IMPRESS High Level Documentation

HOHENSTEIN

Bönnigheim, Germany

Illia Rohalskyi

August 2023

Contents

Chapter 1

Introduction

1.1 Purpose of the Document

The purpose of this document is to give a reader an overview of a project, it's structure and functionalities. It serves as a comprehensive yet concise document that could be read to understand the project scope, architecture and other essential details without delving into intricate technical details.

1.2 Scope

The scope of this document encompasses the entire "IMPRESS" project including its objectives, features, architecture, deployment strategy, and testing approach. The document does not delve into detailed technical implementation but provides an overview to help a reader understand the project's essence.

1.3 Definitions

· **IMPRESS:** The name of the project being documented.

· **System Architecture:** The high-level structure and organization of the project's components and modules.

· **Deployment Strategy:** The plan for deploying the project in various environments.

· **Testing Strategy:** The strategy for testing the project's functionalities and ensuring quality.

· **Use Cases:** Scenarios that describe how users interact with the project and achieve their goals.

· **Milestones:** Key points in the project timeline that mark significant achievements or progress.

· **References:** External resources and materials used as references during project planning and design.

· **Component:** A modular and self-contained unit of the system that has specific functionalities. Components can interact with each other to achieve higher-level features.

· **Continuous Integration (CI):** A software development practice that involves regularly integrating code changes into a shared repository. CI aims to detect and resolve integration issues early in the development cycle.

· **Continuous Deployment (CD):** A software development practice where changes to code are automatically built, tested, and deployed to production environments. CD aims to reduce manual intervention, minimize deployment delays, and ensure that new features and bug fixes are quickly delivered to end-users.

· **Sprint:** A timeboxed period (usually 1 to 4 weeks) in an Agile development cycle during which a team works on a set of planned tasks. Sprints are designed to deliver specific features or improvements.

· **Agile:** A flexible and iterative approach to software development that emphasizes collaboration, adaptability, and customer feedback. Agile methodologies prioritize delivering small increments of working software in short cycles.

· **Maintainability:** The measure of how easily a software system can be modified, updated, extended, or repaired over its lifecycle. A maintainable system is designed with clear and organized code, well-documented components, and modular architecture, making it more cost-effective and efficient to manage and evolve.

Chapter 2

General Description

2.1 Product Perspective

The project's primary focus is to develop a predictive model for surface tension. Predicting surface tension holds significant importance due to its essential role in the textile manufacturing industry. Surface tension affects various processes like dyeing, finishing, printing, and coating, which are integral to producing high-quality fabrics and materials. Accurate predictions can lead to improved product outcomes, reduced costs, and enhanced overall efficiency in the textile production process.

2.2 Tools Used

In our project, we rely on a variety of tools to effectively manage and enhance our work. These tools play a crucial role in enabling collaboration, version control, automation, and efficient project management.

- **Jira:** Jira is our project management tool that helps us track tasks, assign responsibilities, and maintain a clear overview of our project's progress. It ensures that everyone is on the same page and working towards our common goal.
- **MLFlow:** MLFlow assists us in managing our machine learning models. It enables us to track different model versions, monitor their performance, and store associated artifacts. This allows us to make informed decisions when choosing which model to deploy.
- **DagsHub:** DagsHub serves as our collaborative platform for machine learning projects. It aids in version control, data sharing, and team collaboration. DagsHub ensures that our project components are well-organized and accessible to all team members.

- **DVC (Data Version Control):** DVC helps us manage and track changes to our datasets. It ensures that our data remains consistent and well-documented, making it easier to maintain reliable and accurate machine learning models.
- **Git/Github:** Git and GitHub are essential for version control and code collaboration. Git tracks changes to our codebase, while GitHub provides a platform for storing, sharing, and collaborating on code repositories.
- **Prefect:** Prefect is our tool for workflow automation. It allows us to define, schedule, and monitor complex workflows. Prefect improves efficiency by automating repetitive tasks and ensuring orderly execution.
- **GitHub Actions:** GitHub Actions automates tasks and workflows within GitHub repositories. It helps us maintain code quality by automating testing and deployment processes, reducing errors and ensuring a consistent codebase.

2.3 General Constraints

2.3.1 General Constraints for MLOps Project

- Implement robust error handling and exception management to prevent application crashes. Provide meaningful error messages for troubleshooting.
- Continuously monitor the performance of deployed models using monitoring tools. Detect anomalies, concept drift, and degradation in model performance, triggering alerts.
- Maintain version control for trained models to track changes and revert to previous versions if necessary.
- Design your codebase to support deployment across different environments using configuration files.
- Implement data versioning to ensure consistency across different stages of the pipeline.

2.3.2 Object-Oriented Design Practices

- Utilize proper class and module organization to promote modularity and code reusability.

2.3.3 Continuous Integration and Deployment (CI/CD)

- Set up an automated CI/CD pipeline to streamline the process of testing, building, and deploying code and models.

- Use tools for automating testing and deployment tasks.
- Employ versioned Docker containers to ensure consistent environments across different stages of the pipeline.

2.3.4 Model Monitoring and Governance

- Monitor model performance metrics to detect deviations from expected behavior.
- Implement model explainability techniques to provide insights into model predictions and decisions.
- Maintain a model registry to keep track of deployed models, their versions, and associated metadata.
- Establish a clear process for model retraining and version updates based on changes in data distribution or business requirements.

2.4 Assumptions

- **Data Availability**: We assume that the required input data will be available in the expected format and structure. Any changes or variations in data schema may require adjustments in the data preprocessing and modeling stages.
- **Stable Environment**: We assume that the deployment environment, including infrastructure and dependencies, will remain stable during the project's deployment and operation. Any changes to the environment might impact the system's performance and behavior.
- **Representative Training Data**: We assume that the training data used for model development accurately represents real-world scenarios. Any biases or limitations in the training data might affect the model's generalization and predictions.
- **Stable Model Assumptions**: We assume that the underlying assumptions of the chosen machine learning models hold true within the context of our problem. Deviations from these assumptions might lead to suboptimal or unpredictable results.
- **Limited Scope of Features**: We assume that the features used for model training and prediction are relevant and encompass the most important aspects of the problem. Features not considered might impact the model's performance.
- **Monitoring and Maintenance**: We assume that regular monitoring and maintenance of the deployed models will be conducted to ensure their

continued effectiveness. Any neglect in monitoring might lead to model degradation.

- **User Interaction**: We assume that user interactions with the deployed system will follow expected patterns. Unexpected user behavior might necessitate adjustments in user interfaces or feedback mechanisms.

- **Resource Availability**: We assume that the required computational resources, such as memory, processing power, and storage, will be available for model training, deployment, and monitoring.

Chapter 3

System Architecture

3.1 Overall Architecture

The architecture of the system is organized to provide a clear structure for both development and deployment. It consists of several key components and directories that contribute to the project's functionality. In this documentation, we have omitted unnecessary files and folders to provide a concise and focused representation of the system architecture, allowing for a clearer understanding of the key components and their relationships. Now, let's provide an overview of the project structure with the assistance of a diagram:

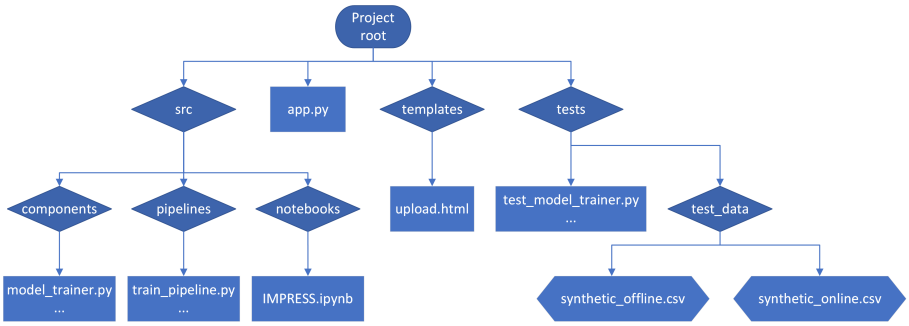

Figure 3.1: Project Structure Diagram

In the diagram above, you'll gain a visual understanding of how the project was structured. As we delve into each component in detail, this diagram will serve as a good reference point.

3.1.1 Source Components (`src/components`)

This directory houses modular components that serve specific functions. These components can either be used independently or combined to accomplish more

extensive tasks.

3.1.2 Pipelines (`src/pipelines`)

The `pipelines` directory contains modules that orchestrate the execution of various components to achieve specific tasks. These pipelines are designed to streamline complex workflows, such as data transformation and model prediction.

3.1.3 Web Application (`app.py`)

The project's main entry point is `app.py`, which is responsible for creating the web application. This application is designed for deployment and serves as the interface for users to interact with the system.

3.1.4 Templates (`templates`)

Within this directory, you'll find HTML templates that define the structure and layout of web pages displayed by the application. These templates are used by `app.py` to render user interfaces.

3.1.5 Testing (`tests`)

The `tests` directory is dedicated to testing the system's components and their integration. It includes unit tests for individual components as well as tests for the entire pipeline to ensure robust functionality.

3.1.6 Notebooks (`src/notebooks`)

This directory contains Jupyter notebooks used for data analysis and exploration. While not directly involved in the project's core functionality, these notebooks are valuable for gaining insights and conducting experiments.

The architecture is designed to promote modularity, maintainability, and scalability. Components and pipelines can be developed and tested independently, making it easier to extend and enhance the system's capabilities. Additionally, the clear separation of concerns between components and the web application simplifies the development and deployment process.

Chapter 4

Deployment

4.1 Deployment Diagram

4.2 Hardware and Software Requirements

Chapter 5

Testing Strategy

5.1 Test Plan

5.2 Testing Scenarios

Chapter 6

Use Cases

6.1 Use Case Diagram

6.2 Use Case Descriptions

Chapter 7

Project Timeline

7.1 Project Milestones

7.2 Timeline and Deliverables

Chapter 8

Retrospective

8.1 Project Achievements

8.2 Challenges Faced

8.3 Areas for Improvement

Chapter 9

References

9.1 External Resources

9.2 Documents and Materials Used