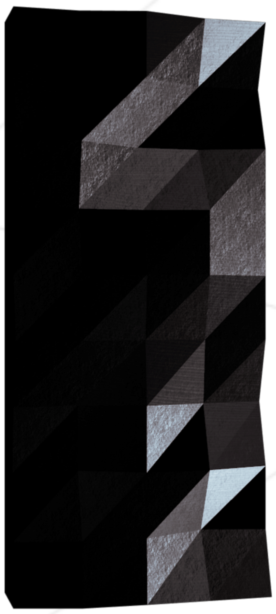


Spring Boot

More, Than Spring

Summary:
let's make a final MVP for our web application using the most current Java-based Spring Boot development technology

Version: 1.00

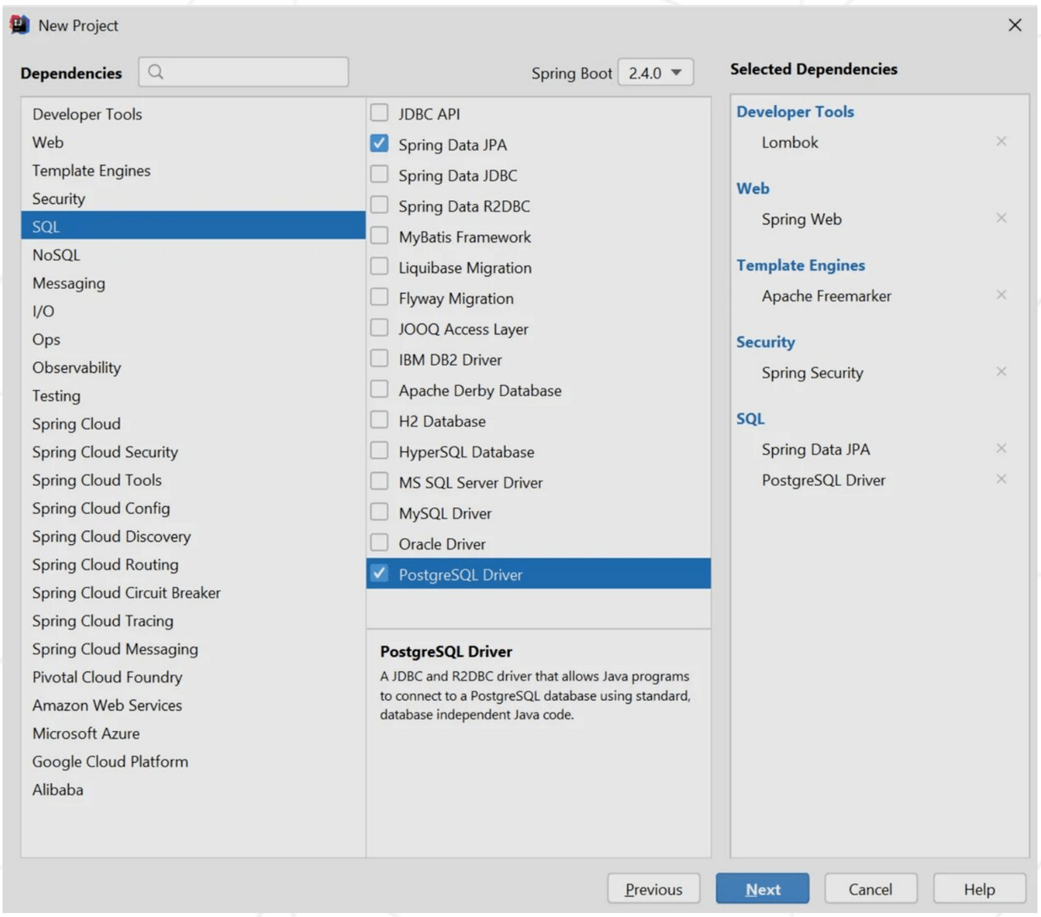
Contents

Chapter I

Preamble

Spring Boot is a reinterpretation of Spring framework. It allows you to focus solely on the development process, eliminating complex manipulations to customize your application. It is exactly this technology that is used to develop modern microservices. All important application parameters are often specified in a special properties file.

The use of Spring Boot substantially speeds up development while minimizing configuration error rate. However, a truly effective use of Spring Boot is only possible with a thorough understanding of web application infrastructure.



New Project

Dependencies — Spring Boot 2.4.0

Developer Tools
Web
Template Engines
Security
SQL
NoSQL
Messaging
I/O
Ops
Observability
Testing
Spring Cloud
Spring Cloud Security
Spring Cloud Tools
Spring Cloud Config
Spring Cloud Discovery
Spring Cloud Routing
Spring Cloud Circuit Breaker
Spring Cloud Tracing
Spring Cloud Messaging
Pivotal Cloud Foundry
Amazon Web Services
Microsoft Azure
Google Cloud Platform
Alibaba

- [ ] JDBC API
- [✓] Spring Data JPA
- [ ] Spring Data JDBC
- [ ] Spring Data R2DBC
- [ ] MyBatis Framework
- [ ] Liquibase Migration
- [ ] Flyway Migration
- [ ] JOOQ Access Layer
- [ ] IBM DB2 Driver
- [ ] Apache Derby Database
- [ ] H2 Database
- [ ] HyperSQL Database
- [ ] MS SQL Server Driver
- [ ] MySQL Driver
- [ ] Oracle Driver
- [✓] PostgreSQL Driver

PostgreSQL Driver
A JDBC and R2DBC driver that allows Java programs to connect to a PostgreSQL database using standard, database independent Java code.

Selected Dependencies

Developer Tools
- Lombok

Web
- Spring Web

Template Engines
- Apache Freemarker

Security
- Spring Security

SQL
- Spring Data JPA
- PostgreSQL Driver

Creating a Spring Boot application using Spring Initializr in IntelliJ IDEA (current version is 3.0.1)

Chapter II

General Rules

- Use this page as the only reference. Do not listen to any rumors and speculations about how to prepare your solution.
- You mus use the latest LTS version of Java. Make sure that compiler and interpreter of this version are installed on your machine.
- You must use GraalVM to run your code.
- You can use IDE to write and debug the source code (we recommend IntelliJ Idea).
- The code is read more often than written. Read carefully the document where code formatting rules are given. When performing each exercise, make sure you follow the generally accepted Oracle standards
- Pay attention to the permissions of your files and directories.
- To be assessed, your solution must be in your GIT repository.
- You should not leave in your directory any other file than those explicitly specified by the exercise instructions. It is recommended that you modify your .gitignore to avoid accidents.
- When you need to get precise output in your programs, it is forbidden to display a precalculated output instead of performing the exercise correctly.
- Have a question? Ask your neighbor on the right. Otherwise, try with your neighbor on the left.
- Your reference manual: mates / Internet / Google. And one more thing. There's an answer to any question you may have on Stackoverflow. Learn how to ask questions correctly.
- Read the examples carefully. They may require things that are not otherwise specified in the subject.
- Use "System.out" for output
- And may the Force be with you!
- Never leave that till tomorrow which you can do today ;)

Chapter III

Rules of the project

- The project you are implementing now can use a database from the previous project.
- All pages in this project should have the same functionality as in previous projects.
- Project structure should match the standard `Spring Boot` structure
- For each task, you shall attach schema.sql and data.sql files where you describe a schema of a database being created and test data, respectively.

Chapter IV

Exercice 00: Spring Security

	Exercise 00
Spring Security	
Turn-in directory : *ex*00/	
Files to turn in : `Cinema-folder`	
Allowed functions : `n/a`	

We will create a truly "secure" service. Use `Spring Security` framework inside your `Spring Boot` application to implement role-based access to all pages:

Role	URL
ADMIN	/admin/panel/halls
ADMIN	/admin/panel/films
ADMIN	/admin/panel/sessions
ADMIN	/profile
Any authorized user	/session/search
Any authorized user	/films/{film-id}/chat/messages
Any authorized user	/films/{film-id}/chat
Any authorized user	/signIn, /signUp

In case an authorized user requests `/signIn` or `/signUp` page, they should be redirected to `/profile` page (`/admin/panel/halls` for administrator).

In case an unauthorized user requests a page other than `/signIn` or `/signUp`, they shall be redirected to the login page.

Repository layer in this task should be implemented using `Spring Data JPA` technology. Below is an example of a `JPA` repository:

```
public interface MessagesRepository extends JpaRepository<Message> {
    Optional<Message> findByText(String text);
    List<Message> findAllByAuthor(User author);
}
```

Additional requirements:

- Prepare implementations of standard `Spring Security` interfaces, `UserDetails` and `UserDetailsService`.
- `/signIn` page shall be prepared independently (use of built-in `Spring Security` page is `prohibited`).
- Implement "remember-me" functionality on `/signIn` page.
- Ensure protection against `csrf` attacks.
- User's role shall be stored as an `Enum` value.

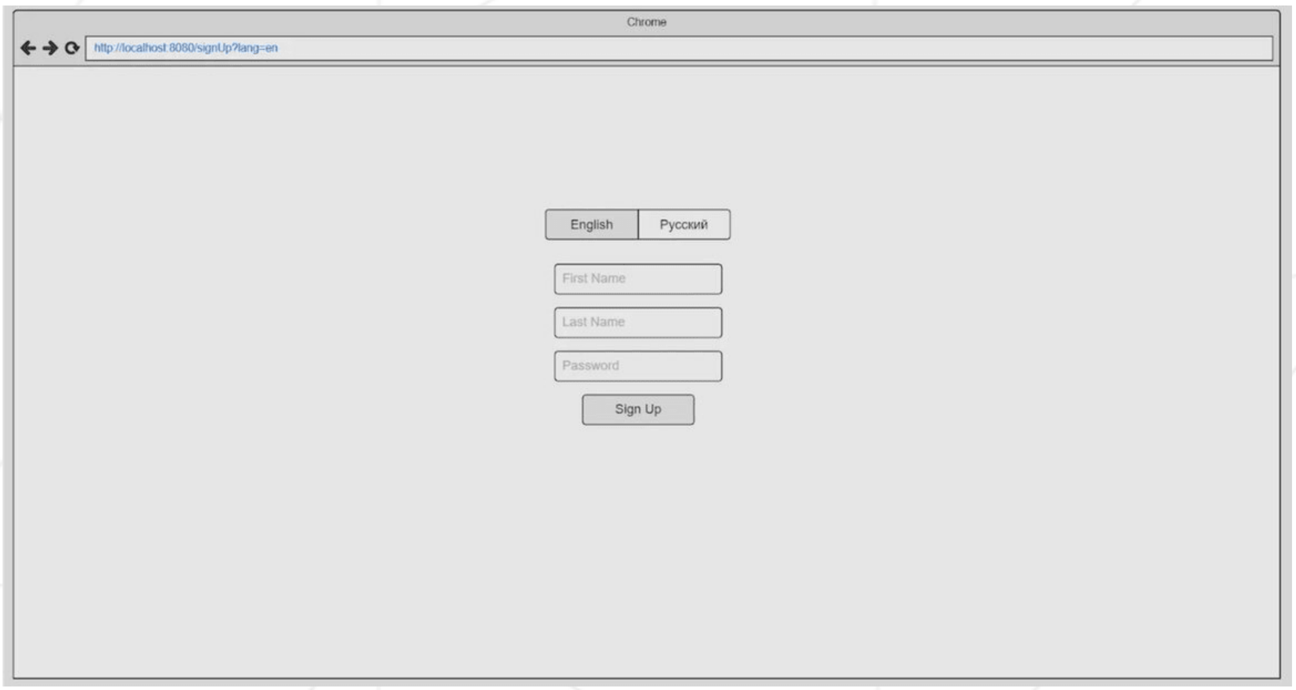
Chapter V

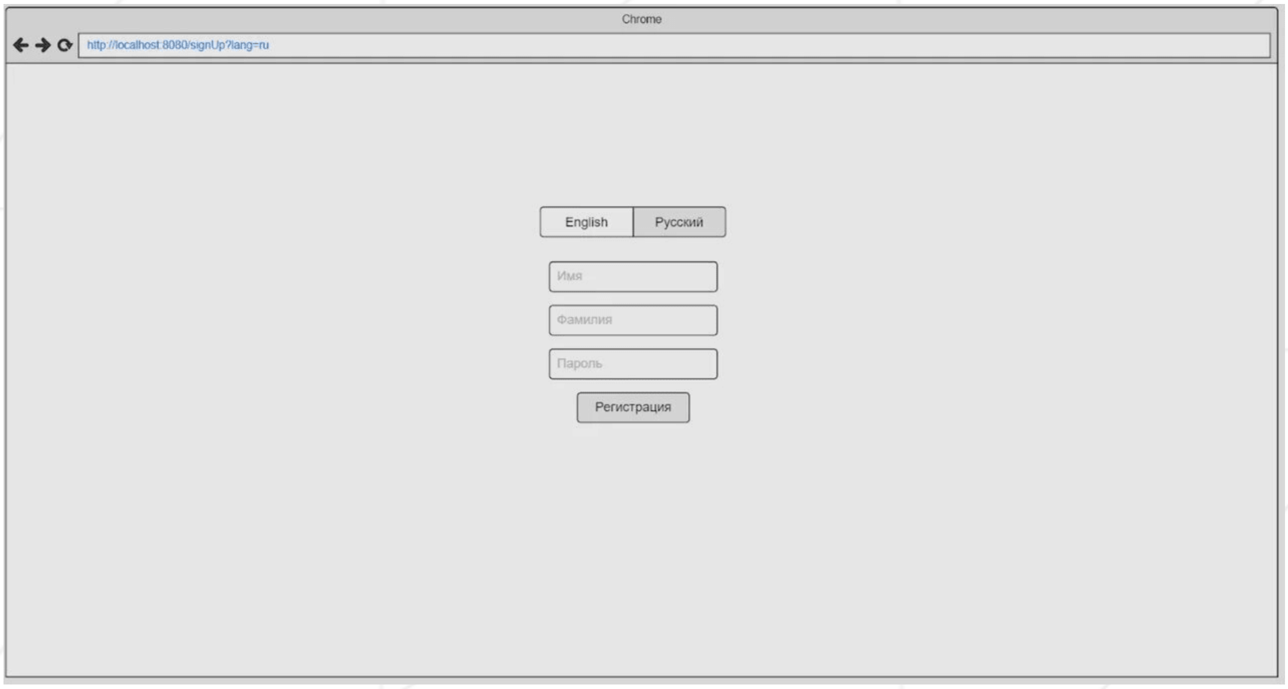
Exercice 01: Localization & Validation

	Exercise 01
Localization & Validation	
Turn-in directory : *ex*01/	
Files to turn in : `Cinema-folder`	
Allowed functions : `n/a`	

Multiple language support (in this task, two languages of your choice) shall be implemented in your applciation.

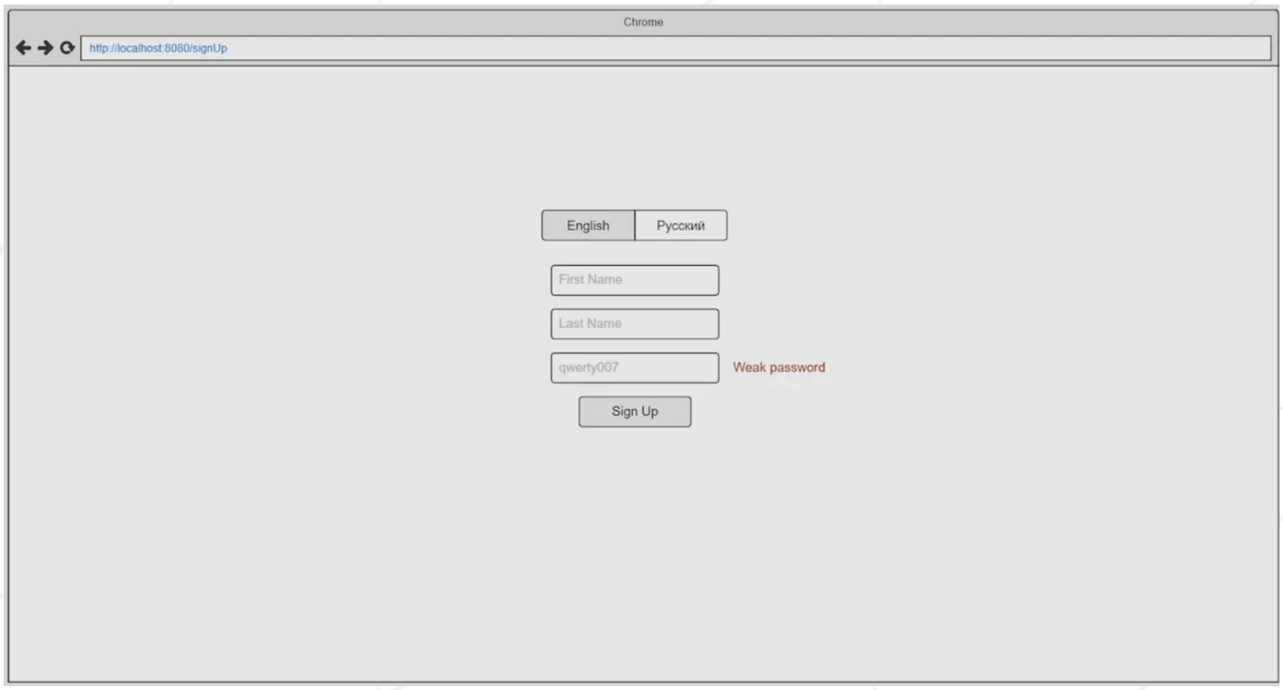
A change of localization shall occur if a request with lang parameter was submitted for the requested page. Examples of how localization works are provided below:





Localization information shall be stored in browser cookies. Therefore, when a page and/or application is reloaded, selected localization shall be preserved. You shall provide localization support for at least any three pages.

Localization is closely related to data validation. Each user shall be able to see a message about incorrectly filled form in the context of selected localization, for example:



In your exercise, you shall implement support for at least two languages and validate a registration form according to the following rules:

- First and last name fields shall be non-empty

- Email shall match the email recording format
- Phone number shall match +(code)digits pattern, e.g., +7(777)777777
- Password shall contain uppercase, lowercase letters, and at least one digit; field length shall be at least 8 characters.

Additional requirements:

- Provide properties files for localization and error messages.
- Provide .bin files - `LocaleResolver`, `LocaleChangeInterceptor`, `LocalValidatorFactoryBean`, `MessageSource`, `MessageCodesResolver`
- Use `javax.validation.constraints.*` annotations.
- Password validation shall be implemented using `javax.validation.ConstraintValidator` and `@ValidPassword` custom annotation. Example:

```
@ValidPassword(message = "{errors.incorrect.password}")
private String password;
```

Chapter VI

Exercice 03 : Mails

Exercise 03
Mails
Turn-in directory : *ex*03/
Files to turn in : `Cinema-folder`
Allowed functions : `n/a`

In this exercise, confirmation of a registered account using a link sent to a user's email shall be implemented.

Thus, for `User` model, you need to add a field that indicates if an account is confirmed (`CONFRIMED`, `NOT_CONFIRMED`). Now, only verified users shall be able to access the application being developed. Administrator is verified by default.

Upon registration, a confirmation link in the following format shall be sent to a user's email: `http://{host:port}/confirm/{UUID}`

When clicking on the link, a user gets a page where they can log into their personal account by entering a username and a password.

Email submission shall be implemented using: `org.springframework.mail.javamail.JavaMailSender`.

To send an email, you need to use an existing mailbox, e.g., `Gmail.` `SMTP` settings for `example@gmail.com` mailbox with `p4ssw0rd` as password:

```
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=example@gmail.com
spring.mail.password=p4ssw0rd
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.properties.mail.smtp.allow8bitmime=true
spring.mail.properties.mail.smtp.ssl.trust=smtp.gmail.com
spring.mail.properties.mail.debug=true
```

The way how we pass this configuration to Spring Boot may have changed, check the documentation of the (latest) version you are using.