**Practical Exposure to Automation Testing, Backend Validation, and Scalable Data Workflows**

*An Internship Report*

Submitted in Partial fulfilment of the Requirements for the Award of Degree

**Bachelor of Technology**

**In**

**Computer Science Engineering**

*Submitted By*

**Pratham Agrawal**

**Scholar ID: 211112446**

*Under the guidance of*

**Dr. Namita Tiwari**

(Internal Supervisor)

A logo of a company

AI-generated content may be incorrect.

**APRIL-MAY 2025**

**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING**

**MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY**

**BHOPAL (M.P.)- 462003**

**Table of Contents**

[DECLARATION 3](#_Toc195552503)

[CERTIFICATE 4](#_Toc195552504)

[ACKNOWLEDGEMENT 5](#_Toc195552505)

[INTRODUCTION 6](#_Toc195552506)

[TOOLS REQUIRED IN THE PROJECT 8](#_Toc195552507)

[LIFE CYCLE REFLECTED IN MY PROJECT 9](#_Toc195552508)

[UNDERSTANDING THE TECH STACK 11](#_Toc195552509)

[Selenium & Behave Framework 11](#_Toc195552510)

[Data & Big Data Tools. 15](#_Toc195552511)

[Data Processing (using Pandas & PySpark) 17](#_Toc195552512)

[Project Contributions and Implementation Experience 20](#_Toc195552513)

[Task 1: Fixed Bugs for UI Testing and Automation 20](#_Toc195552514)

[Task 2: Backend API Automation 21](#_Toc195552515)

[Task 3: API Integration Testing (Java) 22](#_Toc195552516)

[Task 4: UI Sanity Pipeline with YAML and Azure DevOps 23](#_Toc195552517)

[Task 5: Data Validation and Processing 24](#_Toc195552518)

[FUTURE SCOPE 25](#_Toc195552519)

[CONCLUSION 27](#_Toc195552520)

# DECLARATION

We hereby declare that the work which is being presented in this internship report entitled “**Practical Exposure to Automation Testing, Backend Validation and Scalable Data Workflow**”, submitted in partial fulfilment of the requirement for the award of the degree of **Bachelor of Technology in Computer Science Engineering**, has been carried out at Maulana Azad National Institute of Technology, Bhopal and is an authentic record of our own work carried out under the esteemed guidance of **Dr. Namita Tiwari**. The matter embodied in this dissertation, in part or whole has not been presented or submitted by us for any purpose in any other institute or organization for the award of any other degree.

We further declare that the facts mentioned above are true to the best of my knowledge. In case of any unlikely discrepancy that may possibly occur, we will be the one to take responsibility.

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No.** | **Name of the student** | **Scholar Number** | **Signature** |
|  | Pratham Agrawal | 211112446 |  |

Signature

Dr. Namita Tiwari

# CERTIFICATE

This is to certify that the dissertation work entitled “**Practical Exposure to Automation Testing, Backend Validation and Scalable Data Workflow**” is a Bonafide record of the work done by the following students and submitted in partial fulfilment of the requirements for the award of degree of **Bachelor of Technology in Computer Science Engineering.** To the best of my knowledge and belief, the dissertation embodies the work of the following candidates. They have duly completed the work and fulfil the requirement of the ordinance relating to the **Bachelor of Technology** degree from **Department of Computer Science Engineering, Maulana Azad National Institute of Technology, Bhopal, India.**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No.** | **Name of the student** | **Scholar Number** | **Signature** |
|  | Pratham Agrawal | 211112446 |  |

# ACKNOWLEDGEMENT

*"Acknowledgment is the first step to gratitude, and gratitude is the foundation of a fulfilling life."*

I wish to express my profound gratitude to all those who have contributed to the successful completion of this project.

Firstly, I extend my sincere appreciation to my project supervisors, **Dr. Namita Tiwari, Dr. Sanyam Shukla and Dr. Deepak Tomar** for their invaluable guidance, insightful feedback, and unwavering support. Their expertise and patience have been instrumental in shaping the direction and outcome of this project.

I am also deeply indebted to the faculty members of the **Department of Computer Science Engineering** at **Maulana Azad National Institute of Technology** for their support and for providing the necessary resources and facilities.

I would like to acknowledge the contributions of my fellow classmates, whose collaboration and camaraderie have greatly enriched this journey. Their constructive discussions and suggestions have significantly enhanced the quality of this work.

Thank you all for your support and confidence in my abilities.

Pratham Agrawal

Scholar ID: 211112446

# INTRODUCTION

Over the course of my Internship, I had the opportunity to make hands-on contributions across multiple domains, including **automated testing**, **API validation**, **backend data verification**, and **performance optimization**. This experience was not just about exposure to tools and frameworks—it involved actively solving real problems, fixing bugs, building and enhancing test frameworks, and optimizing workflows to improve reliability and efficiency. Through this journey, I developed strong technical proficiency in **Python, Java, SQL, Selenium, Behave, Pytest, YAML pipelines**, and various supporting tools like **Postman** and **Azure DevOps**.

One of my initial focus areas was in **UI automation testing**, where I worked with the **Selenium** and **Behave** frameworks. After gaining a clear understanding of the codebase and the structure of the test suite, I contributed by resolving a series of UI test failures. These were not trivial cosmetic issues—some bugs required rewriting XPATHs to reflect UI changes, modifying logic to correctly extract values from dynamic UI components, and removing deprecated functionalities that were affecting test accuracy.

I also implemented new exception handling strategies to deal with edge cases such as missing issuer names or fullscreen toggle errors. A common issue I tackled was the inconsistency in source filter values due to missing underscores. I resolved this by normalizing values and adding fallback logic, which enhanced the robustness and flexibility of the test automation system. Additionally, I implemented new features in the automation suite like **date functionality** and **page validation**, ensuring that the test cases kept up with evolving application features.

Transitioning from frontend testing, I made significant contributions to the **end-to-end testing of backend logic**, particularly in data validation scenarios. This work required me to dive into complex SQL queries involving **Common Table Expressions (CTEs)** and understand how different validation rules were applied across interrelated database tables. Using **Pytest**, I developed Python functions to compare API responses against database records with high accuracy.

Instead of relying on repeated calls to the database, I optimized the process by introducing **API response caching** and restructuring queries to return more usable, aggregated formats. These improvements not only reduced the number of API and DB calls but also enhanced test performance and reliability. A major breakthrough came when I identified a logical flaw in one of the validation rules (related to a missing condition), which had previously gone undetected. This not only improved test coverage but also led to a fix in the actual rule implementation.

In the area of **API integration testing using Java**, I worked on designing and implementing dynamic and reusable test logic. After understanding the structure and patterns in the existing codebase, I wrote well-structured test functions that used clean naming conventions, minimal nesting, and high reusability. I dynamically generated SQL queries, cast values appropriately for comparison, and wrote **over 70 test cases across 12 APIs**, ensuring comprehensive functional coverage.

Additionally, I contributed to the setup of a **sanity testing pipeline** using **YAML and Azure DevOps**. This pipeline was designed to automatically run lightweight tests after each code change or bug fix. I became familiar with YAML syntax, AzureKeyVaults for secret management, task definitions, artifacts, and telemetry publishing. This experience gave me insight into how **CI/CD pipelines** are structured and how they help maintain stability and quick feedback in a production workflow.

I also participated in project, where I executed and validated data for multiple external vendors. Beyond running existing checks, I enhanced the framework by adding remaining column validation and began optimizing runtime performance. To tackle performance challenges, I explored and implemented **ThreadPool**, **ProcessPool**, and **PySpark-based** parallel processing. With PySpark, I was able to reduce data validation time by **approximately 50%**, demonstrating the power of distributed computing in handling large datasets efficiently.

This experience has been both enriching and transformative, allowing me to move beyond theoretical learning into active, hands-on contribution. From automating UI tests and validating complex business rules to optimizing data workflows and implementing CI/CD pipelines, I had the opportunity to apply my technical skills in real-world scenarios. Each task not only deepened my understanding of modern development practices but also enabled me to deliver tangible improvements across systems. This journey has significantly strengthened my confidence in working with large-scale applications, debugging complex logic, and building reliable, maintainable test frameworks—skills that I will carry forward into my future endeavors

## TOOLS REQUIRED IN THE PROJECT

* **DataBricks:** Databricks is a cloud-based platform designed for big data analytics and machine learning. I used Databricks primarily for running large-scale data validation and transformation tasks. Its notebook interface allowed me to write and execute code efficiently, collaborate with others, and visualize outputs.
* **PySpark:** PySpark is the Python API for Apache Spark, used for fast, parallel data processing. I applied it to validate and transform large datasets, reducing execution time by up to 50% compared to traditional methods.
* **Postman:** Postman is an API testing tool I used to send requests, check responses, and automate test cases. It made it easy to validate API outputs and simulate various user scenarios during backend testing.
* **Selenium & Behave:** Selenium automated the browser interactions, while Behave helped me write BDD-style UI tests. I implemented dynamic selectors and exception handling to make the tests robust and readable.
* **PyTest:** Pytest is a popular Python testing framework known for its simplicity and powerful features. I used it to write test functions that validated API responses against database values. It supported fixtures, assertions, and parameterized tests, making my test code reusable and clean. Pytest’s rich reporting and plugin ecosystem also helped in integrating tests into pipelines and identifying issues quickly.
* **IDE:** Used **IntelliJ IDEA** for Java-based development, especially while working on API integration testing. Its powerful features like intelligent code suggestions, debugging tools, and smooth navigation made writing and maintaining code much easier. For Python-related tasks, I used **PyCharm**, which provided excellent support for test frameworks like Pytest and Behave. Both IDEs significantly boosted my productivity through their clean interfaces, real-time feedback, and efficient version control integration.
* **Git:** Git is a distributed version control system used for tracking changes in source code during software development. It allows multiple developers to collaborate on a project, manage code versions, and maintain a history of changes. Git's branching and merging capabilities make it a powerful tool for managing development workflows and ensuring code integrity.

## LIFE CYCLE REFLECTED IN MY PROJECT

The lifecycle of a website development project typically involves several stages, each crucial for ensuring the final product meets the client's needs and expectations. Let's explore these stages in detail:

* **Planning and Requirement Analysis**

My journey started with thoroughly understanding the existing codebase and the overall system architecture. I took the time to explore the functionality of the UI, Rating Operations, and the associated backend processes. I studied how different data sources interacted and how business rules impacted the system. I also learned to work with complex SQL queries, leveraging Common Table Expressions (CTEs) to extract and manipulate data. This initial phase was critical, as it laid the groundwork for all my subsequent work, ensuring that I could contribute effectively and with a solid grasp of the project's requirements.

* **Planning & Tool Setup**

Once I had a clear understanding of the system, I set up the necessary tools and frameworks to carry out testing and development. I worked with **Selenium** for automating UI tests and **Behave** for behavior-driven development (BDD) testing. In addition, I used **Pytest** for validating APIs and ensuring data consistency between backend systems. I also explored **Azure DevOps** to automate the build and deployment pipeline using YAML, which involved learning how to work with tasks, artifacts, and secrets management through **Azure Key Vault**. This phase was about preparing the testing environment, ensuring I had the tools to build and execute automated tests effectively.

* **Development of Tests / Enhancements**

With the setup in place, I started contributing to the project by focusing on test development and enhancing the system’s functionality. For the DAP UI, I fixed several bugs, including updating XPATHs and functions that extracted values from the UI. I also removed irrelevant functionality and added exception handling to ensure the robustness of the UI tests. In addition, I developed new features, such as a date selection feature, and enhanced the event page's logic to meet specific test requirements. On the backend, I wrote complex SQL queries to validate the data, created reusable test functions, and integrated those tests with automated workflows. This phase was about directly contributing to improving both the system and the quality of its testing.

* **Execution & Testing**

The next step involved executing the tests and ensuring the system was working as expected. I performed **end-to-end testing** of the Rating Operation rules, focusing on API validation and rule validation. I used **Postman** to test API endpoints, validating the responses against the expected results. On the UI side, I ran sanity checks through Azure DevOps pipelines, ensuring that everything was functioning properly after code changes. I also identified and reported issues, like the mismatches in Rule-8, where the condition was not applied correctly. This stage was crucial for verifying that the system worked as intended and that all defects were identified early in the process.

* **Optimization & Automation**

In order to enhance the system’s performance, I implemented several optimizations. For data validation tasks, I used **PySpark** to process large datasets more efficiently, reducing the processing time by 50%. I also implemented caching to minimize the number of API calls, and refactored SQL queries to retrieve data in a more usable format, optimizing database calls. On the testing side, I streamlined the test scripts by removing redundant logic and handling edge cases more effectively. This phase focused on improving the system’s efficiency and reducing resource consumption, making both the testing and production environments more stable and scalable.

* **Reporting & Learning**

Throughout the project, I maintained a continuous cycle of learning and improvement. I regularly documented issues, solutions, and optimizations, contributing to a shared knowledge base within the team. I also engaged in debugging test failures, analyzing root causes, and learning from each challenge. By working with tools like **Git**, I tracked my changes and collaborated with the team, contributing to better code quality and version management.

# UNDERSTANDING THE TECH STACK

## Selenium & Behave Framework

**Selenium** is a powerful open-source framework used for automating web browser interactions. It allows developers and testers to write scripts in multiple programming languages (like Python, Java, C#, etc.) to simulate user actions such as clicking buttons, entering text, navigating between pages, and validating content. Selenium is widely used for functional, regression, and end-to-end UI testing across various browsers like Chrome, Firefox, Safari, and Edge. Its flexibility and compatibility with various tools and frameworks make it a key component in modern automated testing environments.

**Installing Selenium**

Selenium can be easily installed using Python’s package manager pip. Simply run:

A black and white image

AI-generated content may be incorrect.

**Updating Selenium**

To upgrade Selenium to the latest version and access the newest features and compatibility fixes, use:

**A black and white background with white text

AI-generated content may be incorrect.**

**Use**

Selenium allows users to launch a browser, navigate to URLs, and perform actions like clicking buttons, entering text, submitting forms, or validating web content. A test usually starts by importing the webdriver module, launching the browser, and then interacting with elements using different locating strategies like ID, XPath, or CSS Selector. Selenium supports both direct execution and integration into test frameworks like Behave or PyTest.

A computer screen shot of a computer code

AI-generated content may be incorrect.

Selenium is widely used for automating regression testing of web applications. It’s useful in scenarios like form submission validation, login functionality, or checking UI consistency across browsers. It’s also used for web scraping, performance testing (in combination with other tools), and automating repetitive web tasks.

**Key Features**

Support for all modern browsers and platforms

Works with Python, Java, C#, Ruby, and JavaScript

Supports complex user interaction simulations

Integrates with test frameworks and CI/CD pipelines

**Behave Framework: Behavior-Driven Testing with Python**

Behave is a Python-based Behavior Driven Development (BDD) testing framework that allows testers and developers to write test scenarios in natural language. It uses Gherkin syntax to define features and scenarios, providing a clear and structured way to describe application behavior.

Gherkin syntax promotes human-readable test cases by using the Feature, Scenario, Given, When, and Then keywords. This encourages collaboration between developers, testers, and stakeholders by making tests easy to understand and validate.

A screenshot of a computer

AI-generated content may be incorrect.

**Integrating Selenium with Behave**

Behave and Selenium can be seamlessly integrated to automate browser-based functional testing. The Gherkin scenarios serve as test documentation while Selenium performs the actual interactions. This combination provides a powerful structure for end-to-end testing.

Behave stores the browser instance (context.driver) in the context object to allow sharing between steps. Hooks such as before\_scenario and after\_scenario can be used to initialize or close the browser, capture screenshots on failure, or reset state.

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

**Project Structure**

**A screenshot of a computer

AI-generated content may be incorrect.**

This structure separates Gherkin files, step definitions, hooks, and outputs, making the project easy to maintain and scale.

and features that will shape the future of web development.

## Data & Big Data Tools.

**Databricks** is a powerful data analytics platform built on Apache Spark that allows for large-scale data processing, analytics, and machine learning in the cloud. It provides a collaborative environment for data engineers, scientists, and analysts through interactive notebooks that support Python, SQL, Scala, and R. Databricks excels at handling **big data workloads**, enabling distributed computing and real-time streaming analytics.

**Key highlights:**

* Unified platform for data engineering, machine learning, and analytics
* Built-in support for **Delta Lake** for ACID-compliant data lakes
* Highly scalable with auto-cluster and parallel processing support
* Integrates easily with Azure, AWS, and Google Cloud
* Supports notebooks with version control, collaboration, and job scheduling

**Use Case Example:** Running PySpark jobs to process millions of records, clean data, and reduce execution time using parallel processing.

**BigQuery:**

Google BigQuery is a fully managed, serverless data warehouse that enables super-fast SQL queries on massive datasets. It is designed to handle **petabyte-scale** analytics using a distributed architecture that abstracts infrastructure management. BigQuery automatically scales based on data size and query complexity, making it a go-to tool for data analysts and BI professionals.

**Key highlights:**

* Supports ANSI SQL and machine learning with BigQuery ML
* Seamless integration with Google Cloud ecosystem and tools like Data Studio
* Real-time analytics with streaming data ingestion
* Pay-per-query pricing model – no need to manage servers or infrastructure
* High availability and built-in security

**Use Case Example:** Executing complex joins and aggregations across multiple tables using Common Table Expressions (CTEs) to validate business rules at scale.

A screenshot of a computer program

AI-generated content may be incorrect.

## Data Processing (using Pandas & PySpark)

**Pandas** is a powerful and easy-to-use open-source Python library for data manipulation and analysis. It provides fast, flexible data structures like DataFrame and Series that are ideal for handling structured (tabular) data. With Pandas, you can clean, transform, analyze, and visualize your data with just a few lines of code.

Pandas is highly efficient for in-memory processing and is commonly used in small to moderately large datasets in data analysis, machine learning, and reporting workflows.

**Key Features of Pandas:**

* Provides DataFrame and Series objects for working with rows and columns.
* Supports powerful groupby, merge, and pivot operations.
* Allows handling of missingdata, date**/**timedata, and categoricalvariables.
* Easy integration with other libraries like NumPy, Matplotlib, and Scikit-learn.
* Ideal for exploratory data analysis (EDA) and quick prototyping.

**Benefits of Pandas:**

* Easy to learn and write with simple, readable syntax.
* Extensive built-in methods for data cleaning and transformation.
* Fast in-memory processing for medium datasets.
* Huge community support and documentation.

**Use Cases of Pandas:**

* Data cleaning and preprocessing before modeling.
* Loading data from CSV, Excel, JSON, SQL, and more.
* Exploratory data analysis (EDA) using descriptive statistics.
* Data transformation and feature engineering for ML models.
* Generating summary reports or aggregations.

A screenshot of a computer program

AI-generated content may be incorrect.

**PySpark**

PySpark is the Python API for **Apache Spark**, an open-source, distributed computing system designed for big data processing. PySpark allows you to harness the power of Spark using familiar Python syntax. It is built for handling **large-scale data** that cannot fit into memory and enables processing across multiple nodes.

PySpark supports various data operations like SQL queries, streaming data, machine learning, and graph processing through Spark's modular architecture.

**Key Features of PySpark:**

* Distributed processing of huge datasets across clusters.
* High-performance in-memory computation.
* Lazy evaluation and DAG-based execution engine.
* Integrates with SparkSQL, MLlib, GraphX, and Streaming.
* Supports reading from HDFS, S3, Hive, Kafka, etc.

**Benefits of PySpark:**

* Can handle terabytes of data efficiently.
* Great performance boost using parallelism and cluster computing.
* Enables seamless scaling from a laptop to a cluster.
* Combines Python’s simplicity with Spark’s speed and scalability.
* Useful in production pipelines involving batch or streaming data.

**Use Cases of PySpark:**

* ETL (Extract, Transform, Load) operations on large datasets.
* Building scalable machine learning pipelines.
* Real-time stream processing from Kafka or other sources.
* Data transformation in distributed environments like Databricks.
* Joins and aggregations across massive tables.

A screen shot of a computer program

AI-generated content may be incorrect.

Both Pandas and PySpark are essential tools in the data engineer’s and analyst’s toolkit. While Pandas is excellent for working with local data, PySpark is built for high-scale processing and is widely used in platforms like **Databricks** for real-world data pipelines.

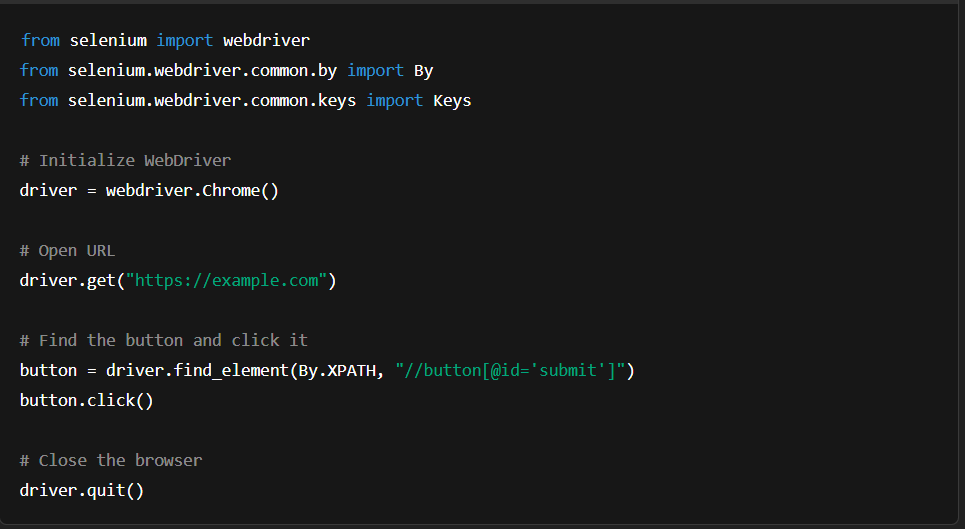
# Project Contributions and Implementation Experience

## Task 1: Fixed Bugs for UI Testing and Automation

In this project, my primary focus was on enhancing the **UI test automation** process using **Selenium** and **Behave**, two powerful tools for automating web applications and enabling behavior-driven development. While I did not write the initial test cases, I worked extensively on **fixing and improving existing test scripts** to ensure that the UI components functioned correctly. Key activities included:

* **Xpath Management**: Updated and fixed XPaths to correctly locate and interact with UI components that had changed over time.
* **Exception Handling**: Implemented exception handling for scenarios where expected elements were not found or where full-screen actions failed.
* **Improving Test Coverage**: Expanded the test coverage to include edge cases such as blank fields and missing required data.

Below is a sample code snippet that showcases a Selenium WebDriver script written for navigating to a page and clicking a button:



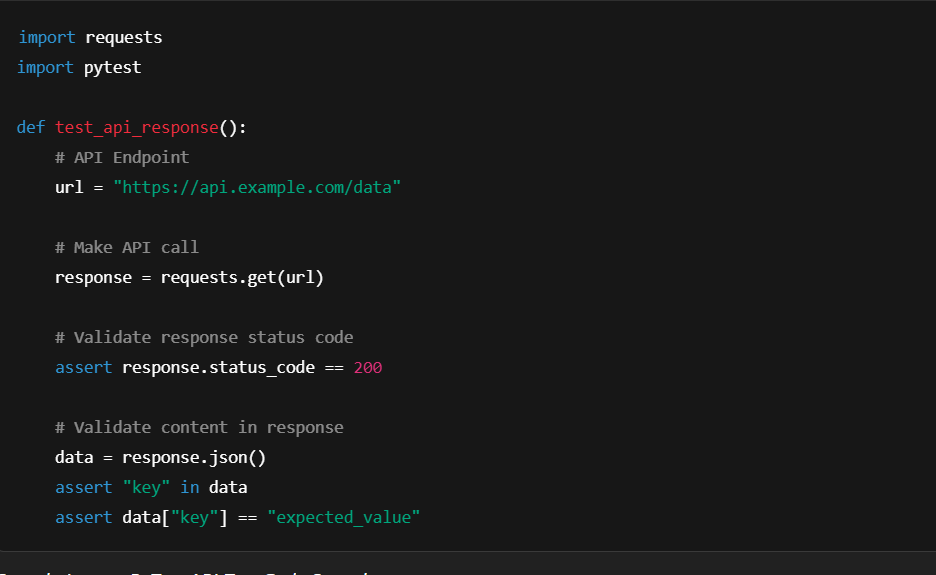
This project not only enhanced my understanding of UI automation but also gave me an opportunity to improve existing workflows. I was able to actively contribute towards making the UI testing framework more reliable, maintainable, and efficient, thereby ensuring better coverage and faster feedback for the development team.

## Task 2: Backend API Automation

In this project, I focused on **automating backend tests** and improving API validation processes. I worked extensively with **Python** and **PyTest** to write test cases that validate data fetched from the API against data from the database. Some of the main contributions included:

* **API Test Development**: I created multiple API tests to validate responses against expected results, checking the data integrity and correctness.
* **Optimizing SQL Queries**: I enhanced SQL queries to improve performance by reducing unnecessary database calls and using **Common Table Expressions (CTEs)** to simplify complex queries.
* **API Caching**: To minimize redundant API calls, I implemented a caching mechanism, which sped up testing by reusing previously fetched data.
* **Validation**: I worked on validation logic to compare API data with backend database values, identifying mismatches and ensuring data consistency.

Here's an example of an API test written using **PyTest** and **requests** library:



This experience strengthened my skills in API validation, SQL optimization, and rule-based logic testing.

## Task 3: API Integration Testing (Java)

In this Java-based project, I focused on writing and executing integration tests for various APIs involved in a backend system. My goal was to validate that each API performed its intended function and returned consistent and correct data. I spent time analyzing the existing Java codebase to understand how clean and maintainable code was structured, including practices like meaningful naming, minimal nesting, and modular functions. My contributions involved:

* **SQL Query Optimization**: I optimized SQL queries for better performance, ensuring they returned data in a more usable format.
* **Integration Tests**: Developed **integration tests** to verify data flow between the API and the database, ensuring data consistency throughout the system.
* **Dynamic SQL**: Created dynamic SQL queries based on test parameters, allowing for flexible and reusable queries.
* **Error Handling**: Implemented robust error handling to manage scenarios where expected data was missing or incomplete.

Here’s a simple example of SQL query written in Python for integration testing:



This project sharpened my Java programming skills and provided real-world exposure to integration testing in an enterprise environment. It also deepened my appreciation for writing maintainable code and understanding the complete lifecycle of an API from development to production validation

## Task 4: UI Sanity Pipeline with YAML and Azure DevOps

This project revolved around setting up a sanity pipeline to test small but critical parts of a UI application after every deployment. My primary task was to design and implement a pipeline using YAML scripting in Azure DevOps. This pipeline helped validate basic UI functionalities and provided quick feedback to developers before progressing to full regression testing. Key highlights included:

* **Pipeline Creation**: Designed a **CI pipeline** that triggered automated tests, reducing manual testing time after code updates.
* **Integration with Azure KeyVault**: Used **Azure KeyVaults** to securely manage secrets, API keys, and other credentials within the pipeline.
* **Sanity Testing**: Automated sanity tests to ensure basic functionality was intact after each change.
* **Telemetry and Artifact Publishing**: Incorporated telemetry to track test results and artifacts for further analysis.

Here’s a snippet of a simple **YAML** file used to automate testing in Azure DevOps:



Through this project, I gained practical experience in DevOps practices and how test automation can be integrated into deployment pipelines. It strengthened my ability to work with YAML, understand pipeline architecture, and contribute to faster, safer software releases.

## Task 5: Data Validation and Processing

In this data-intensive project, I worked with large volumes of data from multiple vendors. My responsibilities included executing the reporting workflows, validating the accuracy of the data generated, and ensuring that the reference points used in each report matched expectations. Key activities included:

* **Data Quality Checks**: Developed validation scripts to check data integrity, missing values, and anomalies in the dataset.
* **PySpark Optimization**: Leveraged **PySpark** to handle large datasets efficiently, optimizing data transformations to reduce computation time by over 50%.
* **Schema Validation**: Ensured that the data adhered to the expected schema by performing automated checks on column names, data types, and ranges.
* **Integration with BI Tools**: Integrated data validation with **BigQuery** and **Databricks**, enabling real-time reporting and analysis.

A sample code snippet for performing data validation with **PySpark**:



This project exposed me to real-world data engineering challenges and equipped me with skills in distributed data processing, validation, and optimization. It was a strong blend of technical depth and analytical reasoning, allowing me to apply best practices in both code efficiency and data quality assurance.

# FUTURE SCOPE

**In UI Testing Automation**

* Adoption of AI-based testing tools for auto-generating test cases and self-healing scripts when UI changes.
* Greater integration of cloud-based testing platforms for cross-browser and cross-device testing.
* Enhanced collaboration through Behavior-Driven Development (BDD) using tools like Behave, improving clarity and traceability.
* Shift toward low-code/no-code automation tools, reducing the need for deep coding expertise.
* Increased focus on parallel test execution and test scalability using containers and grid services.

**In Backend Automation**

* Growing use of microservices and API-first architecture will demand more dynamic and scalable test frameworks.
* Integration with CI/CD pipelines for continuous backend testing during every code push or deployment.
* Development of mocking/stubbing tools to simulate API dependencies and test isolated components.
* Enhanced support for asynchronous workflows and event-driven systems, making automation more realistic and reliable.
* Use of contract testing (e.g., Pact) to ensure compatibility between services during update

**In Data Validation**

* Increased use of distributed computing frameworks like PySpark for validating large-scale data efficiently.
* Implementation of real-time data validation for streaming pipelines alongside traditional batch validation.
* Use of ML algorithms to detect anomalies, missing values, or data drifting automatically.
* Integration with metadata and data catalog tools for schema validation, column tracking, and data lineage.
* Rise of data observability platforms to monitor, alert, and report on data quality issues proactively.

# CONCLUSION

This Internship served as a significant step in my practical learning journey, allowing me to contribute across multiple stages of software development and testing. I gained experience working on both the UI and backend sides of the system, with exposure to automation tools like **Selenium** and **Behave**, as well as testing frameworks like **PyTest** and integration testing using **Java**. I fixed various UI issues, enhanced test cases by adding exception handling, and implemented new functionality to ensure more reliable and maintainable automation scripts. My contributions helped improve the overall stability and reliability of the testing process.

On the backend, I focused on validating API responses by comparing them with database records using Python and SQL. I wrote dynamic and optimized queries using CTEs, reduced unnecessary API and database calls by implementing caching, and discovered important mismatches that led to the correction of business logic. Additionally, I worked on validation and optimized heavy data workflows using **PySpark** and multiprocessing, which significantly improved performance. Creating and running sanity pipelines using YAML and DevOps principles further gave me insight into how enterprise-level applications are continuously tested and deployed.

The tech stack I worked with included tools like **Postman**, **Git**, **BigQuery**, **Databricks**, **IntelliJ**, and **PyCharm**, each playing a role in enhancing productivity, collaboration, and performance. This experience not only strengthened my technical foundation but also improved my problem-solving, debugging, and optimization skills. Overall, I was able to apply what I learned in real scenarios, while also making valuable contributions to each module I worked on. This report is a reflection of both my growth and the impact I was able to make during the internship.