Data type** | **Constraints** |
| --- | --- | --- |
| id | integer | primary key, autoincrement |
| first\_name | varchar(50) | not null |
| other\_names | varchar(100) | not null |
| email | varchar(100) | not null, unique |
| phone\_number | varchar(20) | not null |
| country | varchar(50) | not null |
| city | varchar(50) | not null |
| dob | date | not null |
| job\_type | varchar(50) | not null |
| course | varchar(100) | not null |
| start\_date | date | not null |

This table would store all the information related to customer registrations, and each record would be identified by a unique ID. The ID would be generated automatically using the autoincrement function. The email column would be set as unique to prevent multiple registrations with the same email address. The other columns would have non-null constraints to ensure that no data is missing in the database.

Table name: validation\_list

| **Column name** | **Data type** | **Constraints** |
| --- | --- | --- |
| id | integer | primary key, autoincrement |
| email | varchar(100) | not null, unique |

This table would store all the email addresses that have been validated, and each record would be identified by a unique ID. The ID would be generated automatically using the autoincrement function. The email column would be set as unique to ensure that there are no duplicate email addresses in the table. The email column would also have a non-null constraint to ensure that no data is missing in the database.

Table name: courses

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| course\_id | int | Unique identifier for the course |
| course\_name | varchar | Name of the course |
| course\_type | varchar | Type of the course |
| course\_duration | int | Duration of the course in weeks |
| course\_cost | int | Cost of the course |

This table is used to store information about courses offered by the company. It includes columns for the course name, description, duration, and cost. This table is used to provide information to customers about available courses.

Table name: course\_schedule

| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** |
| --- | --- | --- | --- | --- | --- |
| id | int(11) | NO | PRI | NULL | auto\_increment |
| Course\_name | varchar(255) | NO |  | NULL |  |
| Start\_date | date | NO |  | NULL |  |
| End\_date | date | NO |  | NULL |  |
| instructor | varchar(255) | NO |  | NULL |  |
| location | varchar(255) | NO |  | NULL |  |
| Total\_capacity | int(11) | NO |  | NULL |  |
| Available\_seats | int(11) | NO |  | NULL |  |
| Created\_at | timestamp | YES |  | NULL |  |
| Updated\_at | timestamp | YES |  | NULL |  |

This table will store information about the scheduled courses, including the course ID, start date, end date, and capacity. The id column serves as the primary key for the table, while the course\_id column is a foreign key referencing the id column in the courses table. This allows for a one-to-many relationship between courses and their scheduled instances.

Table name**:** orders

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| Order\_id | int | Unique identifier for the order |
| Customer\_id | int | Foreign key to link with the customers table |
| Course\_id | int | Foreign key to link with the courses table |
| Schedule\_id | int | Foreign key to link with the course\_schedule table |
| Order\_date | date | Date the order was made |
| Order\_amount | Decimal (10,2) | Amount paid for the order |
| Payment\_status | varchar | Payment status of the order |

The orders table stores information about customer orders for courses. It includes a unique id column for each order, a customer\_id column that references the customer who placed the order in the customers table, a course\_id column that references the course that was ordered in the courses table, an order\_date column to track the date when the order was placed, an order\_amount column to store the total cost of the order, and a payment\_status column to track the status of payment for the order.

### Table relationships

* One customer can make many orders (one-to-many relationship)
* One order is associated with one course (one-to-one relationship)
* One course can have many course schedules (one-to-many relationship)
* One course schedule can have many orders (one-to-many relationship)

## Development Environment Setup:

1. Install Python: Go to the Python website and download the latest version of Python for your operating system. Install Python on your computer.
2. Install VS Code: Go to the VS Code website and download the latest version of VS Code for your operating system. Install VS Code on your computer.
3. Install the AWS Command Line Interface (CLI): Go to the AWS CLI website and follow the instructions to install the AWS CLI on your computer.
4. Install the AWS Elastic Beanstalk CLI: Open a command prompt or terminal window and type the following command: **pip install awsebcli**
5. Create a new Flask application: In VS Code, create a new folder for your Flask application. Open a terminal window in VS Code and navigate to the new folder. Type the following command to create a new Flask application: **pip install flask**
6. Install any necessary dependencies: Depending on your specific project needs, you may need to install additional Python packages or libraries. Use **pip** to install any necessary dependencies.
7. Configure AWS Elastic Beanstalk: Use the AWS Elastic Beanstalk CLI to configure your application and environment settings. Follow the AWS documentation to set up a new Elastic Beanstalk environment.
8. Deploy your application: Once your environment is set up, use the AWS Elastic Beanstalk CLI to deploy your Flask application to the cloud.

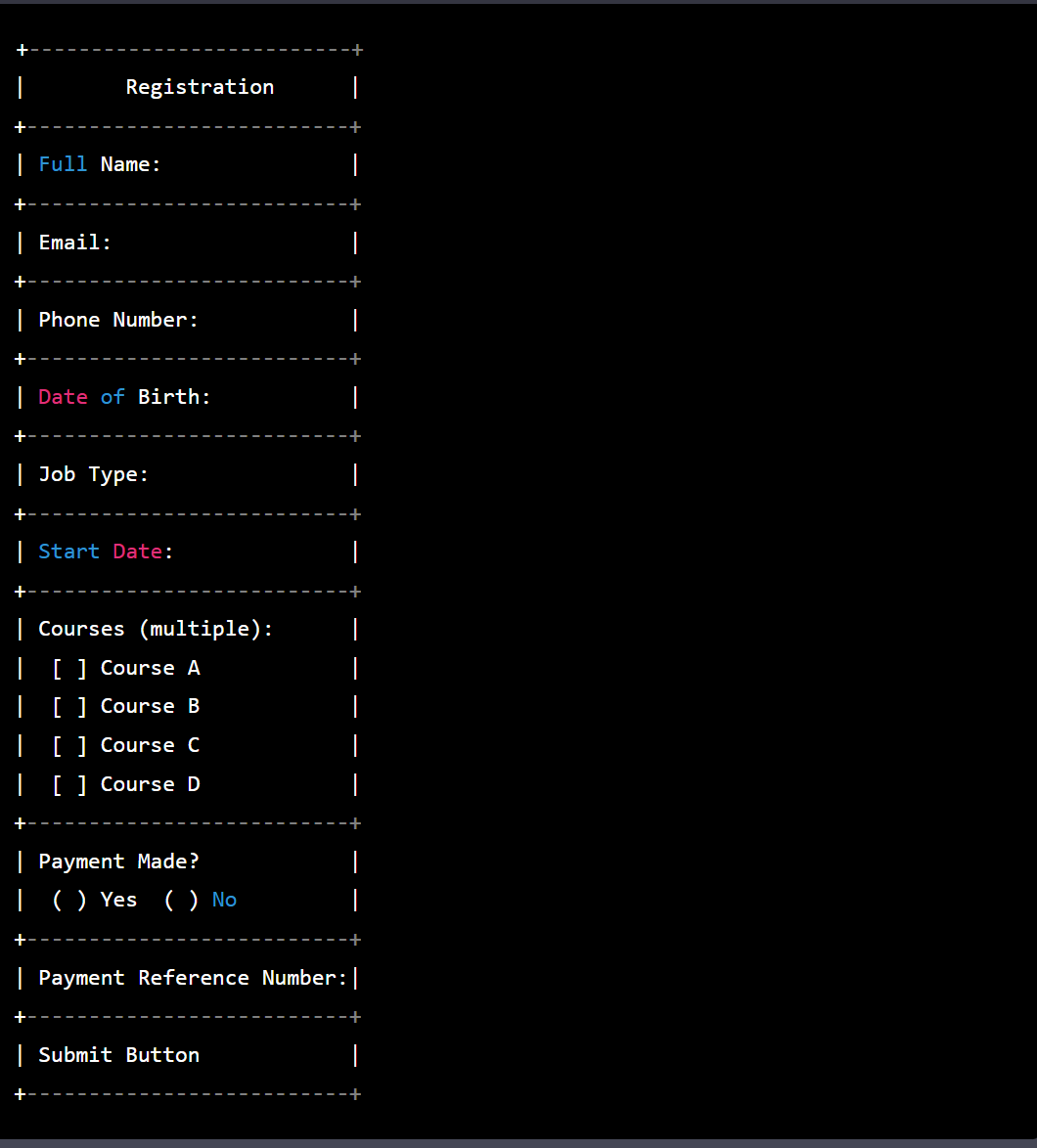
## Dependencies and libraries

* Flask: Flask is a popular Python web framework that provides a simple and easy-to-use interface for building web applications. It will be required to build the web application for Nadyne.
* Flask-MySQL: This is a MySQL database driver for Flask that provides support for connecting to a MySQL database from Flask. It is required to connect to and interact with the MySQL database that will store Nadyne's data.
* Flask-WTF: Flask-WTF is a Flask extension that provides integration with the WTForms library, which simplifies the process of building and validating web forms. It will be required for building and processing forms in the web application.
* Flask-Login: This is a Flask extension that provides user authentication functionality for Flask-based applications. It will be required to manage user authentication in Nadyne.
* Flask-Migrate: This is a Flask extension that provides database migration support for Flask applications using SQLAlchemy. It will be required to manage database migrations for Nadyne.
* SQLAlchemy: SQLAlchemy is a Python SQL toolkit and Object-Relational Mapping (ORM) library that provides a set of high-level API for connecting to and interacting with relational databases. It will be required to manage the database in Nadyne.
* PyMySQL: PyMySQL is a pure-Python MySQL client library that provides a low-level interface for connecting to and interacting with a MySQL database from Python. It will be required to connect to and interact with the MySQL database that will store Nadyne's data.
* WTForms: WTForms is a Python library for building web forms. It provides a set of high-level API for defining and validating web forms. It will be required to build and validate forms in the web application.

## Frontend development.

* HTML
* CSS
* Bootstrap
* JavaScript
* Python
* Flask-WTF (for form handling in Flask)

## Wireframes



### Upon submit click, here are the UX outcomes:

1. Success message: A success message could be displayed to the user indicating that their form submission was successful. The message could also include a summary of the user's input for their reference.
2. Error message: If the form submission fails for any reason, an error message should be displayed to the user. The message should be informative and provide guidance to the user on what went wrong and how to fix it.
3. Clear form: Once the form has been submitted successfully, the form fields should be cleared so that the user can easily input new data for another submission.
4. Confirmation email: An automated email could be sent to the user confirming that their form submission was received successfully. The email could include a summary of the user's input as well as any next steps they need to take.
5. Redirect: The user could be redirected to a confirmation page that summarizes their input and thanks them for their submission. The page could also include any next steps or important information related to their submission.
6. Add an "Other" option in the dropdown menu with a text field to enter the name of the course. When the user selects "Other", the text field will become active and they can enter the name of the course they want.
7. To hide the payment made button for users who select "Other", add some JavaScript to the frontend. When the user selects "Other", the JavaScript will hide the payment made button. When the user selects any other course from the dropdown menu, the JavaScript will show the payment made button.
8. When the user clicks submit and payment is not made, display a message asking the user to make payment before submitting. If payment is made, display a message asking the user to enter the payment reference number.
9. Once the user submits the form successfully, display a success message to let the user know that their information has been submitted. The form fields are cleared to prevent double sending.

## Backend development.

### Define the required endpoints

/api/register - User registration endpoint

/api/login - User login endpoint

/api/logout - User logout endpoint

/api/user - Get user details endpoint

/api/user/<user\_id> - Get or update a specific user's details

/api/courses - Get all courses available

/api/courses/<course\_id> - Get or update a specific course's details

/api/applications - Get all job applications

/api/applications/<application\_id> - Get or update a specific job application's details

/api/payments - Get all payments made

/api/payments/<payment\_id> - Get or update a specific payment's details

### Authentication

Flask-Login will be used

### Data Models

1. User: This model would store information about the users of the application, such as their name, email address, and password.
2. Course: This model would store information about the courses offered by the institution, such as the course name, description, and fee.
3. Enrolment: This model would store information about the enrolment of students in the different courses, such as the student's ID, the course ID, the start date, and the payment status.
4. Payment: This model would store information about the payments made by the students, such as the payment reference number, the amount paid, and the date of payment.
5. Email: This model would store information about the emails generated by the application, such as the recipient's email address, the subject, and the body of the email.

These models would need to be defined in the backend using SQLAlchemy, which would allow the application to interact with the database in a more intuitive and object-oriented way.

### Implementing MySQL database

1. Normalize your data: Normalization is the process of organizing data in a database so that it is consistent and can be easily queried. It involves breaking down large tables into smaller ones and ensuring that each table has a unique primary key.
2. Use indexes: Indexes can significantly improve query performance by allowing the database to quickly find the data you are looking for. Be sure to create indexes on columns that are frequently used in WHERE clauses or JOIN statements.
3. Use transactions: Transactions allow you to group a set of SQL statements into a single unit of work. They ensure that either all of the statements are executed or none of them are, helping to maintain data consistency.
4. Implement proper security: Make sure to set strong passwords for your users and limit access to the database to only those who need it. Use SSL encryption to protect data in transit.
5. Back up your data regularly: Backing up your database is essential in case of data loss or corruption. It's best to have a regular backup schedule and to store backups in a secure location.
6. Optimize your queries: Use EXPLAIN to analyze your queries and identify areas for optimization. Consider using tools like MySQL Workbench to help you optimize your database schema and queries.
7. Monitor your database: Keep an eye on your database's performance and health by monitoring system metrics like CPU usage, memory usage, and disk I/O. Use tools like MySQL Enterprise Monitor to help you proactively identify and resolve issues.

## Email API integration

Sendgrid: for sending transaction emails.

Mailchimp: for sending marketing emails.

## Testing

1. Unit Testing:

User authentication and authorization

Database integration and CRUD operations

API integration and communication

Input validation and error handling in both frontend and backend

User interface components, such as forms and buttons

Business logic, such as calculations and algorithms

Email sending and receipt confirmation

Payment processing and confirmation

Session management and security features

1. Integration Testing:

After the individual components have been tested, integrate them to ensure that they work together as expected.

1. End-to-End Testing:

Test the entire application from start to finish. It involves simulating user interactions with the application to ensure that everything works as expected.

1. Load Testing:

Choice made between Apache JMeter or Gatling to create scenarios that simulate user behavior such as logging in, filling out the form, and submitting it

1. Security Testing:

OWASP ZAP will be for security testing vvulnerabilities such as SQL injection attacks, cross-site scripting, and other common security issues.

1. Usability Testing:

Feedback from users will taking to improve the application's user interface and user experience.

1. Acceptance Testing: This involves getting feedback from the client or stakeholders to ensure that the application meets their requirements and is ready for deployment.

## Deployment

### Hosting

AWS Elastic Beanstalk will be used for deploying Python Flask applications on AWS with a MySQL database to simplify the process of deploying and managing applications by handling the deployment details such as capacity provisioning, load balancing, and automatic scaling.

## Monitor and maintenance.

### Metrices to track

1. User engagement: Measures how frequently users are interacting with the platform. Metrics such as daily or monthly active users, session duration, and pageviews per session can provide insight into user engagement.
2. User retention: Measures how well the platform is retaining users over time. Metrics such as retention rate, churn rate, and customer lifetime value can provide insight into user retention.
3. Conversion rate: Measures the percentage of users who complete a desired action on the platform, such as filling out a form or making a purchase.
4. User satisfaction: Measures how satisfied users are with the platform. Metrics such as Net Promoter Score (NPS), customer satisfaction (CSAT) score, and user feedback can provide insight into user satisfaction.
5. Revenue: Measures the revenue generated by the platform. Metrics such as average revenue per user (ARPU), conversion rate, and customer lifetime value can provide insight into revenue.

Expertise

Job Summary: We are seeking a highly skilled full stack developer to design and build our new web application, You will be responsible for the development of the front-end and back-end of the application, integration with third-party APIs, database integration, testing and deployment. The ideal candidate will have experience with Python, Flask, MySQL, AWS, and various front-end technologies such as Bootstrap, JavaScript, and React. You should also have experience with user authentication and authorization, unit testing, and performance testing.

Responsibilities:

* Collaborate with the product management team to understand the requirements for the application and translate them into technical specifications.
* Design, develop and deploy the front-end and back-end of the application using Python, Flask, MySQL, AWS, and various front-end technologies such as Bootstrap, JavaScript, and React.
* Ensure the application is secure, scalable, and highly available.
* Integrate the application with third-party APIs.
* Implement user authentication and authorization.
* Write unit tests and performance tests to ensure the application is working properly.
* Monitor and optimize the application's performance.
* Work with the DevOps team to deploy the application to production.

Requirements:

* Proven experience as a full stack developer, with at least 3 years of experience.
* Strong proficiency in Python, Flask, MySQL, AWS, Bootstrap, JavaScript, and React.
* Experience with user authentication and authorization.
* Strong experience in unit testing and performance testing.
* Ability to work independently with little supervision.
* Excellent communication skills and the ability to work collaboratively with a team.