Phone Book Spring Boot - Input Validation

This document outlines the design and implementation details of a Spring Boot web application designed to manage phone book entries.

Running the Application

Docker Build: Build the Docker image using:

```
docker build -t sxk7070_project:prod .
```

Run Application: Start the application on port 8080:

```
docker run -p 8080:8080 sxk7070_project:prod
```

Unit Tests

Unit tests are located in `src/test/java/com/cse5382/assignment/Controller/ControllerTest.java`. To run tests and view the report:

Docker Build (Tests): Build the image with the `tests` stage:

```
docker build -t sxk7070_project:tests . --target tests -o
<OUTPUT_DIRECTORY>
```

NOTE: Replace `<OUTPUT_DIRECTORY>` with your desired location on the host machine.

Description

This application utilizes Spring Boot and the DAO pattern to interact with an SQLite database for phone book data persistence.

REST API Endpoints

- `/phoneBook/add` (POST): Adds a new phone book entry.
- `/phoneBook/deleteByName` (PUT): Deletes an entry by name.
- `/phoneBook/deleteByNumber` (PUT): Deletes an entry by phone number.
- `/phoneBook/list` (GET): Retrieves all phone book entries.

These endpoints return a `PhoneBookResponse.java` object containing the HTTP status code and relevant feedback.

Database

- SQLite is used for data persistence.
- ORMLite is used to interact with the database. (ORMLite)
- The database schema includes two tables:
  - `phonebook`: Stores the name (primary key) and phone number.
  - `users`: Stores username (primary key), password (bcrypt encoded), and role.

Authentication & Authorization

- JWT authentication is implemented using Spring Security.
- Each request requires a valid JWT token in the `Authorization` header with a `Bearer <jwt_token>` value.
- A user can obtain a token by providing credentials at `/phoneBook/api/auth/authenticate`.
- Two predefined users exist with different access levels (`READ` and `READ_WRITE`). User details are stored in `application.properties` and `application-test.properties`.

Input Validation

- Regular expressions validate name and phone number formats in the controller layer to prevent injection attacks. Patterns are defined in `AppConstants.java`.
- Name: Ensures it starts with a capital letter, allows middle names/spaces/hyphens/apostrophes, and handles initials/suffixes.

```
^[A-Z][a-zA-Z]*[-']?[a-zA-Z]+,?
?[a-zA-Z]*[-']?[a-zA-Z]+
?[a-zA-Z]*[-']?[a-zA-Z]*[.]?$
```

- Phone Number: Supports various formats including US numbers with/without separators and international numbers with country codes.

```
^\d{5}$|

^\d{5}[. ]\d{5}$|

^\d{3}[-. ]\d{4}$|
^\+?\b([1-9]|[1-9][0-9]|[1-9][0-9][0-8])\b[-.\(
]{0,2}\d{2,3}[ \-.\)]{0,2}\d{3}[-. ]\d{4}$|

^[-.\( ]?\d{2,3}[ \-.\)]\d{3}[-. ]\d{4}$|

^(00|011)[-.\( ]?\d{0,3}[ -.\)][-.\( ]?\d{2,3}[
-.\)]\d{3}[-. ]\d{4}$|

^[+45. ]{0,4}\d{4}[. ]\d{4}$|

^[+45. ]{0,4}\d{2}[. ]\d{2}[. ]\d{2}[. ]\d{2}$
```

Logging

- Phone book operations are logged in `audits.log` (Service layer) and the console. Configuration details are in `logback.xml`.

Testing

- Unit tests (JUnit) are written in `ControllerTest.java` to test controller methods with various inputs.
- Separate configuration (`application-test.properties`) is used for testing with Spring Profiles.

Errors and Exceptions

- `PhonebookControllerAdvice.java` handles errors and exceptions globally, including `SQLException` and custom business logic exceptions.

Assumptions

1. Two predefined users exist in the database with roles.
2. Phone book logs are cleared on application restart.
3. In-memory H2 database is used for isolated testing.
4. JWT tokens are valid for 30 minutes with BCrypt password encoding.

Pros

1. Input validation using annotations (`@Pattern`, `@Valid`) to prevent malicious data.
2. Separate testing and production databases.
3. ORMLite for lightweight ORM performance.
4. Thread pooling is used for database connections (`JdbcPooledConnectionSource`).
5. Stateless JWT authentication.

6. Centralized error handling.

Cons

1. While usernames/passwords are not hardcoded, a more secure configuration management approach is recommended. Ex: Config Server