Design Report for Prescription Database Web Application

CST363 - Intro to Database Systems

Yusra Ashar, Sam Numan, Joseph Lee Young

February 18, 2025

1. Introduction

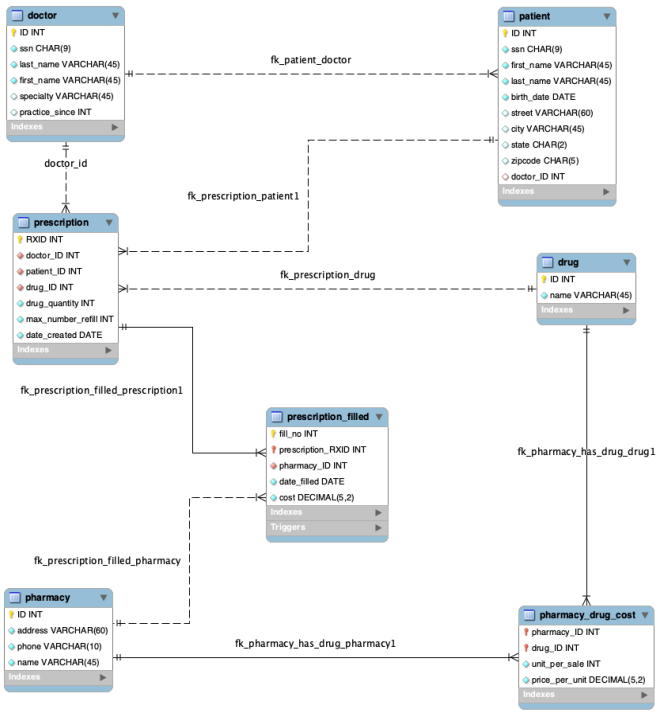
The Prescription Database Web Application is a robust and efficient system designed to manage prescription data for doctors, patients, and pharmacies. The goal of this application is to streamline prescription handling by ensuring that prescriptions are accurately created, stored, and processed for fulfillment. With an intuitive web interface, doctors can write new prescriptions, patients can request refills, and pharmacists can verify and process prescription requests.

This project is built using Spring Boot for backend development, MySQL for database management, and standard HTML forms for user interaction. The database is structured to include doctors, patients, drugs, pharmacies, and prescription records, ensuring a seamless connection between each entity. The system implements data validation to prevent errors, such as invalid patient IDs, incorrect prescription refills, or nonexistent pharmacy records.

Throughout this report, we present the updated database design, schema implementation, and working user interface, demonstrating how the system performs under different scenarios. The report also outlines the testing procedures used to validate different functionalities, ensuring that prescription creation, refills, and updates are processed correctly while preventing unauthorized actions.

2. ER Diagram

Below is the ER diagram for the prescription database, which was initially designed in Lab 18 and updated for this lab.



Description of Entities and Relationships:

Patients, doctors, prescriptions, drugs, and pharmacies are strong entities in the database. They are all identified by an ID, which is each entity's primary key, except for the prescription entity, which is identified by ‘rxid’. A patient’s name and other personal identifying information (PII) are attributes of the patient entity. The patient entity has one foreign key, which is a reference to the primary key of the doctor table. Doctors have attributes for PII and also have additional fields for their first year in practice and their specialty. Drugs have an attribute for their name. Information such as the cost of the drug is dependent on where it is distributed and is found elsewhere in the database. Pharmacy entities have attributes for their name, address, and contact information. Prescription entities contain attributes that reference the doctor

that wrote the prescription, the patient the prescription is for, and the drug being prescribed. Additional attributes in the prescription entity contain information pertaining to the cost and the amount of the drug.

Information pertaining to the fills of prescription are contained in the prescription_filled entity. The prescription_filled composite primary key contains its own unique ID and a reference to the 'rxid' of the prescription. It also contains a foreign key that references the ID of the pharmacy that filled the prescription. Additional attributes in prescription_filled provide information on the date and cost of the prescription. The cost and units per sale of the drug can be found using the pharmacy_drug_cost entity. Its composite primary key contains two foreign keys that reference a drug and the pharmacy that distributed it.

Relationships:

Doctors share a many to one relationship with patients meaning that a doctor can have many patients, but patients can only have one primary doctor. Doctors can write many prescriptions, patients can have many prescriptions, and a drug can be prescribed many times, but each prescription only belongs to a singular doctor, patient and drug. The relationship between prescription and pharmacy is many to many and is represented by the prescription_filled entity. Pharmacies and drugs also share a many to many relationship and are represented by the pharmacy_drug_cost entity.

3. SQL Schema

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE
,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION'
;

-- -----------------------------------------------------
-- Schema prescription
-- -----------------------------------------------------
DROP SCHEMA IF EXISTS `prescription` ;
CREATE SCHEMA IF NOT EXISTS `prescription` DEFAULT CHARACTER SET
utf8 ;
USE `prescription` ;

-- -----------------------------------------------------
-- Table `prescription`.`doctor`
-- -----------------------------------------------------
```

```
CREATE TABLE IF NOT EXISTS `prescription`.`doctor` (
  `ID` INT NOT NULL AUTO_INCREMENT,
  `ssn` CHAR(9) NOT NULL,
  `last_name` VARCHAR(45) NOT NULL,
  `first_name` VARCHAR(45) NOT NULL,
  `specialty` VARCHAR(45) NULL,
  `practice_since` INT NULL,
  PRIMARY KEY (`ID`),
  UNIQUE INDEX `social_security_UNIQUE` (`ssn` ASC) VISIBLE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `prescription`.`patient`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `prescription`.`patient` (
  `ID` INT NOT NULL AUTO_INCREMENT,
  `ssn` CHAR(9) NOT NULL,
  `first_name` VARCHAR(45) NOT NULL,
  `last_name` VARCHAR(45) NOT NULL,
  `birth_date` DATE NOT NULL,
  `street` VARCHAR(60) NULL,
  `city` VARCHAR(45) NULL,
  `state` CHAR(2) NULL,
```

```
  `zipcode` CHAR(5) NULL,
  `doctor_ID` INT NULL,
  PRIMARY KEY (`ID`),
  UNIQUE INDEX `ID_UNIQUE` (`ID` ASC) VISIBLE,
  UNIQUE INDEX `ssn_UNIQUE` (`ssn` ASC) VISIBLE,
  INDEX `fk_patient_doctor_idx` (`doctor_ID` ASC) VISIBLE,
  CONSTRAINT `fk_patient_doctor`
    FOREIGN KEY (`doctor_ID`)
    REFERENCES `prescription`.`doctor` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `prescription`.`drug`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `prescription`.`drug` (
  `ID` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`ID`),
  UNIQUE INDEX `name_UNIQUE` (`name` ASC) VISIBLE)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `prescription`.`prescription`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `prescription`.`prescription` (
  `RXID` INT NOT NULL AUTO_INCREMENT,
  `doctor_ID` INT NOT NULL,
  `patient_ID` INT NOT NULL,
  `drug_ID` INT NOT NULL,
  `drug_quantity` INT UNSIGNED NOT NULL,
  `max_number_refill` INT UNSIGNED NOT NULL,
  `date_created` DATE NOT NULL,
  PRIMARY KEY (`RXID`),
  INDEX `fk_prescription_patient1_idx` (`patient_ID` ASC)
VISIBLE,
  UNIQUE INDEX `RXID_UNIQUE` (`RXID` ASC) VISIBLE,
  INDEX `fk_prescription_drug_idx` (`drug_ID` ASC) VISIBLE,
  CONSTRAINT `doctor_id`
    FOREIGN KEY (`doctor_ID`)
    REFERENCES `prescription`.`doctor` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_prescription_patient1`
    FOREIGN KEY (`patient_ID`)
```

```
    REFERENCES `prescription`.`patient` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_prescription_drug`
    FOREIGN KEY (`drug_ID`)
    REFERENCES `prescription`.`drug` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `prescription`.`pharmacy`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `prescription`.`pharmacy` (
  `ID` INT NOT NULL AUTO_INCREMENT,
  `address` VARCHAR(60) NOT NULL,
  `phone` VARCHAR(10) NOT NULL,
  `name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`ID`),
  UNIQUE INDEX `address_UNIQUE` (`address` ASC) VISIBLE)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `prescription`.`pharmacy_drug_cost`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `prescription`.`pharmacy_drug_cost` (
  `pharmacy_ID` INT NOT NULL,
  `drug_ID` INT NOT NULL,
  `unit_per_sale` INT UNSIGNED NOT NULL,
  `price_per_unit` DECIMAL(5,2) UNSIGNED NOT NULL,
  PRIMARY KEY (`pharmacy_ID`, `drug_ID`),
  INDEX `fk_pharmacy_has_drug_drug1_idx` (`drug_ID` ASC)
VISIBLE,
  INDEX `fk_pharmacy_has_drug_pharmacy1_idx` (`pharmacy_ID` ASC)
VISIBLE,
  CONSTRAINT `fk_pharmacy_has_drug_pharmacy1`
    FOREIGN KEY (`pharmacy_ID`)
    REFERENCES `prescription`.`pharmacy` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_pharmacy_has_drug_drug1`
    FOREIGN KEY (`drug_ID`)
    REFERENCES `prescription`.`drug` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `prescription`.`prescription_filled`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `prescription`.`prescription_filled`
(
  `fill_no` INT NOT NULL AUTO_INCREMENT,
  `prescription_RXID` INT NOT NULL,
  `pharmacy_ID` INT NOT NULL,
  `date_filled` DATE NOT NULL,
  `cost` DECIMAL(5,2) NOT NULL,
  PRIMARY KEY (`fill_no`, `prescription_RXID`),
  INDEX `fk_prescription_filled_pharmacy_idx` (`pharmacy_ID`
ASC) VISIBLE,
  CONSTRAINT `fk_prescription_filled_prescription1`
    FOREIGN KEY (`prescription_RXID`)
    REFERENCES `prescription`.`prescription` (`RXID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_prescription_filled_pharmacy`
    FOREIGN KEY (`pharmacy_ID`)
    REFERENCES `prescription`.`pharmacy` (`ID`)
    ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
USE `prescription`;


DELIMITER $$


CREATE DEFINER = CURRENT_USER TRIGGER
`prescription`.`prescription_filled_BEFORE_INSERT` BEFORE INSERT
ON `prescription_filled` FOR EACH ROW
BEGIN
    DECLARE refill_count INT;
    DECLARE max_refills INT;


    SELECT COUNT(*), p.max_number_refill INTO refill_count,
max_refills
    FROM prescription_filled pf
    INNER JOIN prescription p ON pf.prescription_RXID = p.RXID
    WHERE pf.prescription_RXID = NEW.prescription_RXID
    GROUP BY p.RXID; -- Crucial: Add GROUP BY clause
```

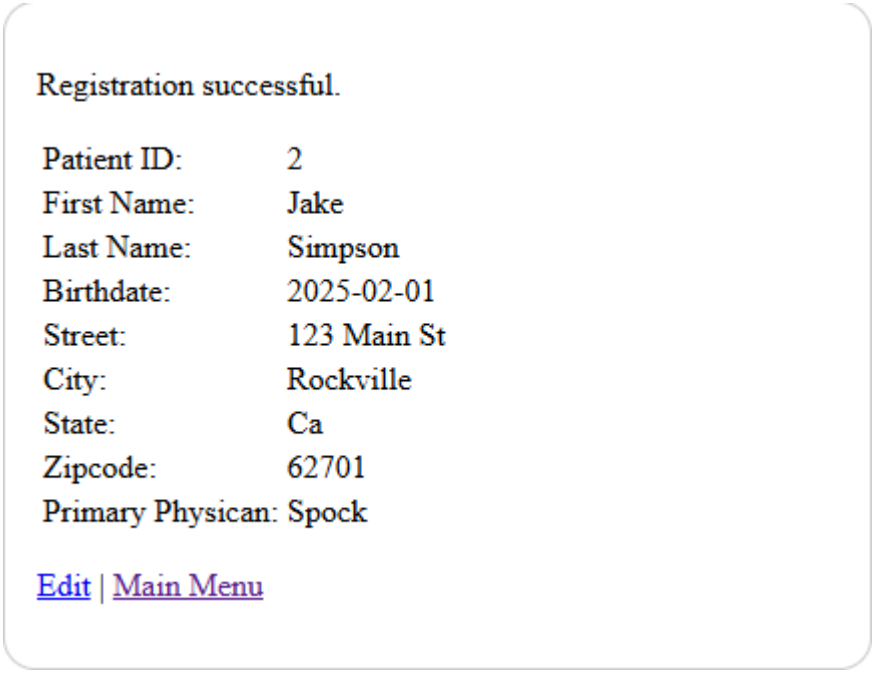
```
    IF refill_count >= max_refills THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "prescription
complete";
    END IF;
END$$

DELIMITER ;
```

---

4. Screenshots of Working Application

Register as a New Patient

- Successful Registration

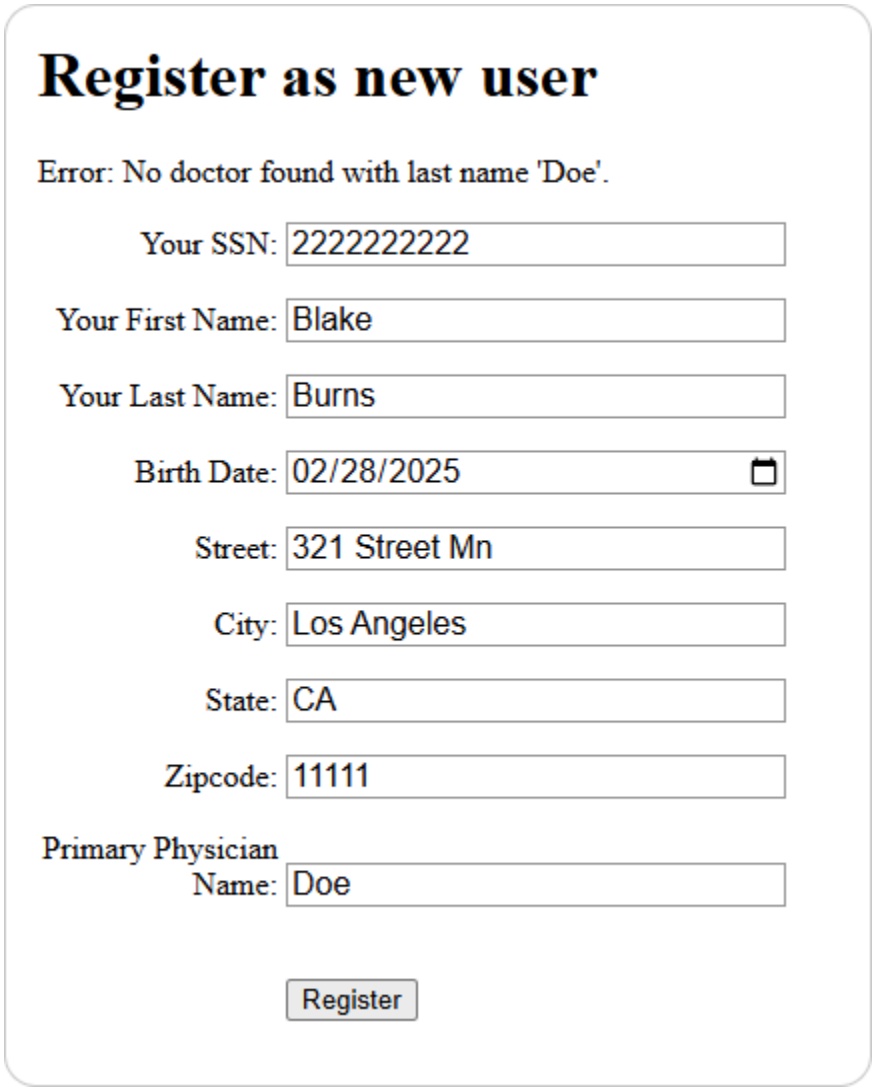


Registration successful.

Patient ID:	2
First Name:	Jake
Last Name:	Simpson
Birthdate:	2025-02-01
Street:	123 Main St
City:	Rockville
State:	Ca
Zipcode:	62701
Primary Physican:	Spock

Edit | Main Menu

- Failed Registration



Create a Prescription

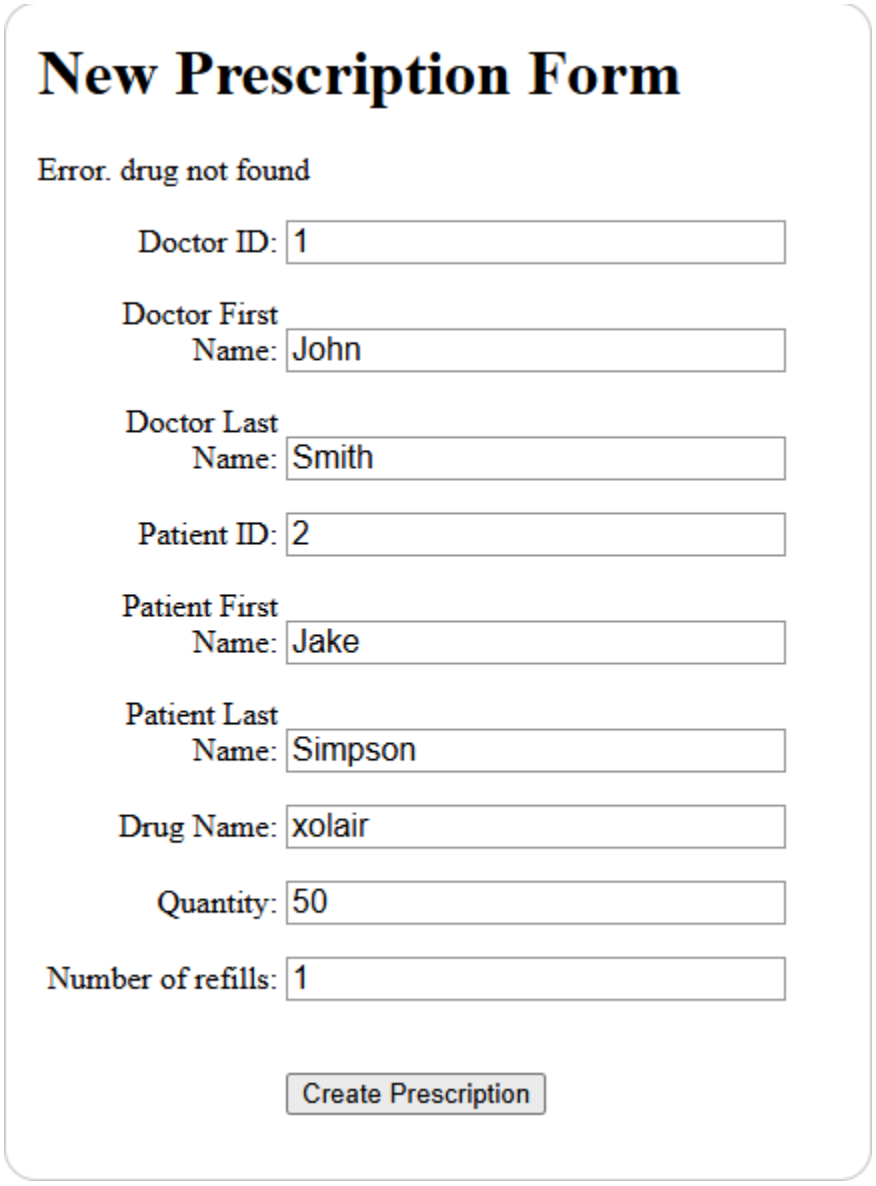
- Successful Prescription Creation

Prescription created successfully

Rx:	1
Doctor ID:	3
First Name:	John
Last Name:	Spock
Patient ID:	2
First Name:	Jake
Last Name:	Simpson
Drug:	lisinopril
Quantity:	90
Refills remaining:	0
Pharmacy ID:	0
Name:	
Address:	
Phone:	
Date Filled:	
Cost: $	

Main Menu

- Failed Prescription Creation



Fill a Prescription

- Failed Fill with Invalid Pharmacy Name

- Failed Fill with Invalid Rxid

- Successful Prescription Fill

Prescription filled successfully.

Rx:	1
Doctor ID:	1
First Name:	John
Last Name:	Smith
Patient ID:	2
First Name:	Jake
Last Name:	Simpson
Drug:	1
Quantity:	90
Refills remaining:	9
Pharmacy ID:	1
Name:	CVS
Address:	123 Main St
Phone:	1111111111
Date Filled:	2025-02-17
Cost: $	123.99

Main Menu

Patient Profile and Update

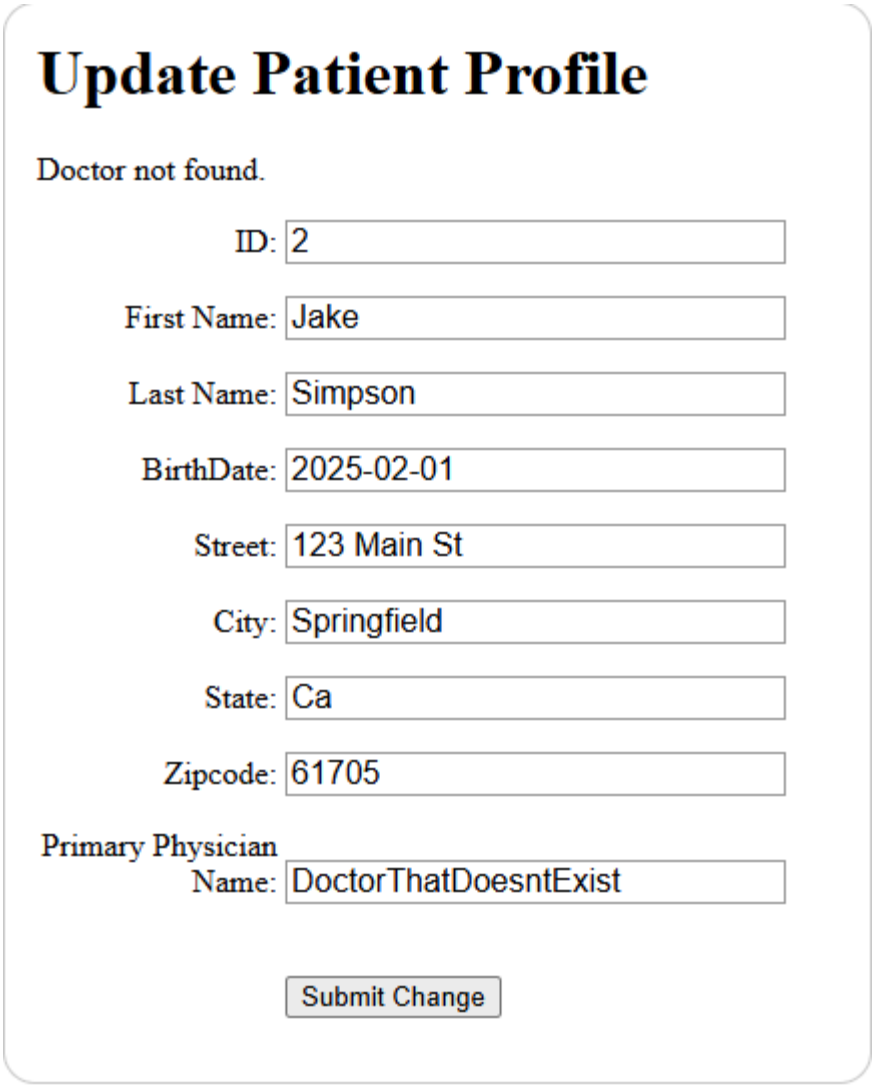
- Successful Patient Update

Update successful

Patient ID:	2
First Name:	Jake
Last Name:	Simpson
Birthdate:	2025-02-01
Street:	123 Main St
City:	Springfield
State:	Ca
Zipcode:	61705
Primary Physican:	Smith

Edit | Main Menu

- Failed Patient Update



5. Conclusion

The Prescription Database Web Application successfully implements a system for managing prescriptions, allowing doctors to create prescriptions, patients to request refills, and pharmacies to process prescription fulfillment. The project effectively integrates Spring Boot for backend logic, MySQL for data storage, and standard HTML forms for user interaction, ensuring a structured and organized workflow.

Throughout Lab 19, we developed and tested key functionalities, including doctor and patient registration, prescription creation, and prescription fulfillment. The database schema was designed to maintain relationships between doctors, patients, prescriptions, and pharmacies, ensuring that all records were properly linked. The application processes valid prescription requests, tracks refill limits, and verifies pharmacy details, demonstrating its ability to handle essential prescription management tasks.

The system performed as expected during testing, correctly handling prescription creation, retrieval, and fulfillment while maintaining data accuracy. Users were able to register, submit prescription requests, and view their details through a simple and functional web interface.

While the current implementation meets the project requirements, future improvements could include:

- Enhancing the user interface for better usability.
- Adding authentication and role-based access control for security.
- Implementing additional reporting features to track prescription history.

Overall, this project provided hands-on experience in database design, SQL integration, and web application development using Java and Spring Boot. The Prescription Database Web Application serves as a solid foundation for learning about full-stack development and database-driven applications.