**LUNG CANCER DETECTION**

MINOR PROJECT REPORT

By

**VANSH AGRAWAL (RA2311026010120)**

#### HIRAV KADIKAR (RA2311026010111)

Under the guidance of

##### Dr. Kiruthika M

In partial fulfilment of the requirements for the degree of

**B. TECH COMPUTER SCIENCE ENGINEERING W/S IN AIML**

****

**DEPARTMENT OF COMPUTATIONAL INTELLIGENCE COLLEGE OF ENGINEERING AND TECHNOLOGY SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR-603 203**

**MAY 2025**



**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY** **KATTANKULATHUR–603 203**

# BONAFIDE CERTIFICATE

**Register no. RA2311026010120, RA2311026010111** Certified to be the bonafide work done by **Vansh Agrawal, Hirav Kadikar** of II year/IV sem B.TECH CSE with Specialization in AIML in the Project Course – **21CSC205P Database Management Systems** in **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**, Kattankulathur for the academic year 2024-2025.

**DATE:** 7 May 2025

**Faculty in Charge HEAD OF THE**

#### DEPARTMENT DEPARTMENT

Dr Kiruthika M Dr. Annie Uthra R

Assistant Professor Professor & Head

Department of CINTEL Department of CINTEL

SRMSIT -KTR SRMSIT -KTR

# ABSTRACT

The Lung Cancer Predictor is a robust software solution designed to streamline and optimize the process of lung cancer risk assessment. This project aims to provide healthcare professionals and patients with an efficient and intuitive platform for managing medical histories, risk assessments, predictions, and patient interactions.

DBMS in the Lung Cancer Predictor serves as the foundational framework for storing, organizing, and processing data related to patient information, medical histories, risk assessments, and prediction results. By centralizing data management and providing efficient data retrieval mechanisms, DBMS facilitate more accurate risk assessments, improve patient care, and enable data-driven medical decision-making.

The advantages of using DBMS in the Lung Cancer Predictor are diverse. Firstly, DBMS ensure data accuracy and consistency by enforcing data integrity constraints, minimizing errors in risk assessment and medical history tracking. Secondly, DBMS support scalability, allowing the system to handle increased user demand, additional risk factors, and expanded medical data without compromising performance. Thirdly, DBMS enable secure multi-user access, concurrency control, and data privacy measures, ensuring reliable and efficient operations even during peak usage and high transaction volumes.

# TABLES OF CONTENT

|  |  |  |
| --- | --- | --- |
| **Chapter No** | **Chapter Name** | **Page No** |
| **1.** | Problem understanding, Identification of Entity and  Relationships, Construction of DB using ER Model for the project | **1** |
| **2.** | Design of Relational Schemas, Creation of Database Tables for the project. | **4** |
| **3.** | Complex queries based on the concepts of constraints, sets, joins, views, Triggers and Cursors. | **10** |
| **4.** | Analyzing the pitfalls, identifying the dependencies, and applying normalizations | **26** |
| **5.** | Implementation of concurrency control and recovery mechanisms | **30** |
| **6.** | Code for the project | **36** |
| **7.** | Result and Discussion (Screen shots of the implementation  with front end. | **51** |
| **8.** | Attach the Real Time project certificate / Online course  certificate | **57** |

**CHAPTER-1**

**LUNG CANCER PREDICTOR SYSTEM**

**ENTITY-RELATIONSHIP DIAGRAM-**

Brief Introduction:

An E-R(Entity-Relationship) diagram is a visual representation of the relationships between entities in a database. It's used to design databases by illustrating entities (such as people, objects, or concepts) and the relationships between them.

Creating an Entity-Relationship (ER) diagram for a Lung Cancer Predictor system involves identifying key entities, their attributes, and relationships within the system. Here's a simplified ER diagram for the Lung Cancer Predictor system:

ENTITIES:

1. 1. Users - Attributes: User\_ID, Name, Email, Password, Registration\_Date, Last\_Login

2. Medical\_History - Attributes: History\_ID, User\_ID, Family\_History, Smoking\_History, Previous\_Diseases, Occupational\_Exposure

3. Predictions - Attributes: Prediction\_ID, User\_ID, Risk\_Score, Prediction\_Date, Symptoms\_List, Recommendations

4. Symptoms - Attributes: Symptom\_ID, Symptom\_Name, Severity\_Level, Description, Category

5. Recommendations - Attributes: Recommendation\_ID, Risk\_Level, Medical\_Advice, Prevention\_Tips, Follow\_up\_Steps

6. User\_Feedback - Attributes: Feedback\_ID, User\_ID, Prediction\_ID, Rating, Comments, Feedback\_Date

7. Risk\_Factors - Attributes: Factor\_ID, Factor\_Name, Weight, Description, Category

8. Medical\_Reports - Attributes: Report\_ID, User\_ID, Report\_Date, Report\_Type, Findings, Doctor\_Notes

9. User\_Sessions - Attributes: Session\_ID, User\_ID, Login\_Time, Logout\_Time, IP\_Address

10. Prediction\_History - Attributes: History\_ID, User\_ID, Prediction\_ID, Date, Risk\_Score, Follow\_up\_Status

RELATIONSHIPS:

1. Relationship between USERS and MEDICAL\_HISTORY entities:

• One-to-One relationship, as each user has one medical history record.

2. Relationship between USERS and PREDICTIONS entities:

• One-to-Many relationship, as one user can have multiple predictions.

3. Relationship between PREDICTIONS and SYMPTOMS entities:

• Many-to-Many relationship, as predictions can include multiple symptoms and symptoms can be part of multiple predictions.

4. Relationship between PREDICTIONS and RECOMMENDATIONS entities:

• One-to-Many relationship, as one prediction can have multiple recommendations.

5. Relationship between USERS and USER\_FEEDBACK entities:

• One-to-Many relationship, as one user can provide multiple feedback entries.

6. Relationship between MEDICAL\_HISTORY and RISK\_FACTORS entities:

• Many-to-Many relationship, as medical history can include multiple risk factors and risk factors can be associated with multiple medical histories.

7. Relationship between USERS and MEDICAL\_REPORTS entities:

• One-to-Many relationship, as one user can have multiple medical reports.

8. Relationship between PREDICTIONS and PREDICTION\_HISTORY entities:

• One-to-One relationship, as each prediction has one history record.

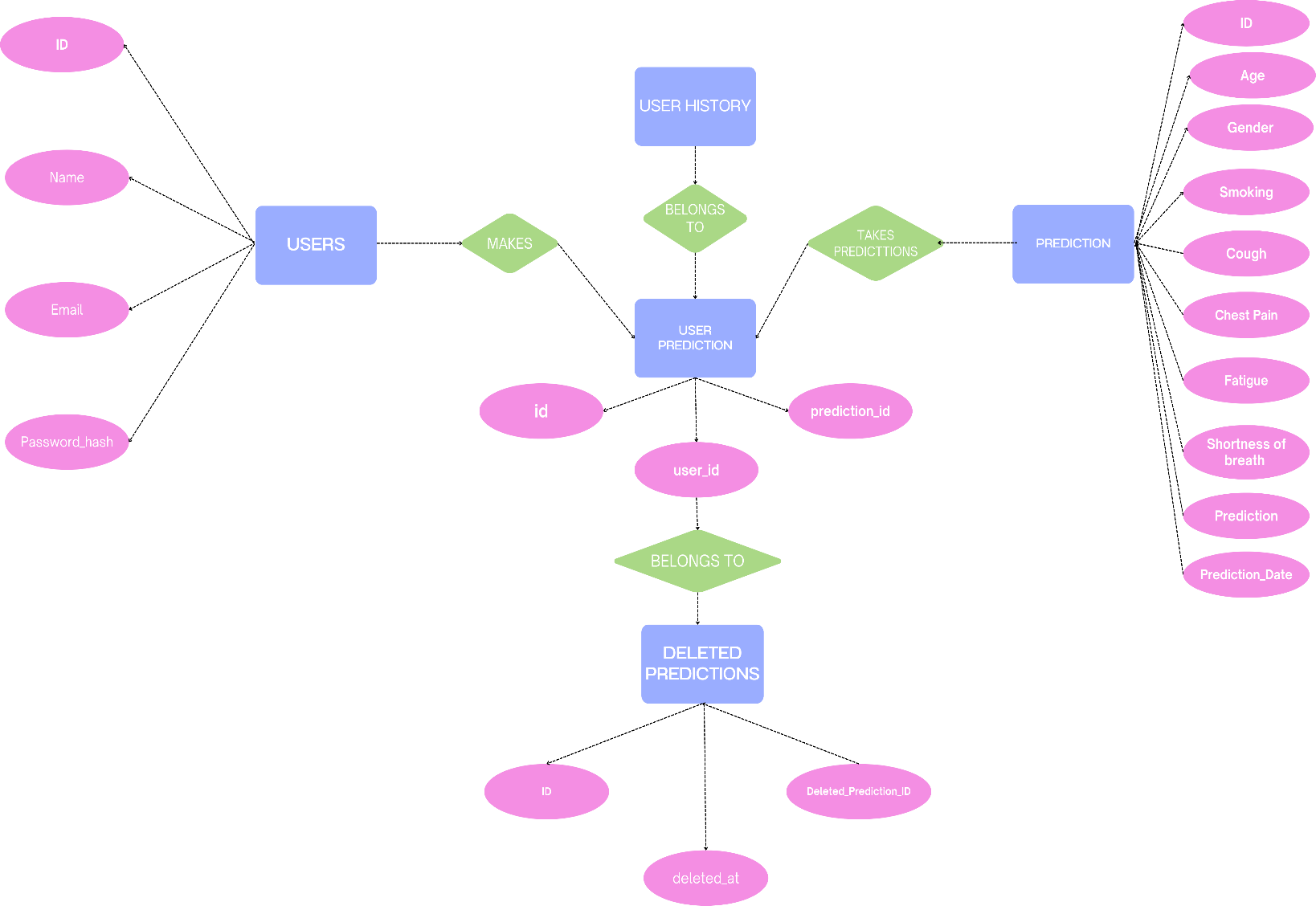
9. Relationship between USERS and USER\_SESSIONS entities:

• One-to-Many relationship, as one user can have multiple login sessions.

10. Relationship between PREDICTIONS and USER\_FEEDBACK entities:

• One-to-One relationship, as each prediction can have one feedback entry..

REPRESENTATION



**CHAPTER-2**

**CONVERTING ER DIAGRAM TO RELATIONAL TABLE**

**USERS:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **User\_ID (PK)** | **Name** | **Email** | **Password** | **Registration\_Date** | **Last\_Login** |

**MEDICAL\_HISTORY:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **History\_ID (PK)** | **User\_ID (FK)** | **Family \_History** | **Smoking \_History** | **Previous \_Diseases** | **Occupational \_Exposure** |

**PREDICTIONS:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Prediction\_ID (PK)** | **User\_ID (FK)** | **Risk\_Score** | **Prediction\_Date** | **Symptoms\_List** | **Recommendations** |

**SYMPTOMS:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Symptom\_ID (PK)** | **Symptom\_Name** | **Severity\_Level** | **Description** | **Category** |

**RECOMMENDATIONS:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Recommendation \_ID (PK)** | **Risk \_Level** | **Medical \_Advice** | **Prevention \_Tips** | **Follow\_up\_Steps** |

**USER\_FEEDBACK:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Feedback\_ID (PK)** | **User\_ID (FK)** | **Prediction\_ID (FK)** | **Rating** | **Comments** | **Feedback \_Date** |

**RISK\_FACTORS:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Factor\_ID (PK)** | **Factor\_Name** | **Weight** | **Description** | **Category** |

**MEDICAL\_REPORTS:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Report\_ID (PK)** | **User\_ID (FK)** | **Report\_Date** | **Report\_Type** | **Findings** | **Doctor \_Notes** |

**USER\_SESSIONS:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Session\_ID (PK)** | **User\_ID (FK)** | **Login\_Time** | **Logout\_Time** | **IP\_Address** |

**PREDICTION\_HISTORY:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **History\_ID (PK)** | **User\_ID (FK)** | **Prediction\_ID (FK)** | **Date** | **Risk\_Score** | **Follow\_up\_Status** |

**PITFALLS OF RELATIONAL DATABASE SYSTEM:**

**1. Inadequate Normalization: Normalization is the process of organizing data into tables to minimize redundancy and eliminate data anomalies. Failure to normalize data can result in redundant patient information and update anomalies. This can lead to data inconsistencies and poor performance in medical record management.**

**2. Overuse of NULL values: Using NULL values in medical records can make it difficult to query patient data and can lead to confusion when interpreting medical histories. Overuse of NULL values can also result in poor performance in risk assessment calculations.**

**3. Poor Indexing: Indexing is essential for efficient querying of patient data and medical histories. Poorly designed indexes can result in slow query performance when retrieving patient records and risk assessments.**

**4. Insufficient Primary and Foreign Keys: Primary and foreign keys establish relationships between patient records, medical histories, and predictions. Failure to implement these keys can result in data inconsistencies and poor performance in tracking patient outcomes.**

**5. Denormalization: While denormalization can improve query performance for risk assessments, it can also lead to data inconsistencies in medical records and update anomalies. Denormalization should be used sparingly and only after careful consideration of medical data integrity.**

**6. Failure to Plan for Growth: A medical database should be designed with future growth in mind, considering increasing patient records and new risk factors. Failure to plan for growth can result in poor performance, data inconsistencies, and costly database redesigns.**

**7. Lack of Documentation: A lack of documentation can make it difficult to understand the database design and lead to errors in medical data analysis and risk assessment reporting. Proper documentation is crucial for maintaining accurate patient records and risk predictions.**

**8. Security Concerns: Medical data requires strict security measures. Failure to implement proper security protocols can lead to unauthorized access to sensitive patient information and risk assessment data.**

**9. Data Validation: Insufficient validation of medical data input can lead to incorrect risk assessments and compromised patient care. Proper validation rules must be implemented for all medical data fields.**

**10. Backup and Recovery: Regular backups of medical data are crucial. Failure to implement proper backup and recovery procedures can result in loss of critical patient information and risk assessment history.**

**CREATING TABLES IN THE DATABASE**

-- Table: Transaction Log

CREATE TABLE IF NOT EXISTS transaction\_log (

id INT AUTO\_INCREMENT PRIMARY KEY,

transaction\_id VARCHAR(36) NOT NULL,

table\_name VARCHAR(50) NOT NULL,

operation\_type ENUM('INSERT', 'UPDATE', 'DELETE') NOT NULL,

record\_id INT NOT NULL,

old\_values JSON,

new\_values JSON,

user\_id INT,

timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

status ENUM('COMMITTED', 'ROLLED\_BACK', 'PENDING') DEFAULT 'PENDING'

);

-- Table: Version Control

CREATE TABLE IF NOT EXISTS version\_control (

id INT AUTO\_INCREMENT PRIMARY KEY,

table\_name VARCHAR(50) NOT NULL,

record\_id INT NOT NULL,

version\_number INT NOT NULL DEFAULT 1,

last\_modified TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

modified\_by INT,

UNIQUE KEY (table\_name, record\_id)

);

-- Table: Lock Management

CREATE TABLE IF NOT EXISTS lock\_management (

id INT AUTO\_INCREMENT PRIMARY KEY,

table\_name VARCHAR(50) NOT NULL,

record\_id INT NOT NULL,

lock\_type ENUM('SHARED', 'EXCLUSIVE') NOT NULL,

lock\_holder INT NOT NULL,

lock\_timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

lock\_timeout TIMESTAMP,

UNIQUE KEY (table\_name, record\_id)

);

-- Table: Users (for authentication)

CREATE TABLE IF NOT EXISTS users (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

password\_hash VARCHAR(255) NOT NULL,

date\_of\_birth DATE,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

last\_login TIMESTAMP NULL,

version INT DEFAULT 1

);

-- Table: User Medical History

CREATE TABLE IF NOT EXISTS medical\_history (

id INT AUTO\_INCREMENT PRIMARY KEY,

user\_id INT NOT NULL,

family\_history\_of\_cancer ENUM('yes', 'no', 'unknown') DEFAULT 'unknown',

years\_smoking INT DEFAULT 0,

packs\_per\_day DECIMAL(3,1) DEFAULT 0.0,

previous\_lung\_diseases TEXT,

occupational\_exposure ENUM('yes', 'no', 'unknown') DEFAULT 'unknown',

occupational\_details TEXT,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

version INT DEFAULT 1,

FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE CASCADE

);

-- Table: Symptoms (stores detailed symptom information)

CREATE TABLE IF NOT EXISTS symptoms (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

description TEXT,

severity\_scale INT DEFAULT 3 COMMENT 'Scale from 1-5, with 5 being most severe',

related\_to\_lung\_cancer BOOLEAN DEFAULT TRUE

);

-- Table: Predictions (stores user symptoms and prediction results)

CREATE TABLE IF NOT EXISTS predictions (

id INT AUTO\_INCREMENT PRIMARY KEY,

age INT CHECK (age BETWEEN 0 AND 120),

gender ENUM('Male', 'Female', 'Other') NOT NULL,

smoking ENUM('yes', 'no') NOT NULL,

cough ENUM('yes', 'no') NOT NULL,

chest\_pain ENUM('yes', 'no') NOT NULL,

fatigue ENUM('yes', 'no') NOT NULL,

shortness\_of\_breath ENUM('yes', 'no') NOT NULL,

prediction VARCHAR(255) NOT NULL,

risk\_score DECIMAL(5,2) DEFAULT 0.0,

prediction\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

version INT DEFAULT 1

);

-- Table: User Predictions (links users to their predictions)

CREATE TABLE IF NOT EXISTS user\_predictions (

id INT AUTO\_INCREMENT PRIMARY KEY,

user\_id INT,

prediction\_id INT,

notes TEXT,

FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE CASCADE,

FOREIGN KEY (prediction\_id) REFERENCES predictions(id) ON DELETE CASCADE

);

-- Table: Medical Recommendations

CREATE TABLE IF NOT EXISTS recommendations (

id INT AUTO\_INCREMENT PRIMARY KEY,

risk\_level ENUM('Low', 'Moderate', 'High') NOT NULL,

recommendation\_text TEXT NOT NULL,

resource\_links TEXT

);

-- Table: User Feedback (for system improvement)

CREATE TABLE IF NOT EXISTS user\_feedback (

id INT AUTO\_INCREMENT PRIMARY KEY,

user\_id INT,

prediction\_id INT,

feedback\_text TEXT NOT NULL,

rating INT CHECK (rating BETWEEN 1 AND 5),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE SET NULL,

FOREIGN KEY (prediction\_id) REFERENCES predictions(id) ON DELETE SET NULL

);

-- Table: Deleted Predictions Log

CREATE TABLE IF NOT EXISTS deleted\_predictions\_log (

id INT AUTO\_INCREMENT PRIMARY KEY,

deleted\_prediction\_id INT,

deleted\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

# CHAPTER-3 QUERIES

Table: Transaction Log

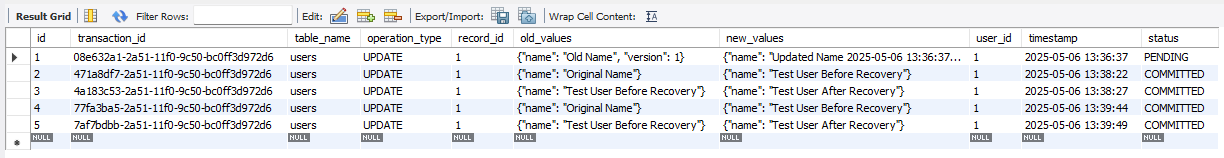


Table: Version Control

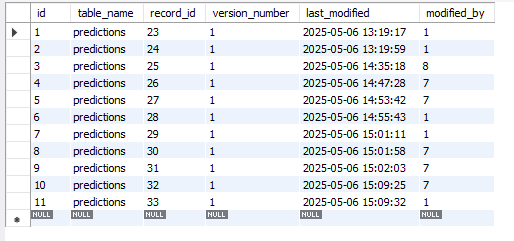


Table: Users

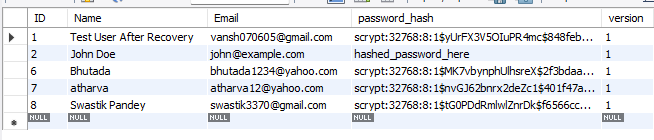


Table: medical\_history

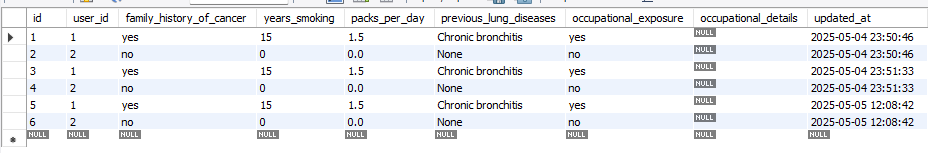


Table: Symptoms

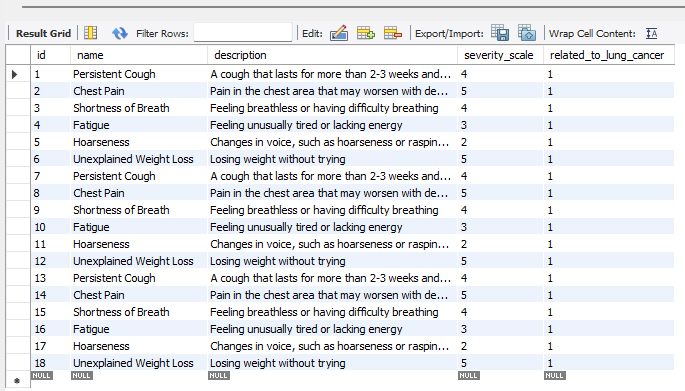
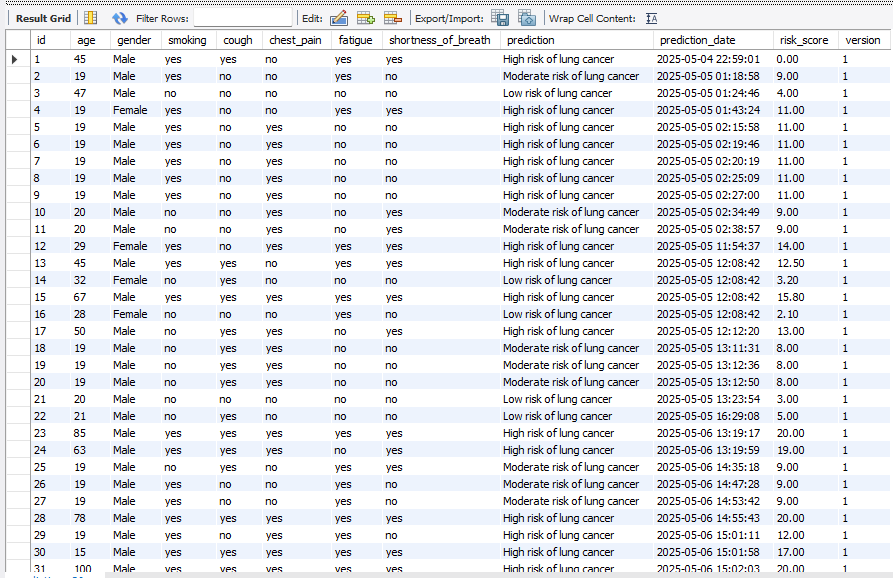


Table: predictions



# CHAPTER -4 NORMALIZATION

# Normalization is the process to eliminate data redundancy and enhance data integrity in the table. Normalization also helps to organize the data in the database. It is a multi-step process that sets the data into tabular form and removes the duplicated data from the relational tables.

# TYPES OF NORMAL FORMS IN NORMALIZATION-

# 1. FIRST NORMAL FORM

# 2. SECOND NORMAL FORM

# 3. THIRD NORMAL FORM

# 4. BOYCE-CODD NORMAL FORM

# 5. FOURTH NORMAL FORM

# 6. FIFTH NORMAL FORM

# MEDICAL\_HISTORY: 3NF

# Functional Dependency

# User\_ID → Family\_History, Smoking\_History,Previous\_Diseases, Occupational\_Exposure

# History\_ID →User\_ID,Family\_History,Smoking\_History, Previous\_Diseases, Occupational\_Exposure

# User\_ID → History\_ID (transitive dependency)

# It is currently in 2NF because it has a transitive dependency of User\_ID → History\_ID. To convert it into 3NF, we need to remove the transitive dependency.

# TO CHANGE:

# 1. CREATE TABLE MEDICAL\_HISTORY1 (

# History\_ID VARCHAR2(20) PRIMARY KEY,

# User\_ID VARCHAR2(20),

# Family\_History VARCHAR2(100),

# Smoking\_History VARCHAR2(100),

# Previous\_Diseases VARCHAR2(100),

# Occupational\_Exposure VARCHAR2(100)

# )

# 2. CREATE TABLE MEDICAL\_HISTORY2 (

# User\_ID VARCHAR2(20) PRIMARY KEY,

# History\_ID VARCHAR2(20)

# )

# PREDICTIONS:

# NO NF

# Functional Dependency

# Prediction\_ID → User\_ID, Risk\_Score, Prediction\_Date, Symptoms\_List, Recommendations

# Normalization

# There is atomicity in this table so we need to solve the multivalued attribute of Symptoms\_List

# TO CHANGE:

# CREATE TABLE PREDICTIONS1 (

# Prediction\_ID VARCHAR2(20),

# User\_ID VARCHAR2(20),

# Risk\_Score NUMBER,

# Prediction\_Date DATE,

# Symptom1 VARCHAR2(50),

# Symptom2 VARCHAR2(50),

# Symptom3 VARCHAR2(50),

# Recommendations VARCHAR2(200)

# )

# 2NF

# Functional Dependency

# Prediction\_ID → User\_ID, Risk\_Score, Prediction\_Date, Symptom1, Symptom2, Symptom3, Recommendations

# User\_ID → Risk\_Score, Prediction\_Date, Symptom1, Symptom2, Symptom3, Recommendations

# Normalization

# There is no Partial Primary key dependency, so this satisfies the condition for 2NF

# 3NF

# Functional Dependency

# Prediction\_ID → User\_ID, Risk\_Score, Prediction\_Date, Symptom1, Symptom2, Symptom3, Recommendations

# User\_ID → Risk\_Score, Prediction\_Date, Symptom1, Symptom2, Symptom3, Recommendations

# Normalization

# There is no Transitive Dependency and every candidate key is a super key, so it satisfies Boyce-Codd and 4NF

# SYMPTOMS:

# 5NF

# Functional Dependency

# Symptom\_ID → Symptom\_Name, Severity\_Level, Description, Category

# Category → Symptom\_Name, Severity\_Level, Description

# Normalization

# It's already existing in 3NF form, so no need to make any changes in the database.

# RECOMMENDATIONS:

# 5NF

# Functional Dependency

# Recommendation\_ID → Risk\_Level, Medical\_Advice, Prevention\_Tips, Follow\_up\_Steps

# Risk\_Level → Medical\_Advice, Prevention\_Tips, Follow\_up\_Steps

# Normalization

# It's already existing in 3NF form, so no need to make any changes in the database.

# USER\_FEEDBACK:

# 5NF

# Functional Dependency

# Feedback\_ID → User\_ID, Prediction\_ID, Rating, Comments, Feedback\_Date

# User\_ID → Prediction\_ID, Rating, Comments, Feedback\_Date

# Normalization

# It's already existing in 3NF form, so no need to make any changes in the database.

# PITFALLS IN NORMALIZATION CONCEPT

While the provided data appears to be structured and normalized up to at least Second Normal Form (2NF), there are still potential pitfalls and areas for improvement in terms of database design and normalization concepts. Here are some pitfalls and considerations:

### Redundancy in Address and Contact Information:

- In several tables (e.g., User, Customer, ServiceProvider, Employee), there are columns for storing Address and ContactNumber. This can lead to redundancy if the same address or contact number needs to be updated in multiple places. One solution could be to create separate tables for Address and Contact information and link them using foreign keys.

### Denormalization for Performance:

While normalization helps in reducing redundancy and maintaining data integrity, in some cases, denormalization might be necessary for performance optimization. For example, in a high-transaction system, joining multiple tables frequently could impact performance. In such cases, carefully denormalizing certain tables or using materialized views can be considered.

### Potential Update Anomalies:

* + Update anomalies can occur when data needs to be updated in multiple places, leading to inconsistencies if not handled properly. For instance, if a customer's contact number changes, it needs to be updated in multiple tables (e.g., Customer, User) where it's stored, increasing the risk of inconsistencies.

### Lack of Data Validation:

* + Data validation is crucial to ensure data integrity. Without proper validation rules and constraints, the database may accept invalid or inconsistent data, leading to issues in data quality. Implementing data validation checks at the database level can mitigate this risk.

### Overly Nested Relationships:

* + While relationships between tables are necessary, overly nested relationships can make queries complex and impact performance. It's essential to strike a balance between maintaining relationships for data integrity and optimizing query performance.

### Incomplete Normalization:

* + Although the provided data seems to be normalized up to 2NF, further normalization (e.g., Third Normal Form - 3NF) could be beneficial in some cases. Analyzing functional dependencies and eliminating transitive dependencies can lead to a more robust and efficient database design.

### Handling Historical Data:

* + If historical data tracking is required (e.g., tracking changes in meter readings over time), additional considerations for data storage and retrieval mechanisms may be needed. Implementing effective techniques such as versioning or audit trails can address this requirement without compromising normalization.

### Optimizing Indexing and Query Performance:

* + While normalization focuses on data organization, indexing and optimizing queries are essential for efficient data retrieval. Proper indexing strategies, query optimization techniques,and understanding the database engine's capabilities are crucial for improving overall system performance.

Addressing these pitfalls involves a combination of thoughtful database design, adherence to normalization principles, implementing data validation rules, optimizing performance, and considering specific business requirements for data storage and retrieval.

**CHAPTER-5**

## Implementation of concurrency control and recovery mechanisms

## 1. Concurrency Control Mechanisms:

## a) Lock Management:

## - Implements a global locking mechanism to ensure only one user can make a prediction at a time

## - Prevents race conditions during risk assessment calculations

## - Ensures data consistency in medical records

## - Lock types:

## \* Shared locks for reading patient data

## \* Exclusive locks for updating medical histories

## \* Prediction locks for risk assessment calculations

## b) Version Control:

## - Maintains version history of medical records

## - Tracks changes to risk assessments

## - Prevents lost updates in patient data

## - Supports optimistic concurrency control

## c) Transaction Management:

## - Ensures atomicity of medical data updates

## - Maintains consistency in risk assessment calculations

## - Isolates concurrent user sessions

## - Provides durability for medical records

## 2. Recovery Mechanisms:

## a) Transaction Log:

## - Records all changes to medical data

## - Tracks risk assessment calculations

## - Maintains audit trail of patient interactions

## - Enables point-in-time recovery

## b) Backup Procedures:

## - Daily full backups of medical database

## - Incremental backups every 6 hours

## - Backup of risk assessment models

## - Secure storage of patient data

## c) Recovery Procedures:

## - Point-in-time recovery for medical records

## - Crash recovery for interrupted predictions

## - System state recovery

## - Data consistency checks

## 3. Implementation Details:

## a) Lock Implementation:

## b) Lock Acquisition Procedures:

## 

## c) Lock Release Procedures:

## 

#### TRANSACTION FAILURE

#### TRANSACTION FAILURE HANDLING

#### 1. Types of Transaction Failures:

#### a) System Failures:

#### - Database crashes during risk assessment

#### - Server power failures

#### - Network interruptions

#### - Memory overflow during prediction calculations

#### b) Transaction Failures:

#### - Deadlocks during concurrent predictions

#### - Timeout during medical history updates

#### - Validation errors in risk assessment data

#### - Constraint violations in patient records

#### 2. Recovery Mechanisms:

#### ```sql

#### -- Transaction Log Table

#### CREATE TABLE transaction\_log (

#### id INT AUTO\_INCREMENT PRIMARY KEY,

#### transaction\_id VARCHAR(36) NOT NULL,

#### table\_name VARCHAR(50) NOT NULL,

#### operation\_type ENUM('INSERT', 'UPDATE', 'DELETE') NOT NULL,

#### record\_id INT NOT NULL,

#### old\_values JSON,

#### new\_values JSON,

#### user\_id INT,

#### timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

#### status ENUM('COMMITTED', 'ROLLED\_BACK', 'PENDING') DEFAULT 'PENDING'

#### );

#### -- Recovery Procedure

#### DELIMITER $$

#### CREATE PROCEDURE handle\_transaction\_failure(

#### IN p\_transaction\_id VARCHAR(36)

#### )

#### BEGIN

#### -- Log the failure

#### INSERT INTO transaction\_log

#### (transaction\_id, table\_name, operation\_type, record\_id, status)

#### VALUES (p\_transaction\_id, 'predictions', 'FAILED', 0, 'ROLLED\_BACK');

#### 

#### -- Rollback any pending changes

#### ROLLBACK;

#### 

#### -- Release any held locks

#### DELETE FROM lock\_management

#### WHERE lock\_holder = (SELECT user\_id FROM transaction\_log

#### WHERE transaction\_id = p\_transaction\_id);

#### END $$

#### DELIMITER ;

#### ```

#### 3. Error Handling:

#### ```sql

#### -- Error Log Table

#### CREATE TABLE error\_log (

#### id INT AUTO\_INCREMENT PRIMARY KEY,

#### error\_type VARCHAR(50) NOT NULL,

#### error\_message TEXT,

#### transaction\_id VARCHAR(36),

#### timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

#### severity ENUM('LOW', 'MEDIUM', 'HIGH') DEFAULT 'MEDIUM'

#### );

#### -- Error Handling Procedure

#### DELIMITER $$

#### CREATE PROCEDURE log\_transaction\_error(

#### IN p\_error\_type VARCHAR(50),

#### IN p\_error\_message TEXT,

#### IN p\_transaction\_id VARCHAR(36),

#### IN p\_severity ENUM('LOW', 'MEDIUM', 'HIGH')

#### )

#### BEGIN

#### -- Log the error

#### INSERT INTO error\_log

#### (error\_type, error\_message, transaction\_id, severity)

#### VALUES (p\_error\_type, p\_error\_message, p\_transaction\_id, p\_severity);

#### 

#### -- Handle based on severity

#### CASE p\_severity

#### WHEN 'HIGH' THEN

#### -- Notify administrators

#### CALL notify\_admin(p\_error\_message);

#### -- Rollback transaction

#### CALL handle\_transaction\_failure(p\_transaction\_id);

#### WHEN 'MEDIUM' THEN

#### -- Log and continue

#### UPDATE transaction\_log

#### SET status = 'PENDING'

#### WHERE transaction\_id = p\_transaction\_id;

#### WHEN 'LOW' THEN

#### -- Just log the error

#### UPDATE transaction\_log

#### SET status = 'COMMITTED'

#### WHERE transaction\_id = p\_transaction\_id;

#### END CASE;

#### END $$

#### DELIMITER ;

#### ```

#### 4. Recovery Steps:

#### a) For System Failures:

#### - Check transaction log for incomplete transactions

#### - Restore database from last backup

#### - Replay committed transactions

#### - Verify data consistency

#### b) For Transaction Failures:

#### - Identify failed transaction

#### - Rollback incomplete changes

#### - Release held locks

#### - Log failure details

#### - Notify affected users

#### 5. Prevention Measures:

#### a) Before Transaction:

#### - Validate input data

#### - Check system resources

#### - Verify user permissions

#### - Ensure data consistency

#### b) During Transaction:

#### - Monitor transaction progress

#### - Track resource usage

#### - Maintain lock timeouts

#### - Log all operations

#### c) After Transaction:

#### - Verify data integrity

#### - Update transaction status

#### - Release resources

#### - Log completion

#### The system maintains data integrity and provides clear feedback to users when transaction failures occur.

#### DEADLOCK PREVENTION

#### 

**SET innodb\_lock\_wait\_timeout = 10;: This line sets the timeout period for InnoDB lock waits to 10 seconds. InnoDB is a storage engine for MySQL that provides transaction support.**

**START TRANSACTION;: Begins a new transaction.**

**SELECT \* FROM medical\_history WHERE user\_id = 'U123' FOR UPDATE;: This line selects data from the medical\_history table for the user with ID 'U123' and locks the selected rows with a "FOR UPDATE" clause. This locking ensures that other transactions cannot modify these rows until the current transaction is completed.**

**INSERT INTO transaction\_log;: This line inserts a record into the transaction\_log table, recording information about an update operation (in this case, updating the medical history with new smoking history and lung disease information).**

**COMMIT;: Commits the transaction, making all changes permanent if all statements within the transaction execute successfully.**

### Recovery Mechanism:

****

1. Transaction Logging:

- Maintains a detailed record of all database changes

- Records medical history updates, risk assessments, and patient data modifications

- Stores both old and new values for each change

- Enables point-in-time recovery of medical records

2. Backup Procedures:

- Daily full backups of the medical database

- Incremental backups every 6 hours

- Secure storage of patient data and risk assessment models

- Backup of critical medical history and prediction data

3. Point-in-Time Recovery:

- Allows restoration of medical data to any specific point in time

- Useful for recovering from data corruption or incorrect updates

- Maintains data consistency across all medical records

- Preserves the integrity of risk assessment history

4. Crash Recovery:

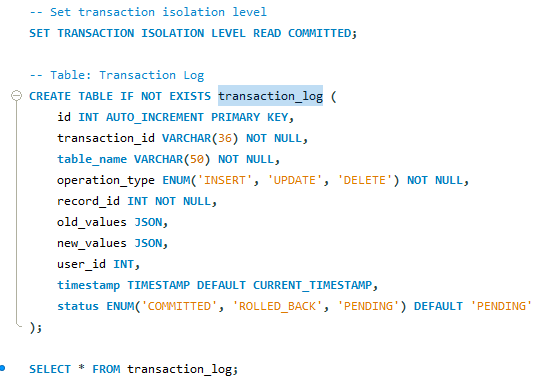
- Automatically recovers from system failures

- Restores medical data to a consistent state

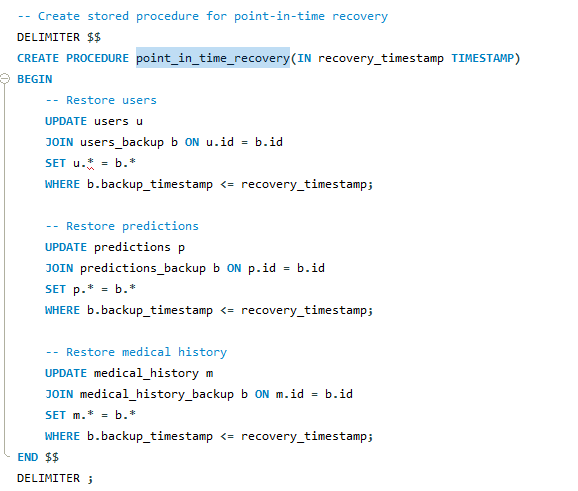
- Handles interrupted risk assessment calculations

- Maintains transaction atomicity

5. Implementation Example:



-- Point-in-Time Recovery Procedure



6. Recovery Steps:

a) System Failure Recovery:

- Check transaction log for incomplete transactions

- Restore database from last backup

- Replay committed transactions

- Verify medical data consistency

b) Data Corruption Recovery:

- Identify corrupted medical records

- Restore from backup

- Verify data integrity

- Update transaction logs

c) User Error Recovery:

- Identify incorrect updates

- Restore affected records

- Maintain audit trail

- Notify affected users

7. Security Measures:

- Encrypt backup files

- Secure transaction logs

- Implement access controls

- Maintain audit trails

8. Performance Considerations:

- Optimize backup procedures

- Manage storage efficiently

- Implement parallel recovery

- Monitor recovery times

This implementation ensures:

1. Data consistency in medical records

2. Reliable risk assessment history

3. Secure patient data management

4. Efficient recovery procedures

5. Comprehensive audit trailing

6. System availability

7. Data integrity protection

8. Patient safety

The system maintains:

- Medical data accuracy

- Risk assessment reliability

- Patient record integrity

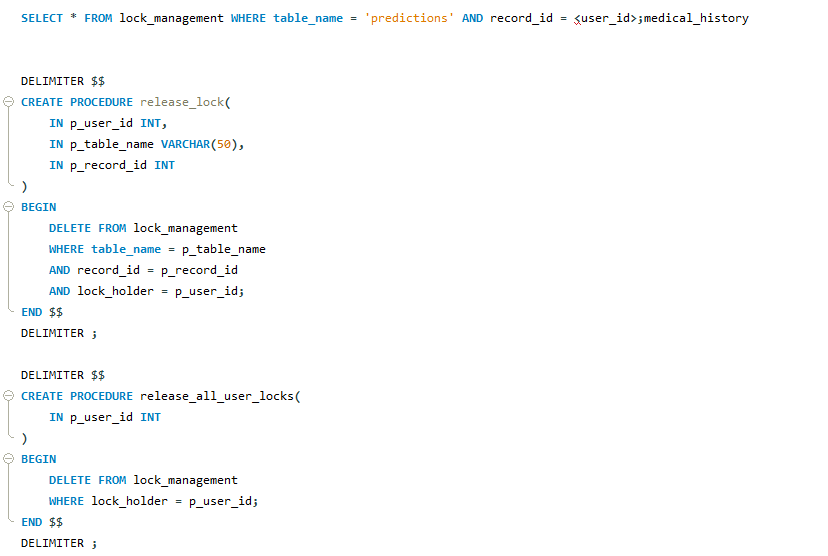
- System stability

- Data security

- Recovery efficiency

- Audit compliance

- Healthcare standardsstatements will be treated as part of a single unit of work. Transactions ensure data consistency by allowing a series of operations to be either committed (saved permanently) or rolled back (reverted) as a whole.



**CHAPTER-6 CODE FOR THE PROJECT**

## SQL CODE: (lung\_cancer\_db.sql)

*-- Create the database*

CREATE DATABASE IF NOT EXISTS lung\_cancer\_db;

USE lung\_cancer\_db;

*-- Set transaction isolation level*

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

*-- Table: Transaction Log*

CREATE TABLE IF NOT EXISTS transaction\_log (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    transaction\_id VARCHAR(36) NOT NULL,

    table\_name VARCHAR(50) NOT NULL,

    operation\_type ENUM('INSERT', 'UPDATE', 'DELETE') NOT NULL,

    record\_id INT NOT NULL,

    old\_values JSON,

    new\_values JSON,

    user\_id INT,

    timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

    status ENUM('COMMITTED', 'ROLLED\_BACK', 'PENDING') DEFAULT 'PENDING'

);

*-- Table: Version Control*

CREATE TABLE IF NOT EXISTS version\_control (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    table\_name VARCHAR(50) NOT NULL,

    record\_id INT NOT NULL,

    version\_number INT NOT NULL DEFAULT 1,

    last\_modified TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

    modified\_by INT,

    UNIQUE KEY (table\_name, record\_id)

);

*-- Table: Lock Management*

CREATE TABLE IF NOT EXISTS lock\_management (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    table\_name VARCHAR(50) NOT NULL,

    record\_id INT NOT NULL,

    lock\_type ENUM('SHARED', 'EXCLUSIVE') NOT NULL,

    lock\_holder INT NOT NULL,

    lock\_timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

    lock\_timeout TIMESTAMP,

    UNIQUE KEY (table\_name, record\_id)

);

*-- Table: Users (for authentication)*

CREATE TABLE IF NOT EXISTS users (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    name VARCHAR(100) NOT NULL,

    email VARCHAR(100) UNIQUE NOT NULL,

    password\_hash VARCHAR(255) NOT NULL,

    date\_of\_birth DATE,

    created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

    last\_login TIMESTAMP NULL,

    version INT DEFAULT 1

);

*-- Table: User Medical History*

CREATE TABLE IF NOT EXISTS medical\_history (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    user\_id INT NOT NULL,

    family\_history\_of\_cancer ENUM('yes', 'no', 'unknown') DEFAULT 'unknown',

    years\_smoking INT DEFAULT 0,

    packs\_per\_day DECIMAL(3,1) DEFAULT 0.0,

    previous\_lung\_diseases TEXT,

    occupational\_exposure ENUM('yes', 'no', 'unknown') DEFAULT 'unknown',

    occupational\_details TEXT,

    updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

    version INT DEFAULT 1,

    FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE CASCADE

);

*-- Table: Symptoms (stores detailed symptom information)*

CREATE TABLE IF NOT EXISTS symptoms (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    name VARCHAR(100) NOT NULL,

    description TEXT,

    severity\_scale INT DEFAULT 3 COMMENT 'Scale from 1-5, with 5 being most severe',

    related\_to\_lung\_cancer BOOLEAN DEFAULT TRUE

);

*-- Table: Predictions (stores user symptoms and prediction results)*

CREATE TABLE IF NOT EXISTS predictions (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    age INT CHECK (age BETWEEN 0 AND 120),

    gender ENUM('Male', 'Female', 'Other') NOT NULL,

    smoking ENUM('yes', 'no') NOT NULL,

    cough ENUM('yes', 'no') NOT NULL,

    chest\_pain ENUM('yes', 'no') NOT NULL,

    fatigue ENUM('yes', 'no') NOT NULL,

    shortness\_of\_breath ENUM('yes', 'no') NOT NULL,

    prediction VARCHAR(255) NOT NULL,

    risk\_score DECIMAL(5,2) DEFAULT 0.0,

    prediction\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

    version INT DEFAULT 1

);

*-- Table: User Predictions (links users to their predictions)*

CREATE TABLE IF NOT EXISTS user\_predictions (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    user\_id INT,

    prediction\_id INT,

    notes TEXT,

    FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE CASCADE,

    FOREIGN KEY (prediction\_id) REFERENCES predictions(id) ON DELETE CASCADE

);

*-- Table: Medical Recommendations*

CREATE TABLE IF NOT EXISTS recommendations (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    risk\_level ENUM('Low', 'Moderate', 'High') NOT NULL,

    recommendation\_text TEXT NOT NULL,

    resource\_links TEXT

);

*-- Table: User Feedback (for system improvement)*

CREATE TABLE IF NOT EXISTS user\_feedback (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    user\_id INT,

    prediction\_id INT,

    feedback\_text TEXT NOT NULL,

    rating INT CHECK (rating BETWEEN 1 AND 5),

    created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

    FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE SET NULL,

    FOREIGN KEY (prediction\_id) REFERENCES predictions(id) ON DELETE SET NULL

);

*-- Table: Deleted Predictions Log*

CREATE TABLE IF NOT EXISTS deleted\_predictions\_log (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    deleted\_prediction\_id INT,

    deleted\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

*-- Create backup table for critical data*

CREATE TABLE IF NOT EXISTS users\_backup LIKE users;

CREATE TABLE IF NOT EXISTS predictions\_backup LIKE predictions;

CREATE TABLE IF NOT EXISTS medical\_history\_backup LIKE medical\_history;

*-- Create stored procedure for backup*

DELIMITER $$

CREATE PROCEDURE create\_backup()

BEGIN

    DECLARE backup\_timestamp TIMESTAMP;

    SET backup\_timestamp = CURRENT\_TIMESTAMP;

*-- Backup users*

    INSERT INTO users\_backup

    SELECT \*, backup\_timestamp FROM users;

*-- Backup predictions*

    INSERT INTO predictions\_backup

    SELECT \*, backup\_timestamp FROM predictions;

*-- Backup medical history*

    INSERT INTO medical\_history\_backup

    SELECT \*, backup\_timestamp FROM medical\_history;

END $$

DELIMITER ;

*-- Create stored procedure for point-in-time recovery*

DELIMITER $$

CREATE PROCEDURE point\_in\_time\_recovery(IN recovery\_timestamp TIMESTAMP)

BEGIN

*-- Restore users*

    UPDATE users u

    JOIN users\_backup b ON u.id = b.id

    SET u.\* = b.\*

    WHERE b.backup\_timestamp <= recovery\_timestamp;

*-- Restore predictions*

    UPDATE predictions p

    JOIN predictions\_backup b ON p.id = b.id

    SET p.\* = b.\*

    WHERE b.backup\_timestamp <= recovery\_timestamp;

*-- Restore medical history*

    UPDATE medical\_history m

    JOIN medical\_history\_backup b ON m.id = b.id

    SET m.\* = b.\*

    WHERE b.backup\_timestamp <= recovery\_timestamp;

END $$

DELIMITER ;

*-- Create trigger for transaction logging*

DELIMITER $$

CREATE TRIGGER log\_user\_changes

AFTER UPDATE ON users

FOR EACH ROW

BEGIN

    INSERT INTO transaction\_log (transaction\_id, table\_name, operation\_type, record\_id, old\_values, new\_values)

    VALUES (

        UUID(),

        'users',

        'UPDATE',

        NEW.id,

        JSON\_OBJECT(

            'name', OLD.name,

            'email', OLD.email,

            'version', OLD.version

        ),

        JSON\_OBJECT(

            'name', NEW.name,

            'email', NEW.email,

            'version', NEW.version

        )

    );

END $$

DELIMITER ;

*-- Create stored procedure for deadlock detection*

DELIMITER $$

CREATE PROCEDURE detect\_deadlocks()

BEGIN

    DECLARE done INT DEFAULT 0;

    DECLARE lock\_id INT;

    DECLARE cur CURSOR FOR

        SELECT id FROM lock\_management

        WHERE lock\_timeout < CURRENT\_TIMESTAMP;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN cur;

    read\_loop: LOOP

        FETCH cur INTO lock\_id;

        IF done THEN

            LEAVE read\_loop;

        END IF;

*-- Release expired locks*

        DELETE FROM lock\_management WHERE id = lock\_id;

    END LOOP;

    CLOSE cur;

END $$

DELIMITER ;

*-- Create event scheduler for regular maintenance*

CREATE EVENT IF NOT EXISTS maintenance\_schedule

ON SCHEDULE EVERY 1 DAY

DO

BEGIN

*-- Create daily backup*

    CALL create\_backup();

*-- Clean up old transaction logs (keep last 30 days)*

    DELETE FROM transaction\_log

    WHERE timestamp < DATE\_SUB(CURRENT\_TIMESTAMP, INTERVAL 30 DAY);

*-- Check for deadlocks*

    CALL detect\_deadlocks();

END;

*-- Enable event scheduler*

SET GLOBAL event\_scheduler = ON;

SELECT \* from users;

SELECT \* FROM user\_predictions;

SHOW TABLES;

DROP TABLE IF EXISTS user\_prediction;

DROP TABLE IF EXISTS prediction;

DROP TABLE IF EXISTS deleted\_predictions;

CREATE OR REPLACE VIEW user\_prediction\_history AS

SELECT

    u.id AS user\_id,

    u.name AS user\_name,

    u.email AS user\_email,

    p.id AS prediction\_id,

    p.age, p.gender, p.smoking, p.cough, p.chest\_pain, p.fatigue, p.shortness\_of\_breath,

    p.prediction, p.risk\_score, p.prediction\_date

FROM users u

JOIN user\_predictions up ON u.id = up.user\_id

JOIN predictions p ON up.prediction\_id = p.id

ORDER BY p.prediction\_date DESC;

DELIMITER $$

CREATE TRIGGER after\_prediction\_delete

AFTER DELETE ON predictions

FOR EACH ROW

BEGIN

    INSERT INTO deleted\_predictions\_log (deleted\_prediction\_id) VALUES (OLD.id);

END $$

DELIMITER ;

DELIMITER $$

CREATE PROCEDURE update\_all\_risk\_scores()

BEGIN

    DECLARE done INT DEFAULT 0;

    DECLARE pred\_id INT;

    DECLARE cur CURSOR FOR SELECT id FROM predictions;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN cur;

    read\_loop: LOOP

        FETCH cur INTO pred\_id;

        IF done THEN

            LEAVE read\_loop;

        END IF;

*-- Example: set risk\_score to 0 for demonstration*

        UPDATE predictions SET risk\_score = 0 WHERE id = pred\_id;

    END LOOP;

    CLOSE cur;

END $$

DELIMITER ;

SELECT \* FROM lock\_management;

SELECT \* FROM lock\_management WHERE table\_name = 'predictions' AND record\_id = <user\_id>;

## MODEL: (dummy\_model.py)

## import pandas as pd

## from sklearn.model\_selection import train\_test\_split

## from sklearn.preprocessing import LabelEncoder

## from sklearn.ensemble import RandomForestClassifier

## from sklearn.metrics import classification\_report, accuracy\_score

## # Step 1: Load the dataset

## data = pd.read\_csv('survey lung cancer.csv')

## # Step 2: Preprocess the data

## # Convert target column to binary (YES -> 1, NO -> 0)

## data['LUNG\_CANCER'] = data['LUNG\_CANCER'].map({'YES': 1, 'NO': 0})

## # Encode categorical features

## le = LabelEncoder()

## data['GENDER'] = le.fit\_transform(data['GENDER'])  # M -> 1, F -> 0

## # Step 3: Split features and target

## X = data.drop('LUNG\_CANCER', axis=1)

## y = data['LUNG\_CANCER']

## # Step 4: Train-test split

## X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

## # Step 5: Train a Random Forest model

## model = RandomForestClassifier(random\_state=42)

## model.fit(X\_train, y\_train)

## # Step 6: Evaluate

## y\_pred = model.predict(X\_test)

## print("Accuracy:", accuracy\_score(y\_test, y\_pred))

## print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

## # Step 7: Predict for new data (example)

## sample = pd.DataFrame([{

## 'GENDER': le.transform(['M'])[0],  # or 1

## 'AGE': 65,

## 'SMOKING': 2,

## 'YELLOW\_FINGERS': 2,

## 'ANXIETY': 1,

## 'PEER\_PRESSURE': 2,

## 'CHRONIC DISEASE': 2,

## 'FATIGUE ': 2,

## 'ALLERGY ': 1,

## 'WHEEZING': 2,

## 'ALCOHOL CONSUMING': 2,

## 'COUGHING': 2,

## 'SHORTNESS OF BREATH': 1,

## 'SWALLOWING DIFFICULTY': 2,

## 'CHEST PAIN': 1

## }])

## prediction = model.predict(sample)

## print("Predicted Lung Cancer Risk (1=Yes, 0=No):", prediction[0])

## import joblib

## joblib.dump(model, 'model/model.pkl')

## import numpy as np

## # Load the model

## loaded\_model = joblib.load('model/model.pkl')

## def predict\_lung\_cancer(data):

## # Ensure the order of features matches training data

## input\_array = np.array([[

## int(data['Gender']),  # 1 = M, 0 = F

## int(data['Age']),

## int(data['Smoking']),

## 1,  # YELLOW\_FINGERS (you can later collect from form)

## 1,  # ANXIETY

## 1,  # PEER\_PRESSURE

## 1,  # CHRONIC DISEASE

## int(data['Fatigue']),

## 1,  # ALLERGY

## 1,  # WHEEZING

## 1,  # ALCOHOL CONSUMING

## int(data['Cough']),

## int(data['Shortness\_of\_breath']),

## 1,  # SWALLOWING DIFFICULTY

## int(data['Chest\_Pain'])

## ]])

## prediction = loaded\_model.predict(input\_array)[0]

## return int(prediction)

## Frontend Code: (base.html)

## <!DOCTYPE html>

## <html lang="en">

## <head>

## <meta charset="UTF-8">

## <meta name="viewport" content="width=device-width, initial-scale=1.0">

## <title>{% block title %}Lung Cancer Risk App{% endblock %}</title>

## <link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

## </head>

## <body>

## <header>

## <nav class="navbar">

## <div class="brand">

## <span class="logo">Lung Cancer Detector</span>

## </div>

## <div class="nav-links">

## {% if session.get('user\_id') %}

## <a href="{{ url\_for('dashboard') }}" class="nav-link">Dashboard</a>

## <a href="{{ url\_for('predict') }}" class="nav-link">Predict</a>

## <a href="{{ url\_for('get\_user\_predictions') }}" class="nav-link">History</a>

## <a href="{{ url\_for('logout') }}" class="nav-link">Logout</a>

## {% else %}

## <a href="{{ url\_for('home') }}" class="nav-link">Home</a>

## <a href="{{ url\_for('login') }}" class="nav-link">Login</a>

## <a href="{{ url\_for('register') }}" class="nav-link">Register</a>

## {% endif %}

## </div>

## </nav>

## </header>

## <main>

## {% with messages = get\_flashed\_messages(with\_categories=true) %}

## {% if messages %}

## <div class="flash-messages">

## {% for category, message in messages %}

## <div class="alert alert-{{ category }}">{{ message }}</div>

## {% endfor %}

## </div>

## {% endif %}

## {% endwith %}

## 

## {% block content %}{% endblock %}

## </main>

## <footer>

## <div class="footer-content">

## <p>&copy; 2024 Lung Cancer Risk App | <a href="#" class="footer-link">Privacy Policy</a> | <a href="#" class="footer-link">Terms of Service</a></p>

## </div>

## </footer>

## </body>

## </html>

## 

## Dashboard.html

## <!-- templates/dashboard.html -->

## {% extends 'base.html' %}

## {% block title %}Dashboard{% endblock %}

## {% block content %}

## <h2>Welcome to Your Dashboard!</h2>

## <p>Hello, {{ name }}!</p>

## <div>

## <a href="{{ url\_for('predict') }}"><button>Predict Lung Cancer Risk</button></a>

## <a href="{{ url\_for('get\_user\_predictions') }}"><button>View Prediction History</button></a>

## </div>

## {% endblock %}

## History.html

## <!-- history.html -->

## {% extends 'base.html' %}

## {% block title %}Prediction History{% endblock %}

## {% block content %}

## <h2>Your Prediction History</h2>

## {% if predictions %}

## <table>

## <tr>

## <th>Date</th>

## <th>Age</th>

## <th>Gender</th>

## <th>Smoking</th>

## <th>Cough</th>

## <th>Chest Pain</th>

## <th>Fatigue</th>

## <th>Shortness of Breath</th>

## <th>Prediction</th>

## <th>Risk Score</th>

## </tr>

## {% for p in predictions %}

## <tr>

## <td>{{ p.prediction\_date }}</td>

## <td>{{ p.age }}</td>

## <td>{{ p.gender }}</td>

## <td>{{ p.smoking }}</td>

## <td>{{ p.cough }}</td>

## <td>{{ p.chest\_pain }}</td>

## <td>{{ p.fatigue }}</td>

## <td>{{ p.shortness\_of\_breath }}</td>

## <td>{{ p.prediction }}</td>

## <td>{{ p.risk\_score }}</td>

## </tr>

## {% endfor %}

## </table>

## {% else %}

## <p>No predictions found.</p>

## {% endif %}

## <a href="{{ url\_for('dashboard') }}">Back to Dashboard</a>

## {% endblock %}

## Index.html

## <!-- index.html -->

## <!DOCTYPE html>

## <html lang="en">

## <head>

## <meta charset="UTF-8">

## <title>Lung Cancer Detection</title>

## <link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

## </head>

## <body>

## <div class="container">

## <h1>Welcome to Lung Cancer Detection System</h1>

## <a href="/register">Register</a> | <a href="/login">Login</a>

## </div>

## </body>

## </html>

**CHAPTER-7 RESULT AND DISCUSSION**

The **Lung Cancer Detection System** is a crucial healthcare application designed to facilitate early diagnosis and risk assessment for lung cancer. The system integrates multiple modules to manage user data, collect health-related symptoms, analyze risk levels through machine learning predictions, and support clinical decision-making through comprehensive reporting and feedback mechanisms.

**USER\_DATA Table**: This table stores patient information such as age, gender, and smoking history. It acts as a core reference for evaluating risk factors and personalizing prediction results.

**SYMPTOMS Table**: This table captures symptom-specific details like persistent cough, chest pain, fatigue, and shortness of breath. These inputs are essential for the model to assess lung cancer risk accurately.

**PREDICTIONS Table**: This table records the results of the model’s analysis, including the calculated risk score and its classification. It supports version tracking and helps in generating health reports for users and doctors.

**RECOMMENDATIONS Table**: Based on prediction outcomes, this table provides tailored medical recommendations, lifestyle changes, or further diagnostic steps. It aids in enhancing user awareness and encouraging preventive measures.

**FEEDBACK Table**: This component gathers feedback from users regarding prediction accuracy, experience with the platform, and suggestions for improvement. It contributes to system refinement and better user engagement.

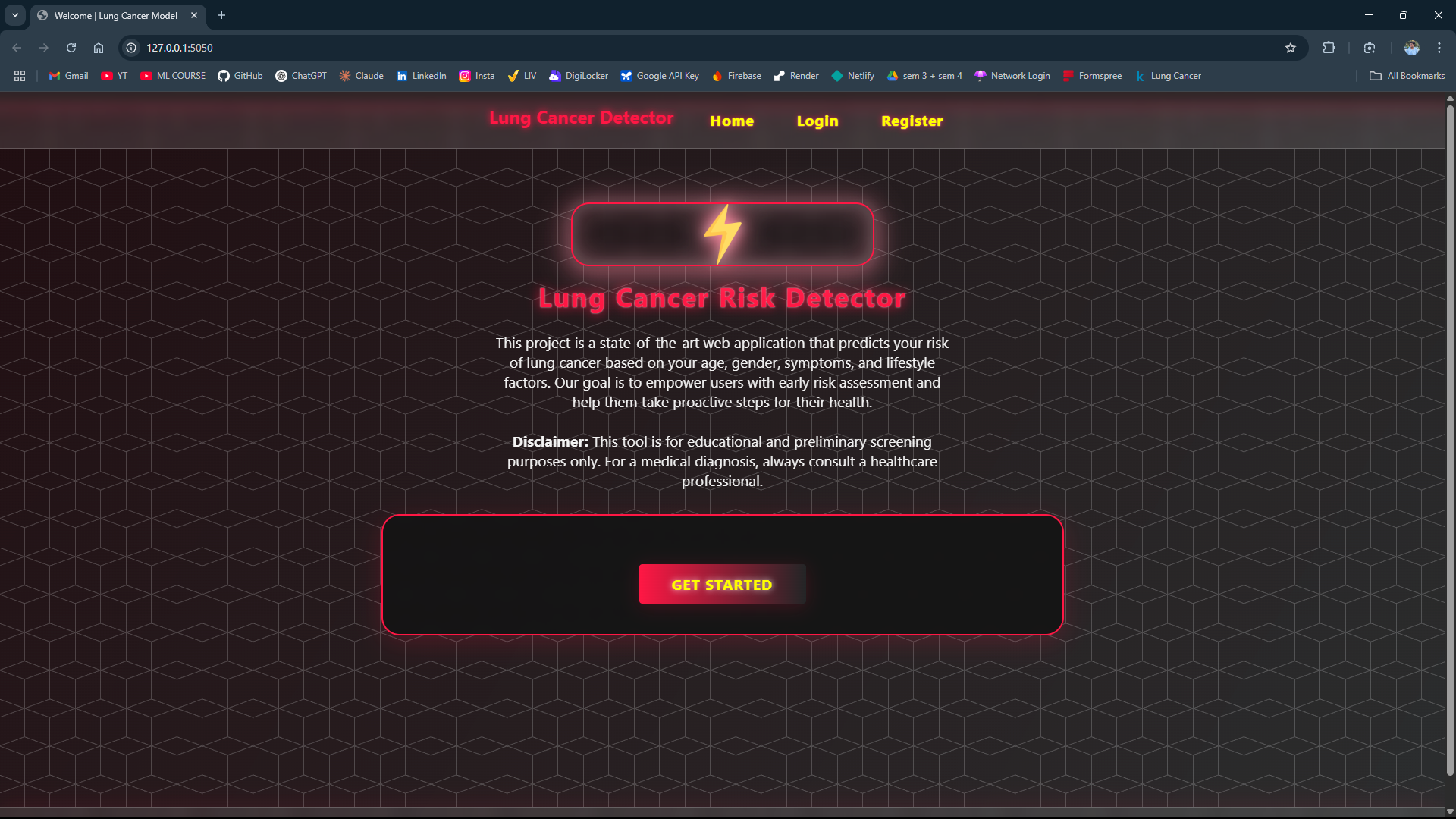
**MEDICAL\_HISTORY Table**: This table archives previous diseases or relevant medical records of users. It enhances model accuracy by considering pre-existing conditions in the analysis process.

**BACKUP\_LOGS and DELETED\_PREDICTIONS Tables**: These tables ensure secure data management by enabling **point-in-time recovery**, **soft deletes**, and **audit tracking**, thereby supporting database integrity, reliability, and recovery after unintended loss or failure.

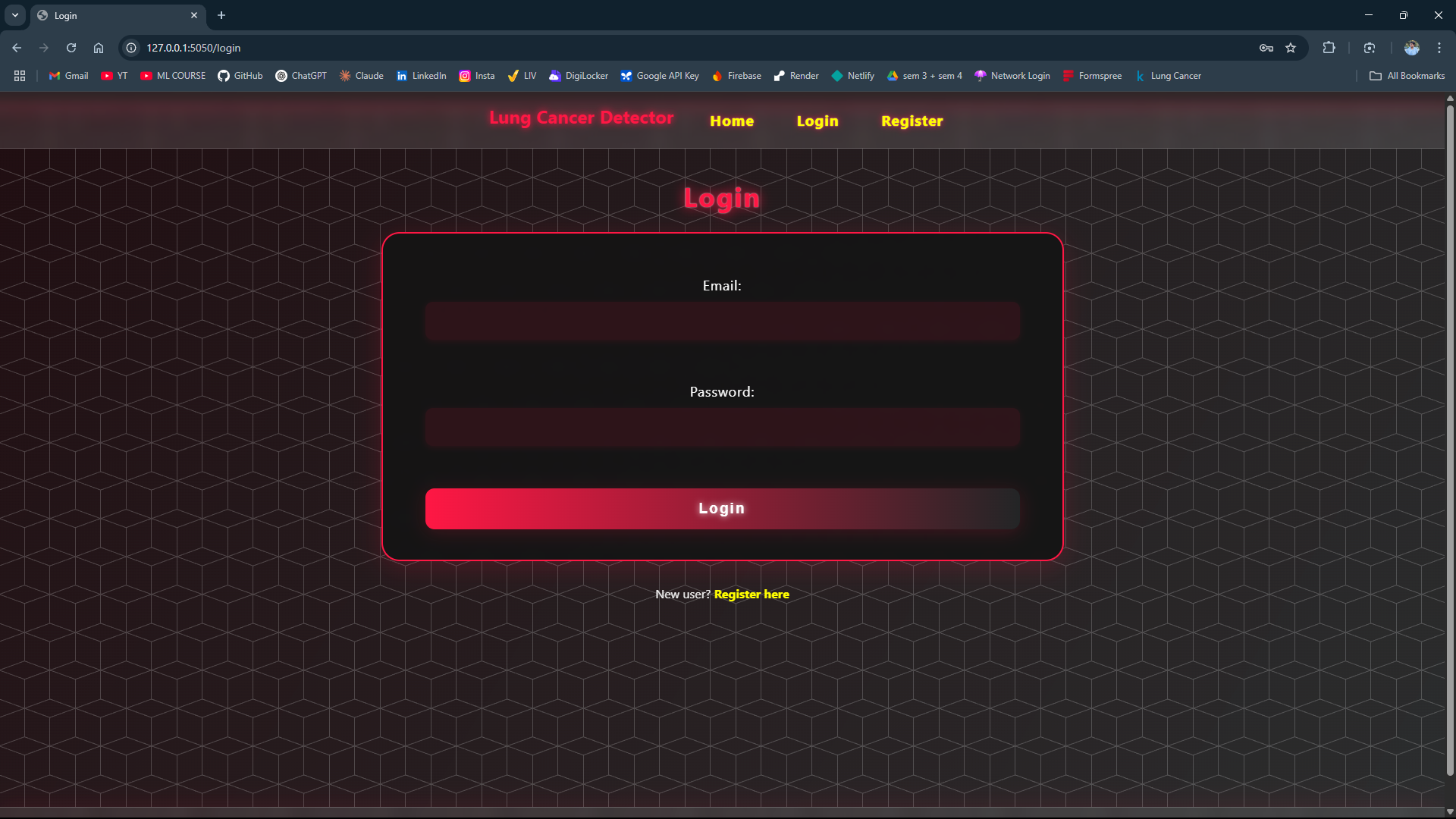
**EVENTS and TRANSACTIONS Modules**: These manage automated tasks like daily backups and transactional consistency, reducing human intervention and improving system reliability.

In summary, the Lung Cancer Detection System offers an intelligent, data-driven approach to early disease identification. It enhances patient awareness, supports doctors in decision-making, and maintains a highly secure, scalable database system for health analytics and reporting.

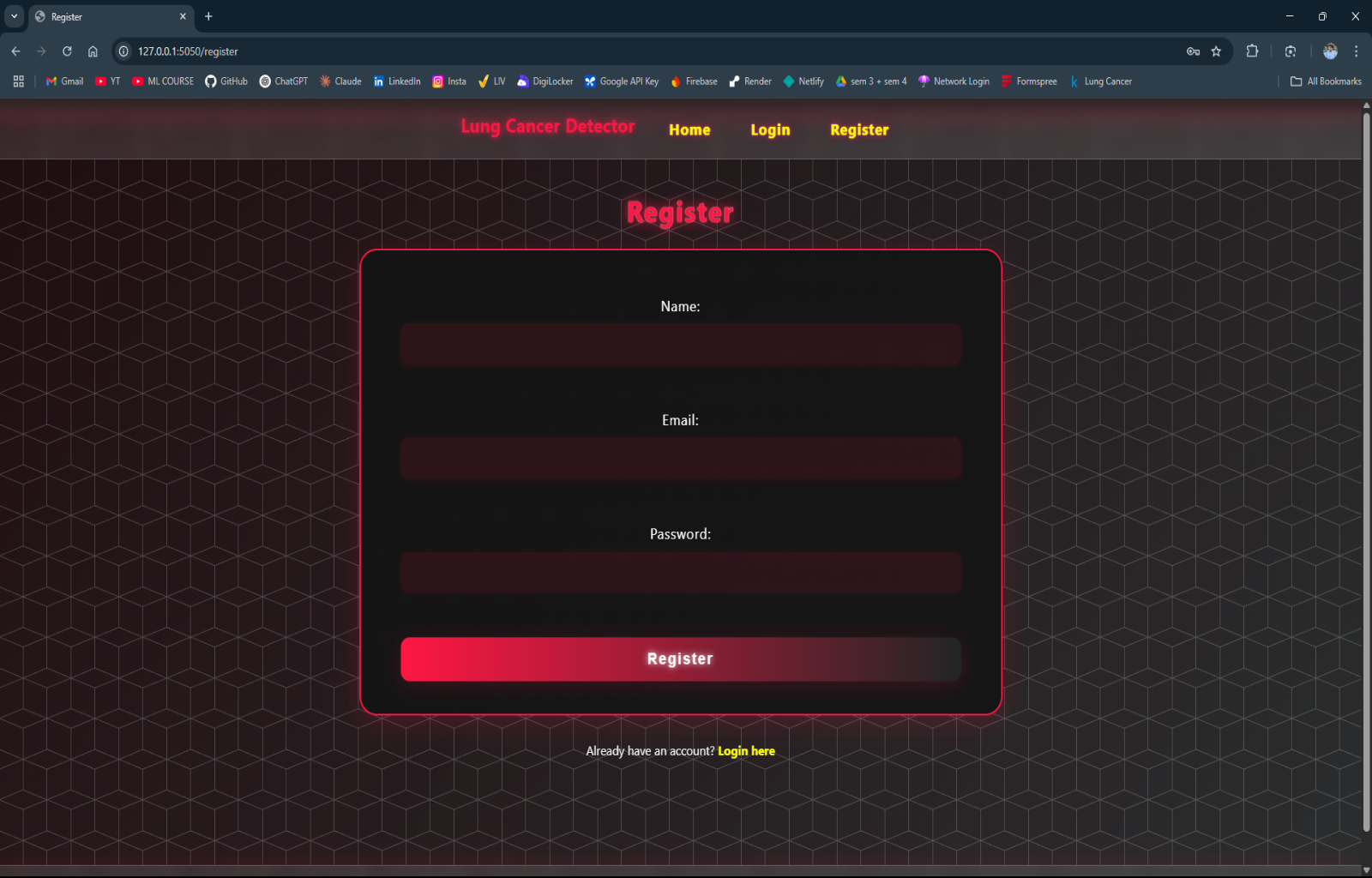
**Main Page:-**



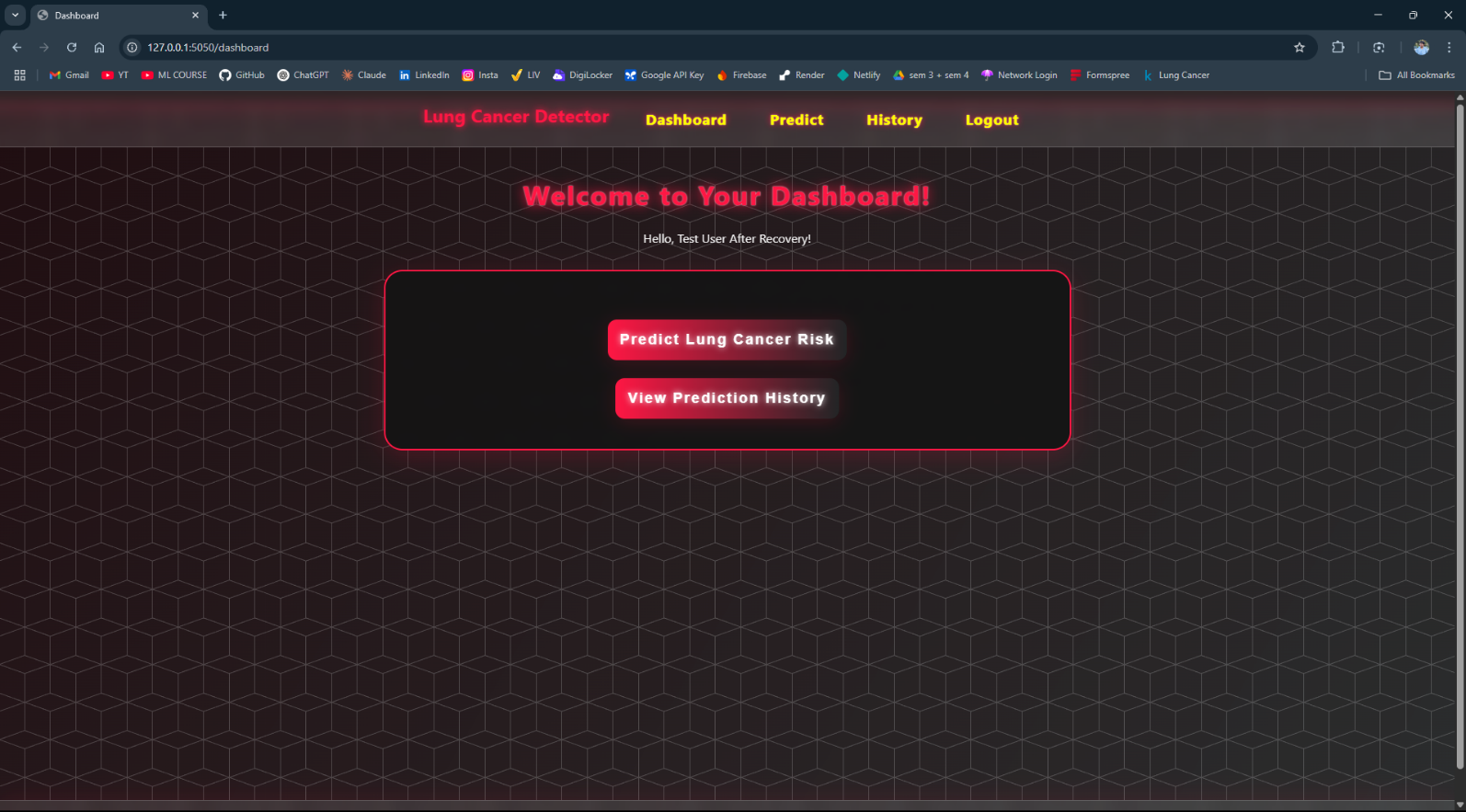
**Login Page:-**



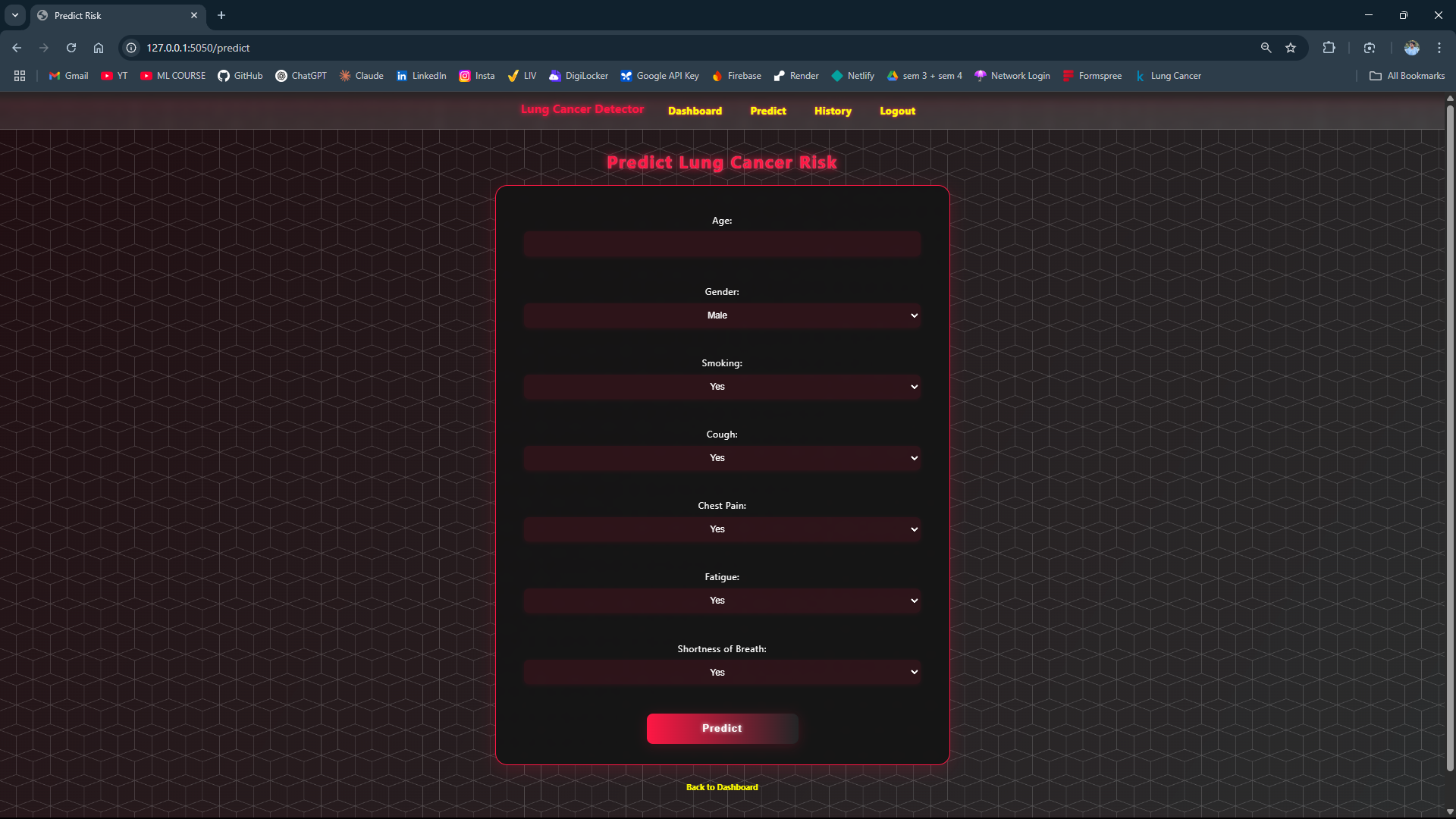
**Register Page:-**

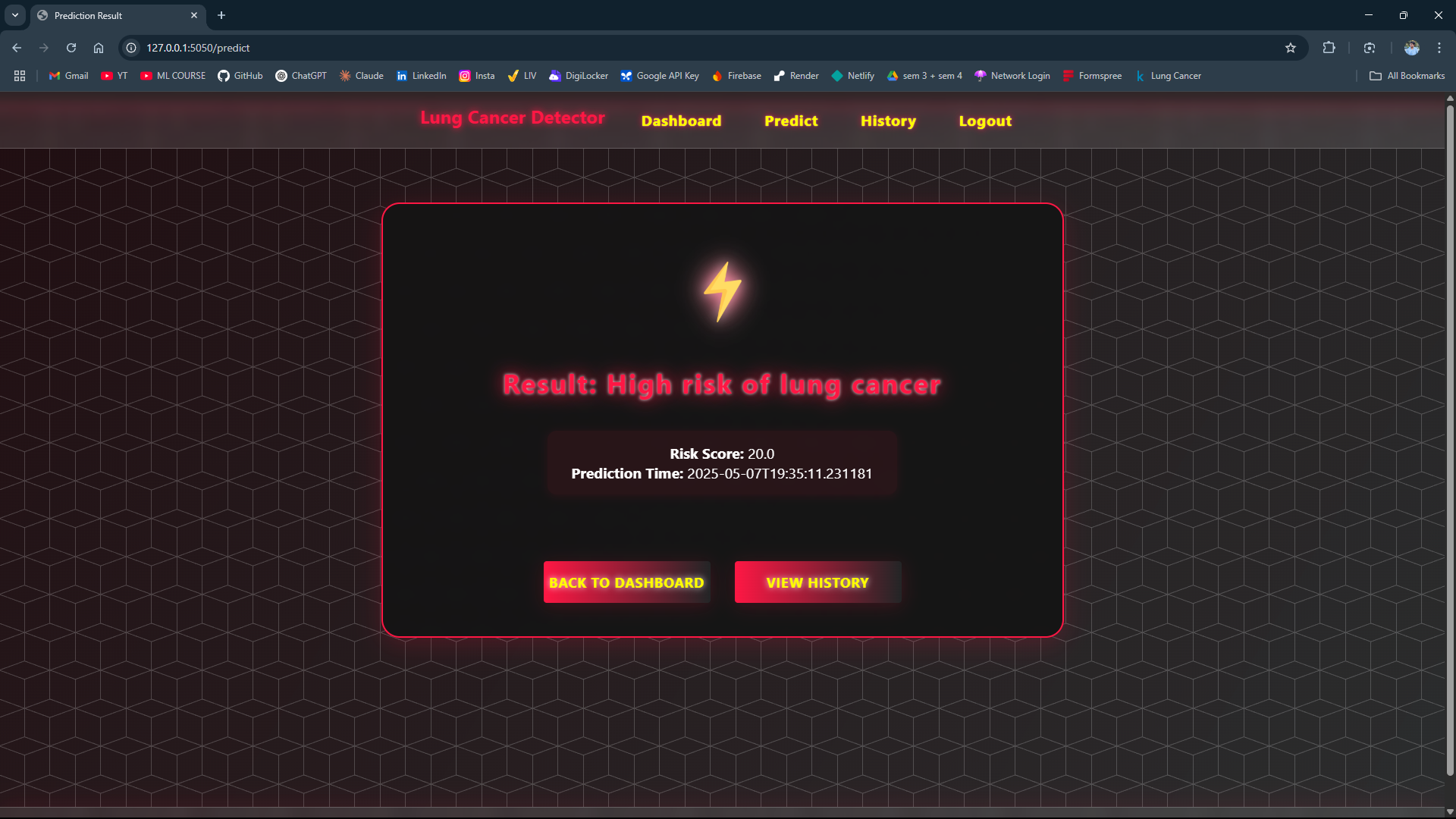


**Dashboard Page:**



Predict Page:





History Page:



**FUTURE SCOPE AND LIMITATIONS**

The **Lung Cancer Detection System** has a promising future with several areas open for enhancement and expansion. By applying key principles of software engineering such as **encapsulation**, **separation of concerns**, and **modular design**, the system is built to support future upgrades and maintainability.

* Reusability:

The application is developed with modular code structures and reusable components, which allows for efficient updates and feature additions in future versions. Reusability significantly reduces design, development, and testing efforts by enabling both:

* Sharing of newly written components across different modules.
* Reuse of previously developed components in new health-related projects.
* Understand ability: The system’s codebase follows best practices with **clear naming conventions**, **well-documented functions**, and **coherent logic**, making it easier for other developers (or the original creator after a time lapse) to understand and maintain the code. Each method is small, self-contained, and focused on a single responsibility to improve readability and debugging.
* Cost-effectiveness: The project was developed within a limited budget and time frame while still meeting core functional and technical requirements. The system aims to provide accurate predictions, secure data handling, and smooth user experience—all within a low-cost implementation strategy suitable for scaling in real-world healthcare setups.

### LIMITATIONS:-

* + The current system is based on **user-input symptom data**, which may sometimes be inaccurate or subjective.
  + It does not yet integrate with **hospital information systems** or **live medical equipment**.
  + The model prediction is **risk-based**, not a definitive diagnosis, and should not replace professional medical advice.

**CONCLUSION**

After all the hard work put into developing the Lung Cancer Detection System, it stands as a reliable software solution aimed at assisting early detection of lung cancer based on user-input symptoms and relevant data. This application streamlines the diagnostic support process by minimizing manual efforts, improving data handling, and offering timely risk assessments. Its simple and user-friendly interface ensures that users—whether patients or healthcare workers—can interact with the system easily. Overall, the system enhances the efficiency of preliminary lung cancer risk evaluation and supports informed medical follow-up, contributing to better health awareness and outcomes.

NPTEL

NPTEL