INTERNSHIP REPORT

Name: Sahil D Choudhary

Project: Air Aware – Smart Air Quality Prediction System
Team No. 2

Project Title: Air Aware – Smart Air Quality Prediction System

(Day 1)

1. Overview

Air Aware is a smart, AI-powered web-based system designed to monitor, analyse, and forecast air quality in real time. The platform integrates Machine Learning (ML), modern web technologies, APIs, and databases to deliver accurate Air Quality Index (AQI) predictions to users.

The system processes pollutant data such as **PM2.5, PM10, NO2, SO2, CO, and Ozone**, collected from datasets (Kaggle, Hugging Face, OpenAQ) and real-time sensors. After processing the input using ML models, Air Aware provides:

- Real-time AQI
- Future AQI forecast
- Weather conditions (Sunny, Rainy, Cloudy, etc.)
- Health implications (Safe / Unsafe to inhale)

The goal is to achieve 95–98% model accuracy and provide interactive dashboards for clean and meaningful insights to the user.

2. Objectives

To build a real-time AQI monitoring system that predicts air quality based on pollutant data.

To integrate Machine Learning models that forecast future AQI levels.

To design a user-friendly Dashboard showing current & historical pollutant trends.

To build and expose REST APIs for communication between backend and frontend.

To store AQI, pollutant levels, and user data using MySQL, MongoDB, and Firebase.

To deploy a scalable and modular system fit for real-world city-level monitoring

---

(Day 2)

3. Frontend Technologies

The frontend of the Air Aware system focuses on speed, responsiveness, and real-time data updates.

Technologies Used

- **HTML5, CSS3** – Page structure & styling
- **JavaScript** – API integration, DOM updates
- **React.js** – Component-based architecture, dynamic dashboard rendering

Frontend Features

- Real-time AQI display
- Historical trends (line charts, bar charts)
- Dynamic cards for pollutant levels (PM2.5, PM10, etc.)
- Weather visuals based on AQI
- Forecast page showing next-day or next-hour predictions
- User authentication (Login / Signup via Firebase)
- Downloadable reports (PDF/DOC)

4. Backend Technologies

The backend forms the core of data processing and prediction.

Technologies Used

- **Python**
- **FastAPI** – For high-performance REST APIs

- **Flask API** – Lightweight microservices
- **Django REST Framework** – For structured and scalable APIs
- **Node.js** – For additional API calls or microservice expansion

Backend Responsibilities

- Receive input payload from frontend
- Preprocess pollutant data
- Load ML model & run prediction
- Return AQI score, category, and analysis
- Handle database communication
- Serve downloadable reports
- Handle authentication using Firebase tokens

5. Databases Used

1. MySQL

- Structured pollutant and user data
- Used for AQI trend storage

2. MongoDB

- NoSQL
- Suitable for sensor data due to frequent updates

3. Firebase

- Authentication – Sign up / Login
- Real-time cloud storage

Database Concepts Applied

- CRUD operations
- DBMS architecture (Relational + Non-relational)

- Normalization rules:
  - **1NF**: Single atomic values
  - **2NF**: No partial dependency
  - **3NF**: No transitive dependency
  - **BCNF, 4NF, 5NF** for redundancy removal

6. Dataset Used

Sources

- Kaggle
- Hugging Face
- OpenAQ API
- Maharashtra State Pollution Data

Dataset Features

- PM2.5 (Particulate Matter 2.5)
- PM10 (Particulate Matter 10)
- NO2 (Nitrogen Dioxide)
- SO2 (Sulphur Dioxide)
- CO (Carbon Monoxide)
- O3 (Ozone)
- Temperature, Humidity (optional)
- Wind Speed (optional)

Dataset is cleaned, normalized, missing values handled, and used for model training.

(Day 3)

7. API Implementation

APIs are developed with:

• FastAPI – fast Python-based API

• Flask API – lightweight API handling

• Django REST Framework – structured backend

Common API Methods:

• GET – retrieve AQI, pollution levels, history

• POST – send inputs (payload) for predictions

Testing done via Postman.

(Day 4)

8. Git Version Control

Commands Used:

Commands Used

- git add filename
- git commit -m "message"
- git push
- git pull
- git fetch
- git branch --all
- git checkout -b newbranch

Branches are created using: git checkout -b <branchname>

9. Virtual Environment & Dependencies

• python -m venv env

• requirements.txt stores all required libraries

• pip install -r requirements.txt installs dependencies

• deactivate exits the virtual environment

(Day 5)

10. Database Concepts & Normalization

DBMS handles storage of AQI, pollutant logs, and historical data.

Normalization Rules Used:

• 1NF, 2NF, 3NF, BCNF (to reduce redundancy)

• Atomic values, no partial dependency, no transitive dependency

Data Types:

• Master data – rarely changes

• Transaction data – frequently updated (AQI, pollutants)

11. UI/UX Pages

• Dashboard – real-time AQI

• About Page – system info

• Reports Page – downloadable AQI reports

• Login / Sign-up Page

• Forecast Page – predicted AQI

(Day 6)

12. AI & ML Models

ML Techniques Used:

• Supervised Learning – Regression, Classification

• Unsupervised Learning – Clustering

• Reinforcement Learning (conceptual)

Approaches Used

- **Supervised Learning**
  - Regression for AQI prediction
  - Classification to categorize AQI (Good/Moderate/Severe)
- **Unsupervised Learning**
  - Clustering pollutant patterns
- **Reinforcement Learning** (conceptual only)

Model Goals

- Achieve **95%–98% accuracy**
- Predict AQI from pollutant inputs
- Predict next-day AQI based on historical patterns

Embedding & Vectors

- PostgreSQL + pgAdmin used for vector storage
- NLTK used for tokenization, embedding generation

13. Embeddings & Vector Storage

• PostgreSQL + pgAdmin used for vector storage

• Vector = numerical representation of text

• Embeddings created using NLTK and other NLP libraries

(Day 7)

Git commands and steps to clone git repository

(Day 8)

1. Artificial Intelligence (AI)

definition:

AI is the ability of machines to perform tasks that typically require human intelligence.

Details

- AI includes techniques that allow computers to learn patterns, make decisions, and solve problems.
- AI helps machines mimic human actions such as learning, reasoning, and problem-solving.

Subfields

- Machine Learning
- Deep Learning
- NLP
- Reinforcement Learning
- Knowledge-Based Systems

---

2. Machine Learning (ML)

definition:

ML is a method that enables machines to learn from data without being explicitly programmed.

Details

- Algorithms improve automatically with experience.
- Used in prediction, pattern recognition, and classification.

Types

Supervised Learning:

Learns from labelled data (input + output given).

Unsupervised Learning:

Learns from unlabelled data by finding patterns.

Reinforcement Learning:

Learns through rewards and penalties by interacting with an environment.

---

3. Deep Learning (DL)

definition:

DL is a subset of ML that uses multi-layered neural networks to learn complex patterns.

Details

- Works well with images, audio, text, and video.
- Uses large datasets and high computation.

Neural Network Types

- ANN – Artificial Neural Network
- CNN – Convolutional Neural Network
- RNN – Recurrent Neural Network
- LSTM – Long Short-Term Memory

---

4. Artificial Neural Network (ANN)

definition:

ANN is a network of neurons inspired by the human brain used for pattern recognition.

---

5. Convolutional Neural Network (CNN)

definition:

CNN is a neural network designed for image and video processing using convolutional filters.

---

6. Recurrent Neural Network (RNN)

definition:

RNN is a neural network that processes sequential data using loops to remember previous inputs.

---

7. Long Short-Term Memory (LSTM)

definition:

LSTM is an advanced RNN that can capture long-term dependencies using gates.

Details

- Designed to handle vanishing gradient problems.
- Works extremely well for time-series and text data.

Three Gates

- Forget Gate → removes unnecessary info
- Input Gate → adds new info
- Output Gate → passes info to next step

(Day 9)

8. Natural Language Processing (NLP)

definition:

NLP is a field of AI that helps computers understand and generate human language.

Applications

- Chatbots
- Sentiment Analysis
- Text Classification
- Translation

Challenges

- Grammar
- Context
- Emotions
- Ambiguity

---

9. NLP Processing Steps

1. Tokenization

definition: Breaking text into smaller units such as words or sentences.

2. Stop Word Removal

definition**:** Removing common words that carry little meaning (like “is”, “the”).

3. Stemming

definition: Reducing words to their root form (play → playing → play).

4. Lemmatization

definition**:** Converting words to their dictionary form (better → good).

5. POS Tagging

definition**:** Identifying whether a word is a noun, verb, adjective, etc.

6. NER – Named Entity Recognition

definition: Identifying names, places, dates in text.

7. TF-IDF

definition**:** A technique to measure how important a word is in a document.

---

10. Logistic Regression

definition**:**

A classification algorithm that predicts probability of a binary outcome.

Details

- Uses sigmoid function.
- Output: 0 or 1 based on probability.

(Day 10)

11. Support Vector Machine (SVM)

definition:

SVM is a supervised algorithm that finds the best boundary (hyperplane) to separate classes.

Details

- Uses margins and support vectors.
- Works well with high-dimensional data.

---

12. Reinforcement Learning (RL)

definition:

RL is a learning method where an agent learns by interacting with an environment and receiving rewards.

Key Concepts

- Agent
- Environment

- Reward
- Policy

Multi-Agent System

Multiple agents working together or competing in an environment.

(Day 11)

13. CSS Key Concepts

Flexbox

definition: A CSS layout model for aligning items horizontally or vertically.

Sizing

- Intrinsic: natural size
- Extrinsic: size defined using CSS

Overflow (visible/hidden/scroll/auto)

definition: Controls what happens when content exceeds the container's size.

Box Sizing

definition: Defines how element width/height is calculated (content-box, border-box).

Flex Properties

- flex-direction
- justify-content
- align-items
- flex-wrap
- order
- align-self

(Day 12)

14. API Basics (OpenAI/Gemini)

definition:

APIs allow your application to communicate with external AI models through requests.

Steps

- Install library
- Import
- Add API key
- Send prompt
- Get response

---

15. Decision Tree

definition:

A tree-like model that splits data into branches to make decisions.

Key Concepts

- Entropy
- Information Gain
- Parent & Child nodes

(Day 13)

16. Random Forest

definition:

An ensemble model that combines many decision trees for more accurate predictions.

Details

- Uses majority vote

- Reduces overfitting

---

17. K-Nearest Neighbours (KNN)

definition:

A distance-based algorithm where classification is done based on nearest data points.

Distance Formula:

$\sqrt{((x1 - x2)^2 + (y1 - y2)^2)}$

---

18. K-Means Clustering

definition:

An unsupervised algorithm that groups data into K clusters based on similarity.

Steps

1. Choose K
2. Select centroids
3. Assign points
4. Recalculate
5. Repeat

---

19. Linear Regression

definition:

A supervised model that predicts continuous numeric values.

Formula

$Y = mX + C$

---

20. XGBoost

definition:

A powerful boosting algorithm that builds small decision trees and combines them for strong predictions.

Details

- Fast
- Uses regularization
- High accuracy

(Day 14)

1. MySQL

- MySQL is used to store, manage, and retrieve data in a structured manner.
- Helps in handling large datasets, performing CRUD operations (Create, Read, Update, Delete).
- Widely used in web apps, data analytics, dashboards, and backend systems.
- Supports data integrity, indexing, relationships, and fast querying.

---

2. Basic SQL Commands (DDL + DML + DQL)

DDL → Data Definition Language

Used to define or modify database structure:

- **CREATE** – creates database/table
- **ALTER** – modifies table structure
- **RENAME** – renames table
- **DROP** – deletes table
- **TRUNCATE** – deletes all rows (faster than DELETE)

DML → Data Manipulation Language

Used to modify table data:

- **INSERT** – add new rows
- **UPDATE** – modify existing rows
- **DELETE** – remove rows

DQL → Data Query Language

Used to fetch data:

- **SELECT** – retrieve data

---

3. SELECT Statement

- Fetches data from a table.

Example:

SELECT * FROM Customers;

SELECT DISTINCT

Used to show unique values from a column:

SELECT DISTINCT Country FROM Customers;

---

4. WHERE Clause

Used to filter records.

Operators used in WHERE:

- =, >, <, >=, <=
- <> or != → not equal
- BETWEEN
- LIKE
- IN
- NOT, AND, OR

Examples:

SELECT * FROM Customers WHERE Country = 'Mexico';

SELECT * FROM Customers WHERE Age BETWEEN 20 AND 30;

SELECT * FROM Customers WHERE City IN ('Mumbai', 'Pune');

---

5. ORDER BY

Sort results in ascending/descending order:

SELECT * FROM Customers ORDER BY Name ASC;

SELECT * FROM Customers ORDER BY Age DESC;

---

6. INSERT INTO

INSERT INTO Customers (Name, City) VALUES ('Sahil', 'Nagpur');

---

7. UPDATE

UPDATE Customers SET City = 'Mumbai' WHERE ID = 5;

---

8. DELETE

DELETE FROM Customers WHERE ID = 5;

---

9. Aggregate Functions

- **MIN()** → lowest value
- **MAX()** → highest value
- **COUNT()** → number of rows
- **AVG()** → average
- **SUM()** → total sum

Example:

SELECT COUNT(*) FROM Orders;

---

10. LIKE Operator (Pattern Matching)

- % → multiple characters
- _ → single character

Examples:

SELECT * FROM Customers WHERE Name LIKE 'S%';

SELECT * FROM Customers WHERE City LIKE 'Mumbai';

---

11. HAVING Clause

Used for filtering after GROUP BY.

Example:

SELECT City, COUNT(*)

FROM Customers

GROUP BY City

HAVING COUNT(*) > 5;

---

12. JOINS

INNER JOIN

Returns matching records from both tables.

SELECT *

FROM Orders

INNER JOIN Customers

ON Orders.CustomerID = Customers.ID;

LEFT JOIN

Returns all records from left table, even if no match.

RIGHT JOIN

Returns all records from right table.

CROSS JOIN

Returns cartesian product of both tables.

---

13. UNION vs UNION ALL

UNION

- Combines results of two SELECT queries
- Removes duplicates
- Slower due to distinct check

UNION ALL

- Combines results
- Allows duplicates
- Faster

(Day 15)

1. STRING METHODS

1. split()

Definition: Splits a string into a list based on a separator.

Example:

```
s = "apple banana mango"
print(s.split())        # ['apple', 'banana', 'mango']
```

```
print(s.split("a"))      # ['', 'pple b', 'n', 'n', ' m', 'ngo']
```

---

2. join()

Definition: Joins a list of strings into one string using a separator.

Example:

```
lst = ["a", "b", "c"]

print("-".join(lst))     # a-b-c
```

---

3. replace()

Definition: Replaces a substring with another substring.

Example:

```
text = "hello world"

print(text.replace("world", "Python"))  # hello Python
```

---

4. lower()

Definition: Converts all characters to lowercase.

Example:

```
print("HELLO".lower())   # hello
```

---

5. upper()

Definition: Converts all characters to uppercase.

Example:

```
print("hello".upper())   # HELLO
```

---

6. startswith()

Definition: Checks if a string starts with a given value.

Example:

```
print("python".startswith("py"))  # True
```

---

7. endswith()

Definition: Checks if a string ends with a given value.

Example:

```
print("hello.txt".endswith(".txt"))  # True
```

---

8. find()

Definition: Returns index of first occurrence of substring, -1 if not found.

Example:

```
print("apple".find("p"))  # 1
print("apple".find("z"))  # -1
```

---

9. isdigit()

Definition: Checks whether all characters are digits.

Example:

```
print("123".isdigit())  # True
print("12a".isdigit())  # False
```

---

10. isalpha()

Definition: Checks whether all characters are alphabets.

Example:

```
print("abc".isalpha())  # True
```

(Day 16)

2. LIST METHODS

1. append()

Definition: Adds an item at end of list.

Example:

```
lst = [1, 2]
lst.append(3)
print(lst)  # [1, 2, 3]
```

---

2. extend()

Definition: Adds elements of another iterable to list.

Example:

```
lst = [1, 2]
lst.extend([3, 4])
print(lst)  # [1, 2, 3, 4]
```

---

3. insert()

Definition: Inserts item at a given index.

Example:

```
lst = [1, 3]
lst.insert(1, 2)
print(lst)  # [1, 2, 3]
```

---

4. remove()

Definition: Removes first matching value from list.

Example:

```
lst = [1, 2, 2, 3]
lst.remove(2)
print(lst)  # [1, 2, 3]
```

---

5. pop()

Definition: Removes item at index (default last) and returns it.

Example:

```
lst = [10, 20, 30]
print(lst.pop())     # 30
print(lst.pop(0))    # 10
```

---

6. index()

Definition: Returns index of first occurrence.

Example:

```
print([10, 20, 30].index(20))  # 1
```

---

7. count()

Definition: Counts occurrences of value.

Example:

```
print([1, 2, 2, 3].count(2))  # 2
```

---

8. sort()

Definition: Sorts list in ascending order (in-place).

Example:

```
lst = [3, 1, 2]
lst.sort()
print(lst)  # [1, 2, 3]
```

---

9. reverse()

Definition: Reverses list in-place.

Example:

```
lst = [1, 2, 3]
lst.reverse()
print(lst)  # [3, 2, 1]
```

---

3. DICTIONARY METHODS

1. get()

Definition: Returns value for key, else default.

Example:

```
d = {"a": 1}
print(d.get("a"))       # 1
print(d.get("b", 0))    # 0
```

---

2. keys()

Definition: Returns list-like view of keys.

Example:

```
d = {"a": 1, "b": 2}
print(d.keys())  # dict_keys(['a', 'b'])
```

---

3. values()

Definition: Returns view of dictionary values.

Example:

```
print(d.values())  # dict_values([1, 2])
```

---

4. items()

Definition: Returns list-like view of key-value pairs.

Example:

```
print(d.items())  # dict_items([('a',1),('b',2)])
```

---

5. update()

Definition: Updates dictionary with another dictionary.

Example:

```
d = {"a": 1}
d.update({"b": 2})
print(d)  # {'a': 1, 'b': 2}
```

---

6. pop()

Definition: Removes key and returns its value.

Example:

```
d = {"a": 1, "b": 2}
print(d.pop("a"))   # 1
```

```
print(d)          # {'b': 2}
```

---

7. popitem()

Definition: Removes last inserted key-value pair.

Example:

```
d = {"a": 1, "b": 2}
print(d.popitem())      # ('b', 2)
```

---

8. clear()

Definition: Removes all items in dictionary.

Example:

```
d.clear()
print(d)    # {}
```

---

4. SET METHODS

1. add()

Adds element to set.

```
s = {1, 2}
s.add(3)
```

---

2. remove()

Removes element, errors if not present.

```
s.remove(2)
```

---

3. discard()

Removes element, **no error** if not present.

s.discard(10)

---

4. pop()

Removes and returns any random element.

print(s.pop())

---

5. clear()

Removes all elements.

s.clear()

---

6. union()

Returns elements of both sets.

a = {1,2,3}

b = {3,4,5}

print(a.union(b)) # {1,2,3,4,5}

---

7. intersection()

Common elements.

print(a.intersection(b)) # {3}

---

8. difference()

Elements in A not in B.

print(a.difference(b)) # {1,2}

---

5. FILE HANDLING METHODS

1. open()

Opens a file.

f = open("demo.txt", "r")

---

2. read()

Reads entire file.

content = f.read()

---

3. readline()

Reads one line at a time.

line = f.readline()

---

4. readlines()

Reads all lines as list.

lines = f.readlines()

---

5. write()

Writes a string to file.

f = open("demo.txt", "w")

f.write("Hello")

---

6. writelines()

Writes list of strings.

f.writelines(["Hello\n", "World\n"])

---

7. close()

Closes the file.

f.close()

---

6. GENERAL PURPOSE FUNCTIONS

1. len()

Returns length.

```
print(len([1,2,3]))  # 3
```

---

2. range()

Returns sequence of numbers.

```
for i in range(1,5):
    print(i)
```

---

3. print()

Prints output.

---

4. type()

Returns data type.

```
print(type(10))
```

---

5. id()

Returns memory location of object.

```
x = 10
```

```
print(id(x))
```

---

6. sorted()

Returns a sorted list.

```
print(sorted([3,1,2]))  # [1,2,3]
```

---

7. enumerate()

Adds counter to iterable.

```
for i, val in enumerate(["a","b"]):
    print(i, val)
```

---

8. zip()

Combines elements from multiple iterables.

```
for x, y in zip([1,2,3], ['a','b','c']):
    print(x, y)
```

(Day 17)

1. TYPE CONVERSION FUNCTIONS

1. int()

Converts value to integer.

```
print(int("12"))     # 12
```

2. float()

Converts value to floating-point number.

```
print(float("3.14"))  # 3.14
```

3. str()

Converts value to string.

print(str(25)) # "25"

4. list()

Converts iterable into list.

print(list("abc")) # ['a','b','c']

5. dict()

Creates dictionary from pairs.

print(dict([(1, 'a'), (2, 'b')]))

6. set()

Converts iterable into set.

print(set([1, 2, 2, 3])) # {1,2,3}

---

2. MATHEMATICAL FUNCTIONS

1. abs()

Absolute value.

print(abs(-10)) # 10

---

2. sum()

Sum of iterable elements.

print(sum([1,2,3])) # 6

---

3. min()

Minimum value.

print(min(1, 5, -2)) # -2

---

4. max()

Maximum value.

print(max(1,5,9)) # 9

---

5. pow()

Power calculation.

print(pow(2, 3)) # 8

---

6. round()

Rounds number to nearest integer.

print(round(3.67)) # 4

(Day 18)

3. FUNCTIONAL PROGRAMMING TOOLS

1. filter()

Filters iterable based on condition.

nums = [1,2,3,4]

print(list(filter(lambda x: x%2==0, nums))) # [2,4]

---

2. map()

Applies a function to each element.

nums = [1,2,3]

print(list(map(lambda x: x*2, nums))) # [2,4,6]

---

3. reduce()

Applies function cumulatively → reduces iterable to a single value.

from functools import reduce

print(reduce(lambda a,b: a+b, [1,2,3,4])) # 10

---

4. lambda

Anonymous function.

add = lambda a,b: a+b

print(add(3,4)) # 7

---

4. I/O FUNCTIONS

1. input()

Takes input from user (string).

name = input("Enter name:")

2. format()

Formats strings.

print("My name is {}".format("Sahil"))

---

5. CLASS & OBJECT RELATED FUNCTIONS

1. del

Deletes a variable, attribute or object.

del x

---

2. getattr()

Gets value of an object attribute.

class A: x = 10

print(getattr(A, "x")) # 10

---

3. setattr()

Sets value of attribute.

setattr(A, "x", 20)

---

4. hasattr()

Checks if attribute exists.

print(hasattr(A, "x")) # True

---

5. delattr()

Deletes attribute from object.

delattr(A, "x")

---

6. isinstance()

Checks if object is instance of class.

print(isinstance(3, int)) # True

7. issubclass()

Checks if a class is subclass of another.

print(issubclass(bool, int)) # True

(Day 19)

6. MISCELLANEOUS FUNCTIONS

1. globals()

Returns global symbol table (global variables).

```
print(globals())
```

---

2. locals()

Returns local variable dictionary.

```
def f():
    x = 10
    print(locals())
```

---

3. callable()

Checks if object can be called (like functions).

```
print(callable(print))  # True
```

---

4. eval()

Evaluates expression string.

```
print(eval("5+10"))  # 15
```

---

5. exec()

Executes dynamically created Python code.

```
exec("x = 10")
print(x)
```

7. ERROR HANDLING

try–except–else–finally

```
try:
    x = 10 / 0
except ZeroDivisionError:
```

```
    print("Error!")
else:
    print("No error")
finally:
    print("Always runs")
```

Meaning:

- **try:** Code that may raise error
- **except:** Handles error
- **else:** Runs only when no error
- **finally:** Runs always (cleanup)

---

8. GC (GARBAGE COLLECTION)

Python automatically manages memory using GC.

1. del()

Removes reference to object.

2. gc.collect()

Manually triggers garbage collection.

```
import gc
gc.collect()
```

Use cases:

- Free unused objects
- Manage memory in large programs
- Clean cyclic references

(Day 20)

9. WORKING WITH ITERABLES

1. iter()

Returns iterator object.

```
it = iter([1,2,3])
```

---

2. next()

Returns next element from iterator.

```
print(next(it))  # 1
print(next(it))  # 2
```

---

10. DECORATORS & METAPROGRAMMING

1. @staticmethod

Method that does not use class or object.

```
class A:
    @staticmethod
    def show():
        print("Hello")
```

---

2. @classmethod

Method that uses class (cls) instead of object.

```
class A:
    count = 0
    @classmethod
    def inc(cls):
        cls.count += 1
```

11. CONTEXT MANAGERS

Used to properly manage resources (files, DB connections etc.)

```
with open("demo.txt", "r") as f:
    data = f.read()
```

Automatically:

- opens file
- reads it
- closes file (even if error occurs)

(Day 21)

SDLC – Software Development Life Cycle

Definition:
SDLC is a structured process used for planning, creating, testing, deploying, and maintaining software. It ensures high-quality software that meets client requirements and is delivered on time and within budget.

STAGES OF SDLC

1. Requirement Analysis

Role: Functional Consultant / Business Analyst

- Interacts directly with the client
- Understands business requirements
- Converts requirements into Functional Specification Document (FSD)
- Reviews, tests, and validates requirements
- Shares FSD with technical team

Output: Functional Specification Document

---

2. Design Phase

Role: Developer / Programmer / Technical Team

- Converts FSD into Technical Specification Document (TSD)
- Decides:
    - Architecture
    - Technology stack
    - Database design
- Prepares prototype or mock-up

Output: Technical Specification Document, Prototype

---

3. Development Phase

- Actual coding is done by developers
- Based on Technical Specifications
- Features and modules are implemented
- Code is version controlled

---

4. Unit Testing

Performed by: Developer

- Testing of individual modules
- Ensures each function works correctly

Example:

```
def add(a, b):
   return a + b


# Unit Test
```

assert add(2, 3) == 5

---

5. Testing / QA Phase

Role: QA (Testing) Team

- Prepares Test Scripts based on FSD
- Tests different scenarios:
    - Positive cases
    - Negative cases
    - Alternate scenarios
- Identifies bugs and reports to developers

---

6. Deployment

3 Main Servers / Environments

1. **Development Server**
    - Used by developers
    - Coding and unit testing done
2. **Quality (QA / Testing) Server**
    - Used by QA team
    - Functional & system testing
3. **Production Server**
    - Live environment
    - Used by end users / customers

---

7. Maintenance

- Bug fixing

- Performance improvement
- Security updates
- Feature enhancements
- Continuous support

SUPPORTING TEAMS IN SDLC

Administration Team

- Manages:
    - Network
    - Database
    - Application servers
- Handles infrastructure issues

Security Team

- Firewall configuration
- Data protection
- Securing production environment
- Prevents unauthorized access

Product / Project Management

- Manages:
    - Developers
    - Designers
    - Testers

- Communicates with clients
- Tracks project timeline and delivery

Sales & Marketing Team

- Promotes the product
- Advertising and branding
- Customer acquisition
- Works collaboratively to sell solutions

Human Resource (HR) Team

- Bridge between employees and management
- Hiring, onboarding
- Employee welfare
- Policy management

Documentation Team

- Prepares:
    - User manuals
    - Policies
    - Product documentation
- Plays a vital role in compliance and handover

Support Team

- Provides support to:
    - Functional team

    - Developers
    - Testers
    - Customers
- Resolves issues and tickets
- Ensures smooth operation

WATERFALL MODEL

Definition:

Waterfall model is a traditional SDLC model introduced by Winston W. Royce in 1970. It follows a linear and sequential approach.

Phases of Waterfall Model

1. Requirement Analysis
2. Design
3. Development
4. Testing
5. Deployment
6. Maintenance

Features

- Sequential flow
- Documentation-driven
- Quality control at each phase
- No overlapping of phases

Advantages

- Easy to understand and manage
- Clear structure and milestones

- Suitable for small projects
- Well-defined documentation

Disadvantages

- No client feedback during development
- No flexibility
- Changes are costly
- Not suitable for large or dynamic projects

AGILE MODEL

Definition:

Agile is a flexible and iterative approach to software development that focuses on continuous delivery and customer feedback.

Working of Agile Model

1. **Pre-Planning**
    - Requirement discussion
    - Backlog creation
2. **Planning**
    - Sprint planning
    - Task allocation
3. **Development & Testing**
    - Coding and testing done together
    - Continuous integration
4. **Review & Feedback**
    - Client review

- Feedback collected

5. **Release / Deploy**
    - Incremental product release
6. **Iterate**
    - Repeat cycle for improvements

Features

- Iterative development
- Continuous feedback
- Flexible to changes
- Faster delivery

Advantages

- High customer satisfaction
- Early bug detection
- Adaptable to changes
- Suitable for complex projects

Disadvantages

- Less documentation
- Requires experienced team
- Difficult to estimate timeline
- Not ideal for fixed-scope projects

(Day 22)

Milestone 3

(Day 23)

A* (A-Star) ALGORITHM

Definition

A* is a path-finding and graph traversal algorithm used to find the shortest path between a start node and a goal node.

It is widely used in AI, games, GPS navigation, and robotics.

Formula Used

$f(n) = g(n) + h(n)$

- **g(n):** Cost from start node to current node
- **h(n):** Heuristic (estimated cost from current node to goal)
- **f(n):** Total estimated cost

Working Steps

1. Add start node to open list
2. Choose node with lowest f(n)
3. Move it to closed list
4. Expand neighbors and calculate f(n)
5. Repeat until goal is reached

Features

- Combines Dijkstra's algorithm and Greedy Best First Search
- Guarantees shortest path if heuristic is admissible
- Efficient and optimal

Advantages

- Finds optimal solution
- Faster than many other algorithms
- Heuristic based search

Disadvantages

- Requires memory
- Depends on quality of heuristic

Example

Finding shortest route in Google Maps from home to office using distance estimation.

SORTING ALGORITHMS

1. Bubble Sort

Definition

Bubble Sort repeatedly swaps adjacent elements if they are in the wrong order.

Working

- Compare adjacent elements
- Swap if needed
- Repeat until list is sorted

Example

Array: [5, 3, 1]

Steps:

5 3 1 → 3 5 1

3 5 1 → 3 1 5

3 1 5 → 1 3 5

Sorted Array: [1, 3, 5]

---

2. Selection Sort

Definition

Selection Sort finds the minimum element and places it at the correct position.

Example

Array: [64, 25, 12, 22]

Steps:

Select 12 → [12, 25, 64, 22]

Select 22 → [12, 22, 64, 25]

Select 25 → [12, 22, 25, 64]

Sorted Array: [12, 22, 25, 64]

---

3. Insertion Sort

Definition

Insertion Sort builds sorted array one element at a time.

Example

Array: [8, 3, 5]

Steps:

[8]

[3, 8]

[3, 5, 8]

Sorted Array: [3, 5, 8]

---

4. Merge Sort

Definition

Merge Sort uses Divide and Conquer technique.

Working

1. Divide array into halves

2. Sort each half
3. Merge sorted halves

Example

Array: [38, 27, 43, 3]

Split → [38,27] [43,3]

Sort → [27,38] [3,43]

Merge → [3,27,38,43]

5. Quick Sort

Definition

Quick Sort selects a pivot element and partitions array around it.

Example

Array: [10, 7, 8, 9]

Pivot = 9

Left → [7,8]

Pivot → 9

Right → [10]

Sorted Array: [7, 8, 9, 10]

---

6. Heap Sort

Definition

Heap Sort uses Binary Heap data structure.

Working

1. Build Max Heap
2. Swap root with last element
3. Heapify remaining elements

Example

Array: [4, 10, 3, 5, 1]

Max Heap → [10, 5, 3, 4, 1]

Sorted → [1, 3, 4, 5, 10]

(Day 24)

Backpropagation Algorithm

Basic Algorithm

- Backpropagation is used to train neural networks
- It adjusts weights and biases to reduce errors
- Errors are calculated and resolved iteratively
- Error is propagated backward through the network
- Helps in learning complex data patterns

Neural Network Layers

1. Input Layer
2. Hidden Layer
3. Output Layer

Activation Functions

- **Sigmoid Function** – outputs values between 0 and 1
- **ReLU (Rectified Linear Unit)** – improves learning speed

Error Calculation

- **Mean Squared Error (MSE)** is used to calculate error

Optimization

- **Gradient Descent** is used to minimize error by updating weights

---

(Day 25)

AI Search Techniques

1. Depth First Search (DFS)

- Explores graph by selecting one path deeply
- Backtracks after reaching the end
- Uses stack / recursion
- Memory efficient but may not give shortest path

2. Breadth First Search (BFS)

- Starts from root node
- Explores nodes level by level
- Uses queue
- Guarantees shortest path in unweighted graphs

3. Depth Limited Search

- DFS with a fixed depth limit
- Avoids infinite loops
- Used when depth is known

4. Uniform Cost Search

- Expands node with lowest cost
- Uses priority queue
- Can use heap queue library
- Guarantees optimal solution

5. Bidirectional Search

- Two-way search:
    - One from start node
    - One from goal node

- Stops when both searches meet in the middle
- Faster than single-direction search

6. Beam Search

- Limited version of BFS
- Keeps only best k nodes at each level
- Faster but may miss optimal solution

Travelling Salesman Problem (TSP)

- Starts from a given city
- Visits all cities exactly once
- Returns to the starting city
- Goal: shortest possible path
- Uses Nearest Neighbour Algorithm
- NP-Hard problem

Machine Learning Techniques

1. SVM (Support Vector Machine)

- Supervised learning algorithm
- Used for classification and regression
- Finds optimal separating hyperplane

2. GMM (Gaussian Mixture Model)

- Probabilistic clustering algorithm
- Uses multiple Gaussian distributions

3. Gaussian Distribution

- Bell-shaped curve
- Defined by mean and variance

4. Fuzzy Logic

- Handles uncertainty
- Values range between 0 and 1
- Used in decision-making systems

5. PCA (Principal Component Analysis)

- Dimensionality reduction technique
- Reduces noise and irrelevant features
- Improves performance and speed
- Converts data into principal components