Dokumentacja PaySlipGenerator

**Autor**

Paweł Żurawski

**Projekt na GitHub**

<https://github.com/19pawel970415/PaySlipGeneratorApp>

**Data ostatniej aktualizacji dokumentacji**

08 kwietnia 2024

**Polimorfizm, a więc wielopostaciowość**

Polimorfizm w programowaniu odnosi się do zdolności różnych obiektów do reagowania na te same komunikaty lub metody w różny sposób. Oznacza to, że obiekty różnych klas mogą być traktowane w ten sam sposób, co umożliwia pisanie bardziej ogólnego i elastycznego kodu.

Polimorfizm pozwala na pisanie bardziej elastycznego i skalowalnego kodu, ponieważ umożliwia traktowanie różnych obiektów w sposób jednolity. Dzięki temu programista może napisać kod, który może działać z wieloma różnymi typami obiektów, co prowadzi do większej przejrzystości i łatwiejszego zarządzania kodem.

W kontekście programowania komponentowego, polimorfizm może być wykorzystywany do tworzenia modułowych, elastycznych komponentów, które mogą być używane i łączone w różnych konfiguracjach, co przyczynia się do budowy skalowalnych i łatwo rozbudowywalnych systemów.

W projekcie realizowany jest polimorfizm poprzez wykorzystanie dziedziczenia i interfejsów.

1. **Dziedziczenie:**
   * Klasy **JavaDeveloper**, **CSharpDeveloper**, **CppDeveloper**, **FrontEndDeveloper** oraz **DatabaseAnalyst** dziedziczą po klasie abstrakcyjnej **Employee**. Dzięki temu, można traktować te obiekty jako obiekty ogólnego typu **Employee**, co pozwala na wykonywanie operacji na nich w sposób jednolity.
   * Metoda **generatePaySlip()** jest zaimplementowana w klasach dziedziczących po **Employee** i nadpisana w każdej z tych klas. Dzięki temu, można wywołać tę metodę na obiektach różnych klas, a w wyniku otrzymać dane dotyczące wypłaty dla konkretnego typu pracownika.
2. **Interfejsy:**
   * Interfejs **DataToDbHandler** definiuje metodę **addDataToDb()**, która jest implementowana przez klasy reprezentujące różne typy pracowników.
   * Każda klasa, która implementuje interfejs **DataToDbHandler**, musi zaimplementować metodę **addDataToDb()**. Dzięki temu, można traktować różne typy pracowników jednolicie poprzez wywołanie tej samej metody, niezależnie od ich konkretnego typu.
   * Przykładowo, klasa **JavaDeveloper** implementuje metodę **addDataToDb()** tak samo jak klasy **CSharpDeveloper**, **CppDeveloper**, **FrontEndDeveloper** oraz **DatabaseAnalyst**, co pozwala na jednolite dodawanie danych do bazy niezależnie od typu pracownika.

Dzięki wykorzystaniu dziedziczenia oraz interfejsów, projekt wykorzystuje polimorfizm, co pozwala na traktowanie różnych typów pracowników w sposób jednolity i elastyczny.

**1. Cel Projektu**

Generowanie pasków wypłat dla różnych typów pracowników w firmie. Klasy reprezentują różne rodzaje pracowników, takie jak programiści Java, C#, C++, Front-end developerzy, analitycy baz danych itp. Każdy z tych pracowników ma swoje specyficzne cechy, takie jak narzędzia używane w pracy (np. IntelliJ, Visual Studio), doświadczenie (junior, mid, senior) oraz wynagrodzenie podstawowe.

Projekt realizuje funkcje generowania wypłat dla tych pracowników na podstawie ich specyficznych danych, takich jak imię, nazwisko, login, hasło, narzędzie, doświadczenie itp. Metody w klasach pracowników takie jak generatePaySlip() generują raporty o wypłacie dla danego pracownika.

Dodatkowo, istnieją metody do dodawania danych do bazy danych, projekt obejmuje zarządzanie danymi pracowników w bazie danych.

W skrócie, celem tego projektu jest automatyzacja procesu generowania pasków wypłat dla różnych typów pracowników w firmie oraz zarządzanie danymi tych pracowników w bazie danych.

Wymagania, które spełnia projekt są następujące:

1. automatyzacja procesu generowania wypłat: Klient oczekuje, że projekt umożliwi automatyczne generowanie raportów o wypłatach dla każdego pracownika na podstawie jego loginu. Dzięki temu firma będzie mogła szybko i efektywnie przetwarzać wypłaty dla swoich pracowników.
2. Elastyczność i skalowalność: Klient oczekuje, że projekt będzie elastyczny i łatwo skalowalny, umożliwiając dodawanie nowych typów pracowników oraz dostosowywanie wypłat do ich specyficznych cech, takich jak narzędzia pracy i doświadczenie.
3. Zarządzanie danymi pracowników: Projekt powinien umożliwiać klientowi łatwe zarządzanie danymi pracowników, przede wszystkim dodawanie nowych pracowników do bazy danych.
4. Bezpieczeństwo danych: Klient oczekuje, że projekt będzie zapewniał wysoki poziom bezpieczeństwa danych pracowników, w tym poufność, integralność i dostępność danych. Dostęp do danych pracowników powinien być kontrolowany za pomocą mechanizmów uwierzytelniania i autoryzacji.
5. Łatwość obsługi: Projekt powinien być łatwy w obsłudze i intuicyjny dla użytkowników końcowych, umożliwiając szybkie generowanie raportów o wypłatach oraz zarządzanie danymi pracowników bez konieczności głębokiego zrozumienia technicznych szczegółów.

**2. Architektura**

Główna struktura projektu:

src

└── main

├── java

│ └── com.example.payslipgenerator

│ ├── employees

│ │ ├── CppDeveloper

│ │ ├── CSharpDeveloper

│ │ ├── DatabaseAnalyst

│ │ ├── DataToDbHandler

│ │ ├── Employee

│ │ ├── Experience

│ │ ├── FrontEndDeveloper

│ │ ├── JavaDeveloper

│ │ └── PythonDeveloper

│ ├── service

│ │ ├── Main

│ │ └── PaySlipGeneratorService

│ ├── LoginController

│ ├── PaySlipGeneratorApplication

│ ├── PaySlipGeneratorController

│ └── RegisterController

├── resources

│ ├── static

│ ├── templates

│ ├── application.properties

│ └── hibernate.cfg.xml

└── webapp

└── WEB-INF

└── views

├── index.jsp

├── loggedIn.jsp

├── notSignedIn.jsp

├── register\_cpp.jsp

├── register\_csharp.jsp

├── register\_database.jsp

├── register\_frontend.jsp

├── register\_java.jsp

├── register\_python.jsp

├── registration\_failure.jsp

├── registration\_success.jsp

├── signin.jsp

└── signup.jsp

1. **java**: Główny katalog zawierający kod źródłowy aplikacji napisany w języku Java.
   * **com.example.payslipgenerator**: Pakiet główny aplikacji.
     + **employees**: Pakiet zawierający klasy reprezentujące różne typy pracowników.
       - W tym pakiecie znajdują się klasy reprezentujące różne role pracowników, takie jak CppDeveloper, CSharpDeveloper itp.
     + **service**: Pakiet zawierający klasy serwisowe.
       - **Main**: Klasa główna aplikacji, odpowiedzialna za uruchomienie aplikacji.
       - **PaySlipGeneratorService**: Klasa obsługująca logikę generowania list płac.
     + **controllers**: Pakiet zawierający kontrolery Spring MVC.
       - **LoginController**: Kontroler obsługujący logowanie użytkowników.
       - **PaySlipGeneratorController**: Kontroler obsługujący żądania związane z generowaniem list płac.
       - **RegisterController**: Kontroler obsługujący rejestrację użytkowników.
   * **PaySlipGeneratorApplication**: Główna klasa aplikacji Spring Boot, zawierająca metodę main, która uruchamia aplikację.
2. **resources**: Katalog zawierający zasoby aplikacji.
   * **static**: Katalog zawierający pliki statyczne, takie jak CSS, JavaScript itp.
   * **templates**: Katalog zawierający szablony JSP.
   * **application.properties**: Plik konfiguracyjny Spring Boot, zawierający ustawienia aplikacji.
   * **hibernate.cfg.xml**: Plik konfiguracyjny Hibernate, zawierający ustawienia dostępu do bazy danych.
3. **webapp**: Katalog zawierający pliki zasobów aplikacji webowej.
   * **WEB-INF**: Katalog zawierający pliki konfiguracyjne aplikacji webowej.
     + **views**: Katalog zawierający pliki JSP używane do renderowania interfejsu użytkownika.
       - Znajdują się tu pliki takie jak index.jsp, loggedIn.jsp, signin.jsp itp., które definiują różne widoki aplikacji.

Ta struktura projektu organizuje kod i zasoby aplikacji w sposób przejrzysty, ułatwiając zarządzanie i rozwój aplikacji.

**3. Funkcjonalności**

3.1 Frontend

3.1.1.index.jsp:

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Pay Slip Generator</title>  
 <style>  
 body {  
 margin: 0;  
 padding: 0;  
 font-family: Arial, sans-serif;  
 background-color: #f0f0f0;  
 }  
  
 .container {  
 text-align: center;  
 position: absolute;  
 top: 50%;  
 left: 50%;  
 transform: translate(-50%, -50%);  
 }  
  
 .title {  
 font-size: 2.5rem;  
 margin-bottom: 20px;  
 color: #333;  
 }  
  
 .btn {  
 display: inline-block;  
 padding: 10px 20px;  
 background-color: #4CAF50;  
 color: white;  
 text-decoration: none;  
 font-size: 1.2rem;  
 margin: 0 10px;  
 border-radius: 5px;  
 transition: background-color 0.3s;  
 border: none;  
 cursor: pointer;  
 outline: none;  
 }  
  
 .btn:hover {  
 background-color: #45a049;  
 }  
 </style>  
</head>  
<body>  
  
<div class="container">  
 <h1 class="title">Pay Slip Generator</h1>  
 <button class="btn" onclick="*location*.href='signin'">Sign in</button>  
 <button class="btn" onclick="*location*.href='signup'">Sign up</button>  
</div>  
  
</body>  
</html>

Opis:

* <!DOCTYPE html>: Deklaracja typu dokumentu HTML.
* <html lang="en">: Otwarcie tagu HTML z określeniem języka dokumentu.
* <head>: Sekcja zawierająca metadane dokumentu oraz odniesienia do zewnętrznych zasobów.
* <meta charset="UTF-8">: Określenie kodowania znaków.
* <meta name="viewport" content="width=device-width, initial-scale=1.0">: Ustawienia viewportu dla responsywności na urządzeniach mobilnych.
* <title>Pay Slip Generator</title>: Tytuł strony.
* <style>: Sekcja zawierająca style CSS dla dokumentu.
* body: Stylizacja dla elementu body.
* .container: Stylizacja dla klasy container, centrująca zawartość na ekranie.
* .title: Stylizacja dla klasy title, określająca wygląd tytułu.
* .btn: Stylizacja dla klasy btn, definiująca wygląd przycisków.
* .btn:hover: Stylizacja dla klasy btn po najechaniu kursorem.
* </style>: Zakończenie sekcji ze stylami.
* </head>: Zamknięcie sekcji head.
* <body>: Sekcja zawierająca zawartość strony.
* .container: Kontener dla treści strony.
* .title: Tytuł strony.
* .btn: Przyciski "Sign in" i "Sign up" z odpowiednimi atrybutami onclick, które przekierowują użytkownika na strony logowania i rejestracji.
* </body>: Zamknięcie sekcji body.
* </html>: Zamknięcie tagu html.

3.1.2.loggedIn.jsp:

<%@ **page** import="com.example.payslipgenerator.service.PaySlipGeneratorService" %>  
<%@ **page** import="com.example.payslipgenerator.employees.Employee" %>  
<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Welcome back</title>  
 <style>  
 body {  
 font-family: Arial, sans-serif;  
 background-color: #f0f0f0;  
 text-align: center;  
 position: relative;  
 }  
  
 .container {  
 margin-top: 50px;  
 }  
  
 h1 {  
 color: #333;  
 margin-bottom: 20px;  
 }  
  
 .error-message {  
 color: red;  
 margin-bottom: 20px;  
 }  
  
 table {  
 margin: auto;  
 border-collapse: collapse;  
 width: 50%;  
 border: 2px solid #000;  
 border-radius: 10px;  
 }  
  
 th, td {  
 padding: 10px;  
 text-align: left;  
 border-bottom: 1px solid #ddd;  
 }  
  
 th {  
 background-color: #4CAF50;  
 color: white;  
 border-right: 1px solid #fff;  
 }  
  
 td {  
 background-color: #f2f2f2;  
 }  
  
 .try-again-btn {  
 padding: 10px 20px;  
 background-color: #4CAF50;  
 color: white;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 text-decoration: none;  
 font-size: 1.2rem;  
 }  
  
 .try-again-btn:hover {  
 background-color: #45a049;  
 }  
  
 .logout-icon {  
 position: absolute;  
 top: 10px;  
 right: 20px;  
 font-size: 40px;  
 color: #333;  
 text-decoration: none;  
 }  
  
 .logout-icon:hover {  
 color: #f00;  
 }  
 </style>  
</head>  
<body>  
<a href="index" class="logout-icon">&#128274;</a>  
<h1></h1>  
<%  
 String username = request.getParameter("username");  
  
 Employee employee = PaySlipGeneratorService.*generatePaySlipForEmployee*(username);  
  
 if (employee != null) {  
 String[] paySlipData = employee.generatePaySlip();  
%>  
<h1>Welcome back <%= username %></h1>  
<article>  
 <div class="box">  
 <h2>Here is your payslip!</h2>  
 <table>  
 <tr>  
 <th>Name</th>  
 <td><%= paySlipData[0] %> <%= paySlipData[1] %></td>  
 </tr>  
 <tr>  
 <th>Position</th>  
 <td><%= paySlipData[2] %></td>  
 </tr>  
 <tr>  
 <th>Experience</th>  
 <td><%= paySlipData[3] %></td>  
 </tr>  
 <tr>  
 <th>Salary</th>  
 <td><%= paySlipData[4] %></td>  
 </tr>  
 </table>  
 </div>  
</article>  
<%  
} else {  
%>  
<div class="error-message">  
 <p>No such user.</p>  
</div>  
<%  
 }  
%>  
</body>  
</html>

Opis:

* **<%@ page import="com.example.payslipgenerator.service.PaySlipGeneratorService" %>**: Importuje klasę **PaySlipGeneratorService** z pakietu **com.example.payslipgenerator.service**.
* **<%@ page import="com.example.payslipgenerator.employees.Employee" %>**: Importuje klasę **Employee** z pakietu **com.example.payslipgenerator.employees**.
* **<a href="index" class="logout-icon">&#128274;</a>**: Ikona wylogowania, przekierowuje do strony głównej.
* **<h1>Welcome back <%= username %></h1>**: Pozdrawia użytkownika po nazwie użytkownika.
* **<% ... %>**: Skrypty Java w kodzie JSP. Tutaj wywoływane są metody serwisu do generowania wypłaty oraz przetwarzane dane w celu wygenerowania tabeli z danymi o wypłacie.
* **String username = request.getParameter("username");**: Pobiera parametr **username** przekazany w żądaniu HTTP.
* **Employee employee = PaySlipGeneratorService.generatePaySlipForEmployee(username);**: Wywołuje metodę **generatePaySlipForEmployee** z serwisu w celu pobrania obiektu **Employee** na podstawie nazwy użytkownika.
* **String[] paySlipData = employee.generatePaySlip();**: Wywołuje metodę **generatePaySlip** obiektu **Employee**, aby uzyskać dane wypłaty.
* **<%= paySlipData[0] %>**: Wyświetla dane wypłaty w tabeli.
* **<div class="error-message">**: Komunikat błędu wyświetlany, gdy nie ma użytkownika o podanej nazwie.
* **</body>**: Koniec sekcji body.
* **</html>**: Koniec dokumentu HTML.

3.1.3.notSignedIn.jsp:

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Login Failed</title>  
 <style>  
 body {  
 font-family: Arial, sans-serif;  
 background-color: #f0f0f0;  
 text-align: center;  
 display: flex;  
 justify-content: center;  
 align-items: center;  
 height: 100vh;  
 position: relative;  
 }  
  
 .container {  
 width: 50%;  
 }  
  
 h1 {  
 color: #333;  
 text-align: center;  
 margin-bottom: 20px;  
 }  
  
 p {  
 color: red;  
 margin-bottom: 20px;  
 }  
  
 .try-again-btn {  
 padding: 10px 20px;  
 background-color: #4CAF50;  
 color: white;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 text-decoration: none;  
 font-size: 1.2rem;  
 }  
  
 .try-again-btn:hover {  
 background-color: #45a049;  
 }  
  
 .home-icon {  
 position: absolute;  
 top: 10px;  
 right: 10px;  
 font-size: 40px;  
 color: #333;  
 text-decoration: none;  
 }  
  
 .home-icon:hover {  
 color: #f00;  
 }  
  
 </style>  
</head>  
<body>  
<a href="index" class="home-icon">&#127968;</a>  
<div class="container">  
 <h1>Login Failed</h1>  
 <p>Incorrect username or password. Please try again.</p>  
 <a href="signin" class="try-again-btn">Try Again</a>  
</div>  
</body>  
</html>

Opis:

* **<!DOCTYPE html>**: Deklaracja typu dokumentu HTML.
* **<html lang="en">**: Otwarcie tagu HTML z określeniem języka dokumentu.
* **<head>**: Sekcja zawierająca metadane dokumentu oraz odniesienia do zewnętrznych zasobów.
* **<meta charset="UTF-8">**: Określenie kodowania znaków.
* **<meta name="viewport" content="width=device-width, initial-scale=1.0">**: Ustawienia viewportu dla responsywności na urządzeniach mobilnych.
* **<title>Login Failed</title>**: Tytuł strony.
* **<style>**: Sekcja zawierająca style CSS dla dokumentu.
* **body**: Stylizacja dla elementu **body**.
* **.container**: Stylizacja dla klasy **container**, określająca jego szerokość i pozycję.
* **h1**: Stylizacja dla nagłówka h1.
* **p**: Stylizacja dla paragrafu.
* **.try-again-btn**: Stylizacja dla przycisku "Try Again".
* **.home-icon**: Stylizacja dla ikony "Home", która przekierowuje użytkownika do strony głównej.
* **</style>**: Zakończenie sekcji ze stylami.
* **</head>**: Zamknięcie sekcji head.
* **<body>**: Sekcja zawierająca zawartość strony.
* **<a href="index" class="home-icon">&#127968;</a>**: Ikona "Home", przekierowująca użytkownika do strony głównej.
* **<div class="container">**: Kontener dla zawartości strony.
* **<h1>Login Failed</h1>**: Nagłówek informujący o nieudanym logowaniu.
* **<p>Incorrect username or password. Please try again.</p>**: Komunikat informujący o błędnym loginie lub haśle.
* **<a href="signin" class="try-again-btn">Try Again</a>**: Przycisk "Try Again" przekierowujący użytkownika do strony logowania.
* **</body>**: Koniec sekcji body.
* **</html>**: Koniec dokumentu HTML.

3.1.4.register\_cpp.jsp:

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Register as C++ Developer</title>  
 <style>  
 body {  
 font-family: Arial, sans-serif;  
 background-color: #f0f0f0;  
 text-align: center;  
 display: flex;  
 justify-content: center;  
 align-items: center;  
 height: 100vh;  
 position: relative;  
 }  
  
 .container {  
 width: 50%;  
 }  
  
 h1 {  
 color: #333;  
 text-align: center;  
 margin-bottom: 20px;  
 }  
  
 .form-container {  
 width: 300px;  
 margin: 0 auto;  
 background-color: #fff;  
 padding: 20px;  
 border-radius: 5px;  
 box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);  
 }  
  
 input[type="text"],  
 input[type="password"],  
 select {  
 width: 100%;  
 padding: 10px;  
 margin: 10px 0;  
 border: 1px solid #ccc;  
 border-radius: 5px;  
 box-sizing: border-box;  
 }  
  
 input[type="submit"] {  
 width: 100%;  
 padding: 10px;  
 background-color: #4CAF50;  
 color: #fff;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 }  
  
 input[type="submit"]:hover {  
 background-color: #45a049;  
 }  
  
 .home-icon {  
 position: absolute;  
 top: 10px;  
 right: 10px;  
 font-size: 40px;  
 color: #333;  
 text-decoration: none;  
 }  
  
 .home-icon:hover {  
 color: #f00;  
 }  
  
 .password-error {  
 border: 1px solid red **!important**;  
 }  
  
 </style>  
</head>  
<body>  
<a href="index" class="home-icon">&#127968;</a>  
<div class="container">  
 <h1>Registration Page for C++ Developer</h1>  
 <div class="form-container">  
 <form action="register\_cpp" method="post" onsubmit="return validatePassword()">  
 <input type="text" name="name" placeholder="Name" required><br>  
 <input type="text" name="surname" placeholder="Surname" required><br>  
 <input type="text" name="login" placeholder="Login" required><br>  
 <input type="password" name="password" id="password" placeholder="Password" required><br>  
 <input type="password" name="confirm\_password" id="confirm\_password" placeholder="Confirm Password" required onkeyup="validatePassword()"><br>  
 <input type="text" name="toolname" placeholder="Tool Name" required><br>  
 <select name="experience" required>  
 <option value="" disabled selected>Select Experience</option>  
 <option value="JUNIOR">Junior</option>  
 <option value="MID">Mid</option>  
 <option value="SENIOR">Senior</option>  
 </select><br>  
 <input type="submit" value="Sign up">  
 </form>  
 </div>  
</div>  
  
<script>  
 function validatePassword() {  
 var password = *document*.getElementById("password");  
 var confirm\_password = *document*.getElementById("confirm\_password");  
  
 if (password.value !== confirm\_password.value) {  
 confirm\_password.classList.add("password-error");  
 return false;  
 } else {  
 confirm\_password.classList.remove("password-error");  
 return true;  
 }  
 }  
</script>  
  
</body>  
</html>

Opis:

* **<!DOCTYPE html>**: Deklaracja typu dokumentu HTML.
* **<html lang="en">**: Otwarcie tagu HTML z określeniem języka dokumentu.
* **<head>**: Sekcja zawierająca metadane dokumentu oraz odniesienia do zewnętrznych zasobów.
* **<meta charset="UTF-8">**: Określenie kodowania znaków.
* **<meta name="viewport" content="width=device-width, initial-scale=1.0">**: Ustawienia viewportu dla responsywności na urządzeniach mobilnych.
* **<title>Register as C++ Developer</title>**: Tytuł strony.
* **<style>**: Sekcja zawierająca style CSS dla dokumentu.
* **body**: Stylizacja dla elementu **body**.
* **.container**: Stylizacja dla klasy **container**, określająca jego szerokość i pozycję.
* **h1**: Stylizacja dla nagłówka h1.
* **.form-container**: Stylizacja dla kontenera formularza rejestracji.
* **input[type="text"], input[type="password"], select**: Stylizacja dla pól tekstowych i wyboru w formularzu.
* **input[type="submit"]**: Stylizacja dla przycisku submit w formularzu.
* **.home-icon**: Stylizacja dla ikony "Home", która przekierowuje użytkownika do strony głównej.
* **.password-error**: Stylizacja dla pola potwierdzania hasła w przypadku błędnego wprowadzenia hasła.
* **</style>**: Zakończenie sekcji ze stylami.
* **</head>**: Zamknięcie sekcji head.
* **<body>**: Sekcja zawierająca zawartość strony.
* **<a href="index" class="home-icon">&#127968;</a>**: Ikona "Home", przekierowująca użytkownika do strony głównej.
* **<div class="container">**: Kontener dla zawartości strony.
* **<h1>Registration Page for C++ Developer</h1>**: Nagłówek informujący o stronie rejestracji dla deweloperów w języku C++.
* **<form>**: Formularz rejestracji dewelopera w języku C++.
* **onsubmit="return validatePassword()"**: Wywołanie funkcji walidacji hasła podczas wysyłania formularza.
* **onkeyup="validatePassword()"**: Wywołanie funkcji walidacji hasła podczas wprowadzania hasła.
* **<script>**: Sekcja zawierająca skrypt JavaScript do walidacji hasła.
* **</body>**: Koniec sekcji body.
* **</html>**: Koniec dokumentu HTML.

3.1.5.register\_csharp.jsp:

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Register as C# Developer</title>  
 <style>  
 body {  
 font-family: Arial, sans-serif;  
 background-color: #f0f0f0;  
 text-align: center;  
 display: flex;  
 justify-content: center;  
 align-items: center;  
 height: 100vh;  
 position: relative;  
 }  
  
 .container {  
 width: 50%;  
 }  
  
 h1 {  
 color: #333;  
 text-align: center;  
 margin-bottom: 20px;  
 }  
  
 .form-container {  
 width: 300px;  
 margin: 0 auto;  
 background-color: #fff;  
 padding: 20px;  
 border-radius: 5px;  
 box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);  
 }  
  
 input[type="text"],  
 input[type="password"],  
 select {  
 width: 100%;  
 padding: 10px;  
 margin: 10px 0;  
 border: 1px solid #ccc;  
 border-radius: 5px;  
 box-sizing: border-box;  
 }  
  
 input[type="submit"] {  
 width: 100%;  
 padding: 10px;  
 background-color: #4CAF50;  
 color: #fff;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 }  
  
 input[type="submit"]:hover {  
 background-color: #45a049;  
 }  
  
 .home-icon {  
 position: absolute;  
 top: 10px;  
 right: 10px;  
 font-size: 40px;  
 color: #333;  
 text-decoration: none;  
 }  
  
 .home-icon:hover {  
 color: #f00;  
 }  
  
 .password-error {  
 border: 1px solid red **!important**;  
 }  
  
 </style>  
</head>  
<body>  
<a href="index" class="home-icon">&#127968;</a>  
<div class="container">  
 <h1>Registration Page for C# Developer</h1>  
 <div class="form-container">  
 <form action="register\_csharp" method="post" onsubmit="return validatePassword()">  
 <input type="text" name="name" placeholder="Name" required><br>  
 <input type="text" name="surname" placeholder="Surname" required><br>  
 <input type="text" name="login" placeholder="Login" required><br>  
 <input type="password" name="password" id="password" placeholder="Password" required><br>  
 <input type="password" name="confirm\_password" id="confirm\_password" placeholder="Confirm Password" required onkeyup="validatePassword()"><br>  
 <input type="text" name="toolname" placeholder="Tool Name" required><br>  
 <select name="experience" required>  
 <option value="" disabled selected>Select Experience</option>  
 <option value="JUNIOR">Junior</option>  
 <option value="MID">Mid</option>  
 <option value="SENIOR">Senior</option>  
 </select><br>  
 <input type="submit" value="Sign up">  
 </form>  
 </div>  
</div>  
  
<script>  
 function validatePassword() {  
 var password = *document*.getElementById("password");  
 var confirm\_password = *document*.getElementById("confirm\_password");  
  
 if (password.value !== confirm\_password.value) {  
 confirm\_password.classList.add("password-error");  
 return false;  
 } else {  
 confirm\_password.classList.remove("password-error");  
 return true;  
 }  
 }  
</script>  
  
</body>  
</html>

Opis:

* **<!DOCTYPE html>**: Deklaracja typu dokumentu HTML.
* **<html lang="en">**: Otwarcie tagu HTML z określeniem języka dokumentu.
* **<head>**: Sekcja zawierająca metadane dokumentu oraz odniesienia do zewnętrznych zasobów.
* **<meta charset="UTF-8">**: Określenie kodowania znaków.
* **<meta name="viewport" content="width=device-width, initial-scale=1.0">**: Ustawienia viewportu dla responsywności na urządzeniach mobilnych.
* **<title>Register as C# Developer</title>**: Tytuł strony.
* **<style>**: Sekcja zawierająca style CSS dla dokumentu.
* **body**: Stylizacja dla elementu **body**.
* **.container**: Stylizacja dla klasy **container**, określająca jego szerokość i pozycję.
* **h1**: Stylizacja dla nagłówka h1.
* **.form-container**: Stylizacja dla kontenera formularza rejestracji.
* **input[type="text"], input[type="password"], select**: Stylizacja dla pól tekstowych i wyboru w formularzu.
* **input[type="submit"]**: Stylizacja dla przycisku submit w formularzu.
* **.home-icon**: Stylizacja dla ikony "Home", która przekierowuje użytkownika do strony głównej.
* **.password-error**: Stylizacja dla pola potwierdzania hasła w przypadku błędnego wprowadzenia hasła.
* **</style>**: Zakończenie sekcji ze stylami.
* **</head>**: Zamknięcie sekcji head.
* **<body>**: Sekcja zawierająca zawartość strony.
* **<a href="index" class="home-icon">&#127968;</a>**: Ikona "Home", przekierowująca użytkownika do strony głównej.
* **<div class="container">**: Kontener dla zawartości strony.
* **<h1>Registration Page for C# Developer</h1>**: Nagłówek informujący o stronie rejestracji dla deweloperów w języku C#.
* **<form>**: Formularz rejestracji dewelopera w języku C#.
* **onsubmit="return validatePassword()"**: Wywołanie funkcji walidacji hasła podczas wysyłania formularza.
* **onkeyup="validatePassword()"**: Wywołanie funkcji walidacji hasła podczas wprowadzania hasła.
* **<script>**: Sekcja zawierająca skrypt JavaScript do walidacji hasła.
* **</body>**: Koniec sekcji body.
* **</html>**: Koniec dokumentu HTML.

3.1.6.register\_database.jsp:

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Register as Database Analyst</title>  
 <style>  
 body {  
 font-family: Arial, sans-serif;  
 background-color: #f0f0f0;  
 text-align: center;  
 display: flex;  
 justify-content: center;  
 align-items: center;  
 height: 100vh;  
 position: relative;  
 }  
  
 .container {  
 width: 50%;  
 }  
  
 h1 {  
 color: #333;  
 text-align: center;  
 margin-bottom: 20px;  
 }  
  
 .form-container {  
 width: 300px;  
 margin: 0 auto;  
 background-color: #fff;  
 padding: 20px;  
 border-radius: 5px;  
 box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);  
 }  
  
 input[type="text"],  
 input[type="password"],  
 select {  
 width: 100%;  
 padding: 10px;  
 margin: 10px 0;  
 border: 1px solid #ccc;  
 border-radius: 5px;  
 box-sizing: border-box;  
 }  
  
 input[type="submit"] {  
 width: 100%;  
 padding: 10px;  
 background-color: #4CAF50;  
 color: #fff;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 }  
  
 input[type="submit"]:hover {  
 background-color: #45a049;  
 }  
  
 .home-icon {  
 position: absolute;  
 top: 10px;  
 right: 10px;  
 font-size: 40px;  
 color: #333;  
 text-decoration: none;  
 }  
  
 .home-icon:hover {  
 color: #f00;  
 }  
  
 .password-error {  
 border: 1px solid red **!important**;  
 }  
  
 </style>  
</head>  
<body>  
<a href="index" class="home-icon">&#127968;</a>  
<div class="container">  
 <h1>Registration Page for Database Analyst</h1>  
 <div class="form-container">  
 <form action="/register\_database" method="post" onsubmit="return validatePassword()">  
 <input type="text" name="name" placeholder="Name" required><br>  
 <input type="text" name="surname" placeholder="Surname" required><br>  
 <input type="text" name="login" placeholder="Login" required><br>  
 <input type="password" name="password" id="password" placeholder="Password" required><br>  
 <input type="password" name="confirm\_password" id="confirm\_password" placeholder="Confirm Password" required onkeyup="validatePassword()"><br>  
 <input type="text" name="toolname" placeholder="Tool Name" required><br>  
 <select name="experience" required>  
 <option value="" disabled selected>Select Experience</option>  
 <option value="JUNIOR">Junior</option>  
 <option value="MID">Mid</option>  
 <option value="SENIOR">Senior</option>  
 </select><br>  
 <input type="submit" value="Sign up">  
 </form>  
 </div>  
</div>  
  
<script>  
 function validatePassword() {  
 var password = *document*.getElementById("password");  
 var confirm\_password = *document*.getElementById("confirm\_password");  
  
 if (password.value !== confirm\_password.value) {  
 confirm\_password.classList.add("password-error");  
 return false;  
 } else {  
 confirm\_password.classList.remove("password-error");  
 return true;  
 }  
 }  
</script>  
  
</body>  
</html>

Opis:

* **<!DOCTYPE html>**: Deklaracja typu dokumentu HTML.
* **<html lang="en">**: Otwarcie tagu HTML z określeniem języka dokumentu.
* **<head>**: Sekcja zawierająca metadane dokumentu oraz odniesienia do zewnętrznych zasobów.
* **<meta charset="UTF-8">**: Określenie kodowania znaków.
* **<meta name="viewport" content="width=device-width, initial-scale=1.0">**: Ustawienia viewportu dla responsywności na urządzeniach mobilnych.
* **<title>Register as Database Analyst</title>**: Tytuł strony.
* **<style>**: Sekcja zawierająca style CSS dla dokumentu.
* **body**: Stylizacja dla elementu **body**.
* **.container**: Stylizacja dla klasy **container**, określająca jego szerokość i pozycję.
* **h1**: Stylizacja dla nagłówka h1.
* **.form-container**: Stylizacja dla kontenera formularza rejestracji.
* **input[type="text"], input[type="password"], select**: Stylizacja dla pól tekstowych i wyboru w formularzu.
* **input[type="submit"]**: Stylizacja dla przycisku submit w formularzu.
* **.home-icon**: Stylizacja dla ikony "Home", która przekierowuje użytkownika do strony głównej.
* **.password-error**: Stylizacja dla pola potwierdzania hasła w przypadku błędnego wprowadzenia hasła.
* **</style>**: Zakończenie sekcji ze stylami.
* **</head>**: Zamknięcie sekcji head.
* **<body>**: Sekcja zawierająca zawartość strony.
* **<a href="index" class="home-icon">&#127968;</a>**: Ikona "Home", przekierowująca użytkownika do strony głównej.
* **<div class="container">**: Kontener dla zawartości strony.
* **<h1>Registration Page for Database Analyst</h1>**: Nagłówek informujący o stronie rejestracji

3.1.7.register\_frontend.jsp:

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Register as Front-end Developer</title>  
 <style>  
 body {  
 font-family: Arial, sans-serif;  
 background-color: #f0f0f0;  
 text-align: center;  
 display: flex;  
 justify-content: center;  
 align-items: center;  
 height: 100vh;  
 position: relative;  
 }  
  
 .container {  
 width: 50%;  
 }  
  
 h1 {  
 color: #333;  
 text-align: center;  
 margin-bottom: 20px;  
 }  
  
 .form-container {  
 width: 300px;  
 margin: 0 auto;  
 background-color: #fff;  
 padding: 20px;  
 border-radius: 5px;  
 box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);  
 }  
  
 input[type="text"],  
 input[type="password"],  
 select {  
 width: 100%;  
 padding: 10px;  
 margin: 10px 0;  
 border: 1px solid #ccc;  
 border-radius: 5px;  
 box-sizing: border-box;  
 }  
  
 input[type="submit"] {  
 width: 100%;  
 padding: 10px;  
 background-color: #4CAF50;  
 color: #fff;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 }  
  
 input[type="submit"]:hover {  
 background-color: #45a049;  
 }  
  
 .home-icon {  
 position: absolute;  
 top: 10px;  
 right: 10px;  
 font-size: 40px;  
 color: #333;  
 text-decoration: none;  
 }  
  
 .home-icon:hover {  
 color: #f00;  
 }  
  
 .password-error {  
 border: 1px solid red **!important**;  
 }  
  
 </style>  
</head>  
<body>  
<a href="index" class="home-icon">&#127968;</a>  
<div class="container">  
 <h1>Registration Page for Front-end Developer</h1>  
 <div class="form-container">  
 <form action="/register\_frontend" method="post" onsubmit="return validatePassword()">  
 <input type="text" name="name" placeholder="Name" required><br>  
 <input type="text" name="surname" placeholder="Surname" required><br>  
 <input type="text" name="login" placeholder="Login" required><br>  
 <input type="password" name="password" id="password" placeholder="Password" required><br>  
 <input type="password" name="confirm\_password" id="confirm\_password" placeholder="Confirm Password" required onkeyup="validatePassword()"><br>  
 <input type="text" name="toolname" placeholder="Tool Name" required><br>  
 <select name="experience" required>  
 <option value="" disabled selected>Select Experience</option>  
 <option value="JUNIOR">Junior</option>  
 <option value="MID">Mid</option>  
 <option value="SENIOR">Senior</option>  
 </select><br>  
 <input type="submit" value="Sign up">  
 </form>  
 </div>  
</div>  
  
<script>  
 function validatePassword() {  
 var password = *document*.getElementById("password");  
 var confirm\_password = *document*.getElementById("confirm\_password");  
  
 if (password.value !== confirm\_password.value) {  
 confirm\_password.classList.add("password-error");  
 return false;  
 } else {  
 confirm\_password.classList.remove("password-error");  
 return true;  
 }  
 }  
</script>  
  
</body>  
</html>

Opis:

* **<!DOCTYPE html>**: Deklaracja typu dokumentu HTML.
* **<html lang="en">**: Otwarcie tagu HTML z określeniem języka dokumentu.
* **<head>**: Sekcja zawierająca metadane dokumentu oraz odniesienia do zewnętrznych zasobów.
* **<meta charset="UTF-8">**: Określenie kodowania znaków.
* **<meta name="viewport" content="width=device-width, initial-scale=1.0">**: Ustawienia viewportu dla responsywności na urządzeniach mobilnych.
* **<title>Register as Front-end Developer</title>**: Tytuł strony.
* **<style>**: Sekcja zawierająca style CSS dla dokumentu.
* **body**: Stylizacja dla elementu **body**.
* **.container**: Stylizacja dla klasy **container**, określająca jego szerokość i pozycję.
* **h1**: Stylizacja dla nagłówka h1.
* **.form-container**: Stylizacja dla kontenera formularza rejestracji.
* **input[type="text"], input[type="password"], select**: Stylizacja dla pól tekstowych i wyboru w formularzu.
* **input[type="submit"]**: Stylizacja dla przycisku submit w formularzu.
* **.home-icon**: Stylizacja dla ikony "Home", która przekierowuje użytkownika do strony głównej.
* **.password-error**: Stylizacja dla pola potwierdzania hasła w przypadku błędnego wprowadzenia hasła.
* **</style>**: Zakończenie sekcji ze stylami.
* **</head>**: Zamknięcie sekcji head.
* **<body>**: Sekcja zawierająca zawartość strony.
* **<a href="index" class="home-icon">&#127968;</a>**: Ikona "Home", przekierowująca użytkownika do strony głównej.
* **<div class="container">**: Kontener dla zawartości strony.
* **<h1>Registration Page for Front-end Developer</h1>**: Nagłówek informujący o stronie rejestracji

3.1.8.register\_java.jsp:

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Register as Java Developer</title>  
 <style>  
 body {  
 font-family: Arial, sans-serif;  
 background-color: #f0f0f0;  
 text-align: center;  
 display: flex;  
 justify-content: center;  
 align-items: center;  
 height: 100vh;  
 position: relative;  
 }  
  
 .container {  
 width: 50%;  
 }  
  
 h1 {  
 color: #333;  
 text-align: center;  
 margin-bottom: 20px;  
 }  
  
 .form-container {  
 width: 300px;  
 margin: 0 auto;  
 background-color: #fff;  
 padding: 20px;  
 border-radius: 5px;  
 box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);  
 }  
  
 input[type="text"],  
 input[type="password"],  
 select {  
 width: 100%;  
 padding: 10px;  
 margin: 10px 0;  
 border: 1px solid #ccc;  
 border-radius: 5px;  
 box-sizing: border-box;  
 }  
  
 input[type="submit"] {  
 width: 100%;  
 padding: 10px;  
 background-color: #4CAF50;  
 color: #fff;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 }  
  
 input[type="submit"]:hover {  
 background-color: #45a049;  
 }  
  
 .home-icon {  
 position: absolute;  
 top: 10px;  
 right: 10px;  
 font-size: 40px;  
 color: #333;  
 text-decoration: none;  
 }  
  
 .home-icon:hover {  
 color: #f00;  
 }  
  
 .password-error {  
 border: 1px solid red **!important**;  
 }  
  
 </style>  
</head>  
<body>  
<a href="index" class="home-icon">&#127968;</a>  
<div class="container">  
 <h1>Registration Page for Python Developer</h1>  
 <div class="form-container">  
 <form action="/register\_java" method="post" onsubmit="return validatePassword()">  
 <input type="text" name="name" placeholder="Name" required><br>  
 <input type="text" name="surname" placeholder="Surname" required><br>  
 <input type="text" name="login" placeholder="Login" required><br>  
 <input type="password" name="password" id="password" placeholder="Password" required><br>  
 <input type="password" name="confirm\_password" id="confirm\_password" placeholder="Confirm Password" required onkeyup="validatePassword()"><br>  
 <input type="text" name="toolname" placeholder="Tool Name" required><br>  
 <select name="experience" required>  
 <option value="" disabled selected>Select Experience</option>  
 <option value="JUNIOR">Junior</option>  
 <option value="MID">Mid</option>  
 <option value="SENIOR">Senior</option>  
 </select><br>  
 <input type="submit" value="Sign up">  
 </form>  
 </div>  
</div>  
  
<script>  
 function validatePassword() {  
 var password = *document*.getElementById("password");  
 var confirm\_password = *document*.getElementById("confirm\_password");  
  
 if (password.value !== confirm\_password.value) {  
 confirm\_password.classList.add("password-error");  
 return false;  
 } else {  
 confirm\_password.classList.remove("password-error");  
 return true;  
 }  
 }  
</script>  
  
</body>  
</html>

Opis:

* **<!DOCTYPE html>**: Deklaracja typu dokumentu HTML.
* **<html lang="en">**: Otwarcie tagu HTML z określeniem języka dokumentu.
* **<head>**: Sekcja zawierająca metadane dokumentu oraz odniesienia do zewnętrznych zasobów.
* **<meta charset="UTF-8">**: Określenie kodowania znaków.
* **<meta name="viewport" content="width=device-width, initial-scale=1.0">**: Ustawienia viewportu dla responsywności na urządzeniach mobilnych.
* **<title>Register as Front-end Developer</title>**: Tytuł strony.
* **<style>**: Sekcja zawierająca style CSS dla dokumentu.
* **body**: Stylizacja dla elementu **body**.
* **.container**: Stylizacja dla klasy **container**, określająca jego szerokość i pozycję.
* **h1**: Stylizacja dla nagłówka h1.
* **.form-container**: Stylizacja dla kontenera formularza rejestracji.
* **input[type="text"], input[type="password"], select**: Stylizacja dla pól tekstowych i wyboru w formularzu.
* **input[type="submit"]**: Stylizacja dla przycisku submit w formularzu.
* **.home-icon**: Stylizacja dla ikony "Home", która przekierowuje użytkownika do strony głównej.
* **.password-error**: Stylizacja dla pola potwierdzania hasła w przypadku błędnego wprowadzenia hasła.
* **</style>**: Zakończenie sekcji ze stylami.
* **</head>**: Zamknięcie sekcji head.
* **<body>**: Sekcja zawierająca zawartość strony.
* **<a href="index" class="home-icon">&#127968;</a>**: Ikona "Home", przekierowująca użytkownika do strony głównej.
* **<div class="container">**: Kontener dla zawartości strony.
* **<h1>Registration Page for Front-end Developer</h1>**: Nagłówek informujący o stronie rejestracji

3.1.9.register\_python.jsp:

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Register as Python Developer</title>  
 <style>  
 body {  
 font-family: Arial, sans-serif;  
 background-color: #f0f0f0;  
 text-align: center;  
 display: flex;  
 justify-content: center;  
 align-items: center;  
 height: 100vh;  
 position: relative;  
 }  
  
 .container {  
 width: 50%;  
 }  
  
 h1 {  
 color: #333;  
 text-align: center;  
 margin-bottom: 20px;  
 }  
  
 .form-container {  
 width: 300px;  
 margin: 0 auto;  
 background-color: #fff;  
 padding: 20px;  
 border-radius: 5px;  
 box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);  
 }  
  
 input[type="text"],  
 input[type="password"],  
 select {  
 width: 100%;  
 padding: 10px;  
 margin: 10px 0;  
 border: 1px solid #ccc;  
 border-radius: 5px;  
 box-sizing: border-box;  
 }  
  
 input[type="submit"] {  
 width: 100%;  
 padding: 10px;  
 background-color: #4CAF50;  
 color: #fff;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 }  
  
 input[type="submit"]:hover {  
 background-color: #45a049;  
 }  
  
 .home-icon {  
 position: absolute;  
 top: 10px;  
 right: 10px;  
 font-size: 40px;  
 color: #333;  
 text-decoration: none;  
 }  
  
 .home-icon:hover {  
 color: #f00;  
 }  
  
 .password-error {  
 border: 1px solid red **!important**;  
 }  
  
 </style>  
</head>  
<body>  
<a href="index" class="home-icon">&#127968;</a>  
<div class="container">  
 <h1>Registration Page for Python Developer</h1>  
 <div class="form-container">  
 <form action="/register\_python" method="post" onsubmit="return validatePassword()">  
 <input type="text" name="name" placeholder="Name" required><br>  
 <input type="text" name="surname" placeholder="Surname" required><br>  
 <input type="text" name="login" placeholder="Login" required><br>  
 <input type="password" name="password" id="password" placeholder="Password" required><br>  
 <input type="password" name="confirm\_password" id="confirm\_password" placeholder="Confirm Password" required onkeyup="validatePassword()"><br>  
 <input type="text" name="toolname" placeholder="Tool Name" required><br>  
 <select name="experience" required>  
 <option value="" disabled selected>Select Experience</option>  
 <option value="JUNIOR">Junior</option>  
 <option value="MID">Mid</option>  
 <option value="SENIOR">Senior</option>  
 </select><br>  
 <input type="submit" value="Sign up">  
 </form>  
 </div>  
</div>  
  
<script>  
 function validatePassword() {  
 var password = *document*.getElementById("password");  
 var confirm\_password = *document*.getElementById("confirm\_password");  
  
 if (password.value !== confirm\_password.value) {  
 confirm\_password.classList.add("password-error");  
 return false;  
 } else {  
 confirm\_password.classList.remove("password-error");  
 return true;  
 }  
 }  
</script>  
  
</body>  
</html>

Opis:

* **<!DOCTYPE html>**: Deklaracja typu dokumentu HTML.
* **<html lang="en">**: Otwarcie tagu HTML z określeniem języka dokumentu.
* **<head>**: Sekcja zawierająca metadane dokumentu oraz odniesienia do zewnętrznych zasobów.
* **<meta charset="UTF-8">**: Określenie kodowania znaków.
* **<meta name="viewport" content="width=device-width, initial-scale=1.0">**: Ustawienia viewportu dla responsywności na urządzeniach mobilnych.
* **<title>Register as Python Developer</title>**: Tytuł strony.
* **<style>**: Sekcja zawierająca style CSS dla dokumentu.
* **body**: Stylizacja dla elementu **body**.
* **.container**: Stylizacja dla klasy **container**, określająca jego szerokość i pozycję.
* **h1**: Stylizacja dla nagłówka h1.
* **.form-container**: Stylizacja dla kontenera formularza rejestracji.
* **input[type="text"], input[type="password"], select**: Stylizacja dla pól tekstowych i wyboru w formularzu.
* **input[type="submit"]**: Stylizacja dla przycisku submit w formularzu.
* **.home-icon**: Stylizacja dla ikony "Home", która przekierowuje użytkownika do strony głównej.
* **.password-error**: Stylizacja dla pola potwierdzania hasła w przypadku błędnego wprowadzenia hasła.
* **</style>**: Zakończenie sekcji ze stylami.
* **</head>**: Zamknięcie sekcji head.
* **<body>**: Sekcja zawierająca zawartość strony.
* **<a href="index" class="home-icon">&#127968;</a>**: Ikona "Home", przekierowująca użytkownika do strony głównej.
* **<div class="container">**: Kontener dla zawartości strony.
* **<h1>Registration Page for Python Developer</h1>**: Nagłówek informujący o stronie rejestracji.

3.1.10.registration\_failure.jsp:

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Registration Failed</title>  
 <style>  
 body {  
 font-family: Arial, sans-serif;  
 background-color: #f0f0f0;  
 text-align: center;  
 display: flex;  
 justify-content: center;  
 align-items: center;  
 height: 100vh;  
 position: relative;  
 }  
  
 .container {  
 width: 50%;  
 }  
  
 h1 {  
 color: #333;  
 text-align: center;  
 margin-bottom: 20px;  
 }  
  
 p {  
 color: red;  
 margin-bottom: 20px;  
 }  
  
 .try-again-btn {  
 padding: 10px 20px;  
 background-color: #4CAF50;  
 color: white;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 text-decoration: none;  
 font-size: 1.2rem;  
 }  
  
 .try-again-btn:hover {  
 background-color: #45a049;  
 }  
  
 .home-icon {  
 position: absolute;  
 top: 10px;  
 right: 10px;  
 font-size: 40px;  
 color: #333;  
 text-decoration: none;  
 }  
  
 .home-icon:hover {  
 color: #f00;  
 }  
  
 </style>  
</head>  
<body>  
<a href="index" class="home-icon">&#127968;</a>  
<div class="container">  
 <h1>Registration Failed</h1>  
 <p>Login already used. Please try again with a different login.</p>  
 <a href="signup" class="try-again-btn">Try Again</a>  
</div>  
</body>  
</html>

Opis:

* **<!DOCTYPE html>**: Deklaruje typ dokumentu i wersję HTML.
* **<html lang="en">**: Rozpoczyna dokument HTML i określa język.
* **<head>**: Zawiera metadane i linki do zewnętrznych zasobów.
* **<meta charset="UTF-8">**: Ustawia kodowanie znaków na UTF-8.
* **<meta name="viewport" content="width=device-width, initial-scale=1.0">**: Konfiguruje widok dla responsywnej strony.
* **<title>Rejestracja nieudana</title>**: Ustawia tytuł dokumentu.
* **<style>**: Zawiera style CSS dla dokumentu.
* **body**: Style dla elementu body.
* **.container**: Style dla kontenera zawierającego komunikat o nieudanej rejestracji.
* **h1**: Style dla nagłówka informującego o niepowodzeniu rejestracji.
* **p**: Style dla akapitu wyświetlającego konkretną przyczynę niepowodzenia.
* **.try-again-btn**: Style dla przycisku pozwalającego użytkownikowi ponownie spróbować rejestracji z innym loginem.
* **.home-icon**: Style dla ikony powrotu do strony głównej.
* **</style>**: Zamyka blok ze stylami.
* **</head>**: Kończy sekcję nagłówka.
* **<body>**: Zawiera zawartość dokumentu.
* **<a href="index" class="home-icon">&#127968;</a>**: Link i stylizacja dla ikony powrotu do strony głównej.
* **<div class="container">**: Kontener dla komunikatu o nieudanej rejestracji.
* **<h1>Rejestracja nieudana</h1>**: Nagłówek informujący o niepowodzeniu rejestracji.
* **<p>Login już używany. Proszę spróbuj ponownie z innym loginem.</p>**: Akapit wyświetlający konkretną przyczynę niepowodzenia.
* **<a href="signup" class="try-again-btn">Spróbuj ponownie</a>**: Przycisk umożliwiający ponowną rejestrację z innym loginem.
* **</div>**: Zamyka kontener.
* **</body>**: Kończy sekcję ciała dokumentu.
* **</html>**: Kończy dokument HTML.

3.1.11.registration\_success.jsp:

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Registration Success</title>  
 <style>  
 body {  
 font-family: Arial, sans-serif;  
 background-color: #f0f0f0;  
 text-align: center;  
 display: flex;  
 justify-content: center;  
 align-items: center;  
 height: 100vh;  
 }  
  
 .container {  
 width: 50%;  
 }  
  
 p {  
 color: #333;  
 margin-bottom: 20px;  
 }  
  
 .home-link {  
 padding: 10px 20px;  
 background-color: #4CAF50;  
 color: white;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 text-decoration: none;  
 font-size: 1.2rem;  
 }  
  
 .home-link:hover {  
 background-color: #45a049;  
 }  
  
 </style>  
</head>  
<body>  
<div class="container">  
 <p>Now you can <a href="signin" class="home-link">sign in</a>!</p>  
</div>  
</body>  
</html>

Opis:

* **<!DOCTYPE html>**: Deklaruje typ dokumentu i wersję HTML.
* **<html lang="en">**: Rozpoczyna dokument HTML i określa język.
* **<head>**: Zawiera metadane i linki do zewnętrznych zasobów.
* **<meta charset="UTF-8">**: Ustawia kodowanie znaków na UTF-8.
* **<meta name="viewport" content="width=device-width, initial-scale=1.0">**: Konfiguruje widok dla responsywnej strony.
* **<title>Registration Success</title>**: Ustawia tytuł dokumentu.
* **<style>**: Zawiera style CSS dla dokumentu.
* **body**: Style dla elementu body.
* **.container**: Style dla kontenera zawierającego komunikat o sukcesie rejestracji.
* **p**: Style dla akapitu zawierającego komunikat o możliwości zalogowania się.
* **.home-link**: Style dla linku prowadzącego do strony logowania.
* **.home-link:hover**: Style dla linku po najechaniu na niego kursorem.
* **</style>**: Zamyka blok ze stylami.
* **</head>**: Kończy sekcję nagłówka.
* **<body>**: Zawiera zawartość dokumentu.
* **<div class="container">**: Kontener dla komunikatu o sukcesie rejestracji.
* **<p>Teraz możesz <a href="signin" class="home-link">zalogować się</a>!</p>**: Komunikat informujący o możliwości zalogowania się.
* **</div>**: Zamyka kontener.
* **</body>**: Kończy sekcję ciała dokumentu.
* **</html>**: Kończy dokument HTML.

3.1.12.signin.jsp:

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Sign in</title>  
 <style>  
 body {  
 font-family: Arial, sans-serif;  
 background-color: #f0f0f0;  
 text-align: center;  
 display: flex;  
 justify-content: center;  
 align-items: center;  
 height: 100vh;  
 }  
  
 .container {  
 width: 50%;  
 }  
  
 h1 {  
 color: #333;  
 text-align: center;  
 }  
  
 .form-container {  
 width: 300px;  
 margin: 0 auto;  
 background-color: #fff;  
 padding: 20px;  
 border-radius: 5px;  
 box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);  
 }  
  
 input[type="text"],  
 input[type="password"] {  
 width: 100%;  
 padding: 10px;  
 margin: 10px 0;  
 border: 1px solid #ccc;  
 border-radius: 5px;  
 box-sizing: border-box;  
 }  
  
 input[type="submit"] {  
 width: 100%;  
 padding: 10px;  
 background-color: #4CAF50;  
 color: #fff;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 }  
  
 input[type="submit"]:hover {  
 background-color: #45a049;  
 }  
  
 .home-icon {  
 position: absolute;  
 top: 10px;  
 right: 10px;  
 font-size: 40px;  
 color: #333;  
 text-decoration: none;  
 }  
  
 .home-icon:hover {  
 color: #f00;  
 }  
 </style>  
</head>  
<body>  
<a href="index" class="home-icon">&#127968;</a>  
<div class="container">  
 <h1>Welcome back!</h1>  
 <div class="form-container">  
 <form action="/login" method="post">  
 <input type="text" name="username" placeholder="Username" required><br>  
 <input type="password" name="password" placeholder="Password" required><br>  
 <input type="submit" value="Sign in">  
 </form>  
 </div>  
</div>  
</body>  
</html>

Opis:

* **<!DOCTYPE html>**: Deklaruje typ dokumentu i wersję HTML.
* **<html lang="en">**: Rozpoczyna dokument HTML i określa język.
* **<head>**: Zawiera metadane i linki do zewnętrznych zasobów.
* **<meta charset="UTF-8">**: Ustawia kodowanie znaków na UTF-8.
* **<meta name="viewport" content="width=device-width, initial-scale=1.0">**: Konfiguruje widok dla responsywnej strony.
* **<title>Sign in</title>**: Ustawia tytuł dokumentu.
* **<style>**: Zawiera style CSS dla dokumentu.
* **body**: Style dla elementu body.
* **.container**: Style dla kontenera zawierającego formularz logowania.
* **h1**: Style dla nagłówka witającego użytkownika.
* **.form-container**: Style dla kontenera formularza logowania.
* **input[type="text"], input[type="password"]**: Style dla pól tekstowych i haseł w formularzu.
* **input[type="submit"]**: Style dla przycisku zatwierdzania formularza.
* **.home-icon**: Style dla ikony powrotu do strony głównej.
* **</style>**: Zamyka blok ze stylami.
* **</head>**: Kończy sekcję nagłówka.
* **<body>**: Zawiera zawartość dokumentu.
* **<a href="index" class="home-icon">&#127968;</a>**: Ikona powrotu do strony głównej.
* **<div class="container">**: Kontener dla formularza logowania.
* **<h1>Witaj z powrotem!</h1>**: Nagłówek witający użytkownika.
* **<form action="/login" method="post">**: Formularz logowania.
* **<input type="text" name="username" placeholder="Nazwa użytkownika" required><br>**: Pole tekstowe na nazwę użytkownika.
* **<input type="password" name="password" placeholder="Hasło" required><br>**: Pole tekstowe na hasło.
* **<input type="submit" value="Zaloguj się">**: Przycisk logowania.
* **</form>**: Zamyka formularz logowania.
* **</div>**: Zamyka kontener.
* **</body>**: Kończy sekcję ciała dokumentu.
* **</html>**: Kończy dokument HTML.

3.1.13.signup.jsp:

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Register</title>  
 <style>  
 body {  
 font-family: Arial, sans-serif;  
 background-color: #f0f0f0;  
 text-align: center;  
 display: flex;  
 justify-content: center;  
 align-items: center;  
 height: 100vh;  
 position: relative;  
 }  
  
 .container {  
 width: 50%;  
 }  
  
 h1 {  
 color: #333;  
 text-align: center;  
 margin-bottom: 20px;  
 }  
  
 .form-container {  
 width: 300px;  
 margin: 0 auto;  
 background-color: #fff;  
 padding: 20px;  
 border-radius: 5px;  
 box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);  
 }  
  
 input[type="text"],  
 input[type="password"],  
 select {  
 width: 100%;  
 padding: 10px;  
 margin: 10px 0;  
 border: 1px solid #ccc;  
 border-radius: 5px;  
 box-sizing: border-box;  
 }  
  
 input[type="submit"],  
 button {  
 width: 100%;  
 padding: 10px;  
 background-color: #4CAF50;  
 color: #fff;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 }  
  
 input[type="submit"]:hover,  
 button:hover {  
 background-color: #45a049;  
 }  
  
 .home-icon {  
 position: absolute;  
 top: 10px;  
 right: 10px;  
 font-size: 40px;  
 color: #333;  
 text-decoration: none;  
 }  
  
 .home-icon:hover {  
 color: #f00;  
 }  
  
 </style>  
</head>  
<body>  
<a href="index" class="home-icon">&#127968;</a>  
<div class="container">  
 <h1>Welcome to Pay Slip Generator!</h1>  
 <h1>Choose your position</h1>  
 <div class="form-container">  
 <form id="registrationForm">  
 <select id="position" onchange="redirect()">  
 <option value="" disabled selected>Select Position</option>  
 <option value="java\_developer">Java Developer</option>  
 <option value="python\_developer">Python Developer</option>  
 <option value="csharp\_developer">C# Developer</option>  
 <option value="cpp\_developer">C++ Developer</option>  
 <option value="frontend\_developer">Front-end Developer</option>  
 <option value="database\_analyst">Database Analyst</option>  
 </select>  
 <button type="button" onclick="choosePosition()">OK</button>  
 </form>  
 </div>  
</div>  
  
<script>  
 function choosePosition() {  
 var positionSelect = *document*.getElementById("position");  
 var selectedValue = positionSelect.value;  
 var redirectUrl;  
  
 switch(selectedValue) {  
 case "java\_developer":  
 redirectUrl = "register\_java";  
 break;  
 case "python\_developer":  
 redirectUrl = "register\_python";  
 break;  
 case "csharp\_developer":  
 redirectUrl = "register\_csharp";  
 break;  
 case "cpp\_developer":  
 redirectUrl = "register\_cpp";  
 break;  
 case "frontend\_developer":  
 redirectUrl = "register\_frontend";  
 break;  
 case "database\_analyst":  
 redirectUrl = "register\_database";  
 break;  
 default:  
 break;  
 }  
  
 if (redirectUrl) {  
 *window*.location.href = redirectUrl;  
 }  
 }  
</script>  
  
</body>  
</html>

Opis:

* **<!DOCTYPE html>**: Deklaruje typ dokumentu i wersję HTML.
* **<html lang="en">**: Rozpoczyna dokument HTML i określa język.
* **<head>**: Zawiera metadane i odwołania do zewnętrznych zasobów.
* **<meta charset="UTF-8">**: Ustawia kodowanie znaków na UTF-8.
* **<meta name="viewport" content="width=device-width, initial-scale=1.0">**: Konfiguruje widok dla responsywnej strony.
* **<title>Register</title>**: Ustawia tytuł dokumentu.
* **<style>**: Zawiera style CSS dla dokumentu.
* **body**: Style dla elementu body.
* **.container**: Style dla kontenera zawierającego formularz rejestracyjny.
* **h1**: Style dla nagłówka witającego użytkownika.
* **.form-container**: Style dla kontenera formularza rejestracyjnego.
* **input[type="text"], input[type="password"]**: Style dla pól tekstowych i haseł w formularzu.
* **input[type="submit"], button**: Style dla przycisku zatwierdzania formularza.
* **.home-icon**: Style dla ikony powrotu do strony głównej.
* **</style>**: Zamyka blok ze stylami.
* **</head>**: Kończy sekcję nagłówka.
* **<body>**: Zawiera zawartość dokumentu.
* **<a href="index" class="home-icon">&#127968;</a>**: Ikona powrotu do strony głównej.
* **<div class="container">**: Kontener dla formularza rejestracyjnego.
* **<h1>Welcome to Pay Slip Generator!</h1>**: Nagłówek witający użytkownika.
* **<h1>Choose your position</h1>**: Nagłówek wybierania pozycji.
* **<form id="registrationForm">**: Formularz rejestracyjny.
* **<select id="position" onchange="redirect()">**: Lista rozwijana z pozycjami.
* **<option value="" disabled selected>Select Position</option>**: Domyślna opcja wyboru.
* **<option value="java\_developer">Java Developer</option>**: Opcja wyboru "Java Developer".
* **<option value="python\_developer">Python Developer</option>**: Opcja wyboru "Python Developer".
* **<option value="csharp\_developer">C# Developer</option>**: Opcja wyboru "C# Developer".
* **<option value="cpp\_developer">C++ Developer</option>**: Opcja wyboru "C++ Developer".
* **<option value="frontend\_developer">Front-end Developer</option>**: Opcja wyboru "Front-end Developer".
* **<option value="database\_analyst">Database Analyst</option>**: Opcja wyboru "Database Analyst".
* **<button type="button" onclick="choosePosition()">OK</button>**: Przycisk zatwierdzający wybór pozycji.
* **</form>**: Zamyka formularz rejestracyjny.
* **</div>**: Zamyka kontener.
* **</body>**: Kończy sekcję ciała dokumentu.
* **</html>**: Kończy dokument HTML.

3.2 Pliki konfiguracyjne

3.2.1.hibernate.cfg.xml:

<?xml version='1.0' encoding='utf-8'?>  
<!DOCTYPE hibernate-configuration PUBLIC  
 "-//Hibernate/Hibernate Configuration DTD//EN"  
 "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">  
<hibernate-configuration>  
 <session-factory>  
 <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/hibernateexs?serverTimezone=Europe/Warsaw</property>  
 <property name="hibernate.connection.driver\_class">com.mysql.jdbc.Driver</property>  
 <property name="hibernate.connection.username">root</property>  
 <property name="hibernate.connection.password"></property>  
 <property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>  
  
 <property name="hibernate.hbm2ddl.auto">update</property>  
  
 <property name="hibernate.show\_sql">true</property>  
 <property name="hibernate.format\_sql">true</property>  
  
 <mapping class="com.example.payslipgenerator.employees.Employee"/>  
 <mapping class="com.example.payslipgenerator.employees.CppDeveloper"/>  
 <mapping class="com.example.payslipgenerator.employees.CSharpDeveloper"/>  
 <mapping class="com.example.payslipgenerator.employees.JavaDeveloper"/>  
 <mapping class="com.example.payslipgenerator.employees.PythonDeveloper"/>  
 <mapping class="com.example.payslipgenerator.employees.FrontEndDeveloper"/>  
 <mapping class="com.example.payslipgenerator.employees.DatabaseAnalyst"/>  
 </session-factory>  
</hibernate-configuration>

Opis:

* **<?xml version='1.0' encoding='utf-8'?>**: Deklaruje wersję XML i kodowanie znaków.
* **<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN" "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">**: Deklaruje typ i lokalizację DTD (Document Type Definition) dla konfiguracji Hibernate.
* **<hibernate-configuration>**: Rozpoczyna sekcję konfiguracji Hibernate.
* **<session-factory>**: Definiuje fabrykę sesji dla Hibernate.
* **<!-- Pozostałe ustawienia konfiguracyjne Hibernate'a -->**: Komentarz o pozostałych ustawieniach konfiguracyjnych Hibernate'a.
* **<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/hibernateexs?serverTimezone=Europe/Warsaw</property>**: Określa URL do bazy danych MySQL.
* **<property name="hibernate.connection.driver\_class">com.mysql.jdbc.Driver</property>**: Określa klasę sterownika JDBC dla MySQL.
* **<property name="hibernate.connection.username">root</property>**: Określa nazwę użytkownika do logowania do bazy danych.
* **<property name="hibernate.connection.password"></property>**: Określa hasło do logowania do bazy danych.
* **<property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>**: Określa dialekt MySQL dla Hibernate.
* **<property name="hibernate.hbm2ddl.auto">update</property>**: Określa strategię aktualizacji schematu bazy danych (w tym przypadku aktualizacja).
* **<property name="hibernate.show\_sql">true</property>**: Ustawia flagę określającą, czy Hibernate ma wyświetlać wygenerowane SQL w konsoli.
* **<property name="hibernate.format\_sql">true</property>**: Ustawia flagę określającą, czy wyświetlany SQL ma być sformatowany.
* **<mapping class="com.example.payslipgenerator.employees.Employee"/>**: Mapuje klasę encji **Employee** na tabelę w bazie danych.
* **<mapping class="com.example.payslipgenerator.employees.CppDeveloper"/>**: Mapuje klasę encji **CppDeveloper** na tabelę w bazie danych.
* **<mapping class="com.example.payslipgenerator.employees.CSharpDeveloper"/>**: Mapuje klasę encji **CSharpDeveloper** na tabelę w bazie danych.
* **<mapping class="com.example.payslipgenerator.employees.JavaDeveloper"/>**: Mapuje klasę encji **JavaDeveloper** na tabelę w bazie danych.
* **<mapping class="com.example.payslipgenerator.employees.PythonDeveloper"/>**: Mapuje klasę encji **PythonDeveloper** na tabelę w bazie danych.
* **<mapping class="com.example.payslipgenerator.employees.FrontEndDeveloper"/>**: Mapuje klasę encji **FrontEndDeveloper** na tabelę w bazie danych.
* **<mapping class="com.example.payslipgenerator.employees.DatabaseAnalyst"/>**: Mapuje klasę encji **DatabaseAnalyst** na tabelę w bazie danych.
* **</session-factory>**: Kończy definicję fabryki sesji.
* **</hibernate-configuration>**: Kończy sekcję konfiguracji Hibernate.

3.2.2.application.properties:

spring.mvc.view.prefix=/WEB-INF/views/  
spring.mvc.view.suffix=.jsp

Opis:

* **spring.mvc.view.prefix=/WEB-INF/views/**: Ustawia prefiks ścieżki dla widoków. Oznacza to, że Spring MVC będzie szukał plików widoku w katalogu **/WEB-INF/views/**.
* **spring.mvc.view.suffix=.jsp**: Ustawia sufiks dla nazw plików widoku. W tym przypadku Spring MVC oczekuje, że pliki widoku będą miały rozszerzenie **.jsp**. Na przykład, jeśli kontroler zwraca nazwę **home**, Spring MVC automatycznie doda prefiks i sufiks, aby znaleźć widok **home.jsp** w katalogu **/WEB-INF/views/**.

3.3 Klasy springowe

3.3.1.RegoisterController:

package com.example.payslipgenerator;  
  
import com.example.payslipgenerator.employees.\*;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestParam;  
  
import static com.example.payslipgenerator.service.PaySlipGeneratorService.*checkLoginDuplicates*;  
  
@Controller  
public class RegisterController {  
  
 @PostMapping("/register\_python")  
 public String registerPythonDeveloper(@RequestParam String name,  
 @RequestParam String surname,  
 @RequestParam String login,  
 @RequestParam String password,  
 @RequestParam String toolname,  
 @RequestParam String experience) {  
  
 boolean isValid = *checkLoginDuplicates*(login);  
 if (!isValid) {  
 return "redirect:/registration\_failure";  
 } else {  
 PythonDeveloper pythonDeveloper = new PythonDeveloper();  
 pythonDeveloper.addDataToDb(name, surname, login, password, toolname, experience);  
  
 return "redirect:/registration\_success";  
 }  
 }  
  
 @PostMapping("/register\_java")  
 public String registerJavaDeveloper(@RequestParam String name,  
 @RequestParam String surname,  
 @RequestParam String login,  
 @RequestParam String password,  
 @RequestParam String toolname,  
 @RequestParam String experience) {  
  
 boolean isValid = *checkLoginDuplicates*(login);  
 if (!isValid) {  
 return "redirect:/registration\_failure";  
 } else {  
 JavaDeveloper javaDeveloper = new JavaDeveloper();  
 javaDeveloper.addDataToDb(name, surname, login, password, toolname, experience);  
  
 return "redirect:/registration\_success";  
 }  
 }  
  
 @PostMapping("/register\_frontend")  
 public String registerFrontEndDeveloper(@RequestParam String name,  
 @RequestParam String surname,  
 @RequestParam String login,  
 @RequestParam String password,  
 @RequestParam String toolname,  
 @RequestParam String experience) {  
  
 boolean isValid = *checkLoginDuplicates*(login);  
 if (!isValid) {  
 return "redirect:/registration\_failure";  
 } else {  
 FrontEndDeveloper frontEndDeveloper = new FrontEndDeveloper();  
 frontEndDeveloper.addDataToDb(name, surname, login, password, toolname, experience);  
  
 return "redirect:/registration\_success";  
 }  
 }  
  
 @PostMapping("/register\_database")  
 public String registerDatabaseDeveloper(@RequestParam String name,  
 @RequestParam String surname,  
 @RequestParam String login,  
 @RequestParam String password,  
 @RequestParam String toolname,  
 @RequestParam String experience) {  
  
 boolean isValid = *checkLoginDuplicates*(login);  
 if (!isValid) {  
 return "redirect:/registration\_failure";  
 } else {  
 DatabaseAnalyst databaseAnalyst = new DatabaseAnalyst();  
 databaseAnalyst.addDataToDb(name, surname, login, password, toolname, experience);  
  
 return "redirect:/registration\_success";  
 }  
 }  
  
 @PostMapping("/register\_csharp")  
 public String registerCsharpDeveloper(@RequestParam String name,  
 @RequestParam String surname,  
 @RequestParam String login,  
 @RequestParam String password,  
 @RequestParam String toolname,  
 @RequestParam String experience) {  
  
 boolean isValid = *checkLoginDuplicates*(login);  
 if (!isValid) {  
 return "redirect:/registration\_failure";  
 } else {  
 CSharpDeveloper cSharpDeveloper = new CSharpDeveloper();  
 cSharpDeveloper.addDataToDb(name, surname, login, password, toolname, experience);  
  
 return "redirect:/registration\_success";  
 }  
 }  
  
 @PostMapping("/register\_cpp")  
 public String registerCppDeveloper(@RequestParam String name,  
 @RequestParam String surname,  
 @RequestParam String login,  
 @RequestParam String password,  
 @RequestParam String toolname,  
 @RequestParam String experience) {  
  
 boolean isValid = *checkLoginDuplicates*(login);  
 if (!isValid) {  
 return "redirect:/registration\_failure";  
 } else {  
 CppDeveloper cppDeveloper = new CppDeveloper();  
 cppDeveloper.addDataToDb(name, surname, login, password, toolname, experience);  
  
 return "redirect:/registration\_success";  
 }  
 }  
}

**Opis:**

* **package com.example.payslipgenerator;**: Pakiet, w którym znajduje się klasa RegisterController.
* **import com.example.payslipgenerator.employees.\*;**: Importuje klasy z pakietu employees.
* **import org.springframework.stereotype.Controller;**: Importuje klasę Controller z frameworka Spring.
* **import org.springframework.web.bind.annotation.PostMapping;**: Importuje adnotację PostMapping z frameworka Spring.
* **import org.springframework.web.bind.annotation.RequestParam;**: Importuje adnotację RequestParam z frameworka Spring.
* **import static com.example.payslipgenerator.service.PaySlipGeneratorService.checkLoginDuplicates;**: Importuje statyczną metodę checkLoginDuplicates z klasy PaySlipGeneratorService.

Klasa RegisterController:

* Klasa obsługująca żądania związane z rejestracją różnych typów pracowników.
* Oznaczona adnotacją **@Controller**, co oznacza, że jest kontrolerem w Spring MVC.
* Posiada wiele metod obsługujących różne rodzaje pracowników, takie jak Python Developer, Java Developer, itp.
* Każda metoda obsługuje żądania POST przesyłane z formularzy rejestracyjnych dla odpowiedniego typu pracownika.

Metody:

1. **registerPythonDeveloper**: Obsługuje rejestrację Python Developerów.
2. **registerJavaDeveloper**: Obsługuje rejestrację Java Developerów.
3. **registerFrontEndDeveloper**: Obsługuje rejestrację Front-end Developerów.
4. **registerDatabaseDeveloper**: Obsługuje rejestrację Database Analystów.
5. **registerCsharpDeveloper**: Obsługuje rejestrację C# Developerów.
6. **registerCppDeveloper**: Obsługuje rejestrację C++ Developerów.

Parametry Metod:

* **@RequestParam String name**: Imię nowego pracownika.
* **@RequestParam String surname**: Nazwisko nowego pracownika.
* **@RequestParam String login**: Login nowego pracownika.
* **@RequestParam String password**: Hasło nowego pracownika.
* **@RequestParam String toolname**: Narzędzia używane przez pracownika.
* **@RequestParam String experience**: Doświadczenie pracownika.

Funkcje:

* Sprawdza duplikaty loginów przy użyciu metody **checkLoginDuplicates**.
* Tworzy nowe instancje odpowiednich klas pracowników.
* Dodaje dane pracownika do bazy danych.
* Przekierowuje użytkownika do odpowiednich stron w przypadku powodzenia lub niepowodzenia rejestracji.

3.3.2. PaySlipGeneratorController:

package com.example.payslipgenerator;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RequestMethod;  
  
@Controller  
public class PaySlipGeneratorController {  
  
 @RequestMapping(value = {"/", "/index"}, method = RequestMethod.*GET*)  
 public String getIndex() {  
 return "index";  
 }  
  
 @RequestMapping(value = "/signin", method = RequestMethod.*GET*)  
 public String getSignInPage() {  
 return "signin";  
 }  
  
 @RequestMapping(value = "/signup", method = RequestMethod.*GET*)  
 public String getSignUpPage() {  
 return "signup";  
 }  
  
 @RequestMapping(value = "/register\_cpp", method = RequestMethod.*GET*)  
 public String getCppDeveloperRegistrationPage() {  
 return "register\_cpp";  
 }  
  
 @RequestMapping(value = "/register\_csharp", method = RequestMethod.*GET*)  
 public String getCsharpDeveloperRegistrationPage() {  
 return "register\_csharp";  
 }  
  
 @RequestMapping(value = "/register\_database", method = RequestMethod.*GET*)  
 public String getDatabaseDeveloperRegistrationPage() {  
 return "register\_database";  
 }  
  
 @RequestMapping(value = "/register\_frontend", method = RequestMethod.*GET*)  
 public String getFrontendDeveloperRegistrationPage() {  
 return "register\_frontend";  
 }  
  
 @RequestMapping(value = "/register\_java", method = RequestMethod.*GET*)  
 public String getJavaDeveloperRegistrationPage() {  
 return "register\_java";  
 }  
  
 @RequestMapping(value = "/register\_python", method = RequestMethod.*GET*)  
 public String getPythonDeveloperRegistrationPage() {  
 return "register\_python";  
 }  
  
 @RequestMapping(value = "/notSignedIn", method = RequestMethod.*GET*)  
 public String getNotSignedIn() {  
 return "notSignedIn";  
 }  
  
 @RequestMapping(value = "/loggedIn", method = RequestMethod.*GET*)  
 public String getSignedIn() {  
 return "loggedIn";  
 }  
  
 @RequestMapping(value = "/registration\_success", method = RequestMethod.*GET*)  
 public String getRegistrationSuccess() {  
 return "registration\_success";  
 }  
  
 @RequestMapping(value = "/registration\_failure", method = RequestMethod.*GET*)  
 public String getRegistrationFailure() {  
 return "registration\_failure";  
 }  
}

**Opis:**

* Klasa **PaySlipGeneratorController** jest kontrolerem obsługującym żądania związane z generowaniem płac.
* Oznaczona adnotacją **@Controller**, co oznacza, że jest kontrolerem w Spring MVC.

Metody:

1. **getIndex**: Zwraca stronę główną aplikacji.
2. **getSignInPage**: Zwraca stronę logowania.
3. **getSignUpPage**: Zwraca stronę rejestracji.
4. **getCppDeveloperRegistrationPage**: Zwraca stronę rejestracji dla C++ Developerów.
5. **getCsharpDeveloperRegistrationPage**: Zwraca stronę rejestracji dla C# Developerów.
6. **getDatabaseDeveloperRegistrationPage**: Zwraca stronę rejestracji dla Database Analystów.
7. **getFrontendDeveloperRegistrationPage**: Zwraca stronę rejestracji dla Front-end Developerów.
8. **getJavaDeveloperRegistrationPage**: Zwraca stronę rejestracji dla Java Developerów.
9. **getPythonDeveloperRegistrationPage**: Zwraca stronę rejestracji dla Python Developerów.
10. **getNotSignedIn**: Zwraca stronę informującą o braku zalogowania.
11. **getSignedIn**: Zwraca stronę informującą o pomyślnym zalogowaniu.
12. **getRegistrationSuccess**: Zwraca stronę informującą o pomyślnym zarejestrowaniu użytkownika.
13. **getRegistrationFailure**: Zwraca stronę informującą o niepowodzeniu rejestracji użytkownika.

Adnotacje:

* **@RequestMapping**: Określa adres URL, pod którym metoda obsługuje żądanie.
* **value**: Określa adres URL, pod którym metoda jest wywoływana.
* **method**: Określa metodę HTTP używaną do obsługi żądania.

Funkcje:

* Każda metoda obsługuje żądanie typu GET na określonym adresie URL i zwraca nazwę widoku, który ma zostać wyświetlony.

3.3.3. PaySlipGeneratorApplication:

package com.example.payslipgenerator;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class PaySlipGeneratorApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(PaySlipGeneratorApplication.class, args);  
 }  
  
}

**Opis:**

* Klasa **PaySlipGeneratorApplication** jest główną klasą aplikacji.
* Oznaczona adnotacją **@SpringBootApplication**, która agreguje wiele adnotacji, w tym **@Configuration**, **@EnableAutoConfiguration** i **@ComponentScan**, co pozwala na automatyczną konfigurację aplikacji Spring Boot.
* Metoda **main** jest punktem wejścia do aplikacji. Uruchamia aplikację Spring Boot, która inicjuje kontekst aplikacji i uruchamia serwer.

Adnotacje:

* **@SpringBootApplication**: Oznacza, że ​​klasa jest klasą konfiguracyjną Spring Boot.

Metody:

1. **main**: Metoda główna aplikacji, która uruchamia aplikację Spring Boot.

Funkcje:

* Metoda **main** inicjuje aplikację Spring Boot, uruchamiając kontekst aplikacji i startując serwer.

3.3.4. LoginController:

package com.example.payslipgenerator;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestParam;  
  
import static com.example.payslipgenerator.service.PaySlipGeneratorService.*checkSignIn*;  
  
@Controller  
public class LoginController {  
  
 @PostMapping("/login")  
 public String login(@RequestParam("username") String username,  
 @RequestParam("password") String password,  
 Model model) {  
 boolean loginSuccessful = *checkSignIn*(username, password);  
 if (loginSuccessful) {  
 model.addAttribute("username", username);  
 return "redirect:/loggedIn?username=" + username;  
 } else {  
 return "redirect:/notSignedIn";  
 }  
 }  
}

**Opis:**

* Klasa **LoginController** jest kontrolerem obsługującym proces logowania użytkownika.
* Oznaczona adnotacją **@Controller**, co oznacza, że jest to komponent Springa, który obsługuje żądania HTTP.
* Metoda **login** jest odpowiedzialna za przetwarzanie żądania logowania wysłanego przez formularz.

Adnotacje:

* **@Controller**: Oznacza, że ​​klasa jest kontrolerem, który obsługuje żądania HTTP.

Metody:

1. **login**: Metoda obsługująca żądanie logowania. Przyjmuje nazwę użytkownika, hasło oraz model Springa.
   * **@PostMapping("/login")**: Oznacza, że metoda obsługuje żądania typu POST na ścieżce "/login".
   * **@RequestParam("username")**: Oznacza, że wartość parametru "username" z formularza zostanie przypisana do parametru metody **username**.
   * **@RequestParam("password")**: Oznacza, że wartość parametru "password" z formularza zostanie przypisana do parametru metody **password**.
   * **Model model**: Parametr modelu Springa, który może być wykorzystany do przekazywania danych do widoku.
   * Sprawdza poprawność danych logowania wywołując metodę **checkSignIn** z klasy **PaySlipGeneratorService**.
   * Jeśli logowanie jest udane, dodaje atrybut **username** do modelu i przekierowuje użytkownika na stronę **/loggedIn**.
   * W przeciwnym razie przekierowuje użytkownika na stronę **/notSignedIn**.

3.4 Klasy service

3.4.1. Main:

package com.example.payslipgenerator.service;  
  
import static com.example.payslipgenerator.service.PaySlipGeneratorService.*setDataInDb*;  
  
public class Main {  
  
 public static void main(String[] args) {  
  
 *setDataInDb*();  
  
 }  
  
}

**Opis:**

* Klasa **Main** jest klasą główną programu.
* Zawiera metodę **main**, która jest punktem wejścia do programu.
* Metoda **main** wywołuje metodę **setDataInDb** z klasy **PaySlipGeneratorService** w celu ustawienia danych w bazie danych.

Metody:

1. **main**: Metoda **main** jest punktem wejścia do programu.
   * Jest oznaczona adnotacją **public static void**, co oznacza, że jest dostępna do wywołania z dowolnego miejsca w programie i nie zwraca żadnej wartości.
   * Przyjmuje tablicę argumentów typu **String[]**, jednak nie jest wykorzystywana w tej aplikacji.
   * Wywołuje metodę **setDataInDb()** z klasy **PaySlipGeneratorService**, aby ustawić dane w bazie danych.

3.4.2. PaySlipGeneratorService:

package com.example.payslipgenerator.service;  
  
import com.example.payslipgenerator.employees.\*;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
import org.hibernate.cfg.Configuration;  
import org.hibernate.query.Query;  
  
import java.util.List;  
import java.util.Optional;  
  
@Data  
@AllArgsConstructor  
public class PaySlipGeneratorService {  
  
 public static void setDataInDb() {  
 SessionFactory sessionFactory = new Configuration()  
 .configure()  
 .buildSessionFactory();  
 try (Session session = sessionFactory.openSession()) {  
 Transaction transaction = session.beginTransaction();  
  
 JavaDeveloper javaJunior1 = new JavaDeveloper(null, "John", "Doe", "login", "password", "IntelliJ", Experience.*JUNIOR*);  
 session.save(javaJunior1);  
  
 JavaDeveloper javaJunior2 = new JavaDeveloper(null, "Alice", "Johnson", "login1", "password1", "NetBeans", Experience.*JUNIOR*);  
 session.save(javaJunior2);  
  
 JavaDeveloper javaMid1 = new JavaDeveloper(null, "Jane", "Smith", "login2", "password2", "Eclipse", Experience.*MID*);  
 session.save(javaMid1);  
  
 JavaDeveloper javaMid2 = new JavaDeveloper(null, "Bob", "Brown", "login3", "password3", "VS Code", Experience.*MID*);  
 session.save(javaMid2);  
  
 CSharpDeveloper cSharpJunior1 = new CSharpDeveloper(null, "Mark", "Davis", "login4", "password4", "Visual Studio", Experience.*JUNIOR*);  
 session.save(cSharpJunior1);  
  
 CSharpDeveloper cSharpMid1 = new CSharpDeveloper(null, "Emma", "Taylor", "login5", "password5", "Visual Studio Code", Experience.*MID*);  
 session.save(cSharpMid1);  
  
 CppDeveloper cppMid1 = new CppDeveloper(null, "David", "Lee", "login6", "password6", "CLion", Experience.*MID*);  
 session.save(cppMid1);  
  
 FrontEndDeveloper frontEndMid1 = new FrontEndDeveloper(null, "Olivia", "White", "login7", "password7", "Sublime Text", Experience.*MID*);  
 session.save(frontEndMid1);  
  
 DatabaseAnalyst dbAnalystMid1 = new DatabaseAnalyst(null, "Liam", "Harris", "login8", "password8", "MySQL Workbench", Experience.*MID*);  
 session.save(dbAnalystMid1);  
  
 PythonDeveloper pythonDeveloper1 = new PythonDeveloper(null, "James", "Bond", "login9", "password9", "PyCharm", Experience.*SENIOR*);  
 session.save(pythonDeveloper1);  
  
 transaction.commit();  
 session.close();  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 sessionFactory.close();  
 }  
 }  
  
 public static Employee generatePaySlipForEmployee(String login) {  
 SessionFactory sessionFactory = new Configuration()  
 .configure()  
 .buildSessionFactory();  
 try (Session session = sessionFactory.openSession()) {  
 Transaction transaction = session.beginTransaction();  
   
 Query<JavaDeveloper> javaQuery = session.createQuery("FROM JavaDeveloper WHERE login = :login", JavaDeveloper.class);  
 javaQuery.setParameter("login", login);  
 JavaDeveloper javaDeveloper = javaQuery.uniqueResult();  
 if (javaDeveloper != null) {  
 return javaDeveloper;  
 }  
   
 Query<CppDeveloper> cppQuery = session.createQuery("FROM CppDeveloper WHERE login = :login", CppDeveloper.class);  
 cppQuery.setParameter("login", login);  
 CppDeveloper cppDeveloper = cppQuery.uniqueResult();  
 if (cppDeveloper != null) {  
 return cppDeveloper;  
 }  
   
 Query<CSharpDeveloper> csharpQuery = session.createQuery("FROM CSharpDeveloper WHERE login = :login", CSharpDeveloper.class);  
 csharpQuery.setParameter("login", login);  
 CSharpDeveloper csharpDeveloper = csharpQuery.uniqueResult();  
 if (csharpDeveloper != null) {  
 return csharpDeveloper;  
 }  
   
 Query<DatabaseAnalyst> dbQuery = session.createQuery("FROM DatabaseAnalyst WHERE login = :login", DatabaseAnalyst.class);  
 dbQuery.setParameter("login", login);  
 DatabaseAnalyst dbAnalyst = dbQuery.uniqueResult();  
 if (dbAnalyst != null) {  
 return dbAnalyst;  
 }  
   
 Query<FrontEndDeveloper> frontEndQuery = session.createQuery("FROM FrontEndDeveloper WHERE login = :login", FrontEndDeveloper.class);  
 frontEndQuery.setParameter("login", login);  
 FrontEndDeveloper frontEndDeveloper = frontEndQuery.uniqueResult();  
 if (frontEndDeveloper != null) {  
 return frontEndDeveloper;  
 }  
   
 Query<PythonDeveloper> pythonQuery = session.createQuery("FROM PythonDeveloper WHERE login = :login", PythonDeveloper.class);  
 pythonQuery.setParameter("login", login);  
 PythonDeveloper pythonDeveloper = pythonQuery.uniqueResult();  
 if (pythonDeveloper != null) {  
 return pythonDeveloper;  
 }  
  
 transaction.commit();  
 session.close();  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 sessionFactory.close();  
 }  
  
 return null;  
 }  
  
 public static boolean checkSignIn(String username, String password) {  
 SessionFactory sessionFactory = new Configuration()  
 .configure()  
 .buildSessionFactory();  
 try (Session session = sessionFactory.openSession()) {  
 Transaction transaction = session.beginTransaction();  
  
 Query<Employee> query = session.createQuery("FROM Employee");  
 List<Employee> resultList = query.getResultList();  
 Optional<Employee> user = resultList.stream().filter(e -> e.getLogin().equals(username) && e.getPassword().equals(password)).findFirst();  
 transaction.commit();  
  
 if (user.isEmpty()) {  
 return false;  
 } else {  
 return true;  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 return false;  
 } finally {  
 sessionFactory.close();  
 }  
 }  
  
 public static boolean checkLoginDuplicates(String login) {  
 SessionFactory sessionFactory = new Configuration()  
 .configure()  
 .buildSessionFactory();  
 try (Session session = sessionFactory.openSession()) {  
 Transaction transaction = session.beginTransaction();  
   
 Query<JavaDeveloper> javaQuery = session.createQuery("FROM JavaDeveloper WHERE login = :login", JavaDeveloper.class);  
 javaQuery.setParameter("login", login);  
 JavaDeveloper javaDeveloper = javaQuery.uniqueResult();  
 if (javaDeveloper != null) {  
 return false;  
 }  
   
 Query<CppDeveloper> cppQuery = session.createQuery("FROM CppDeveloper WHERE login = :login", CppDeveloper.class);  
 cppQuery.setParameter("login", login);  
 CppDeveloper cppDeveloper = cppQuery.uniqueResult();  
 if (cppDeveloper != null) {  
 return false;  
 }  
   
 Query<CSharpDeveloper> csharpQuery = session.createQuery("FROM CSharpDeveloper WHERE login = :login", CSharpDeveloper.class);  
 csharpQuery.setParameter("login", login);  
 CSharpDeveloper csharpDeveloper = csharpQuery.uniqueResult();  
 if (csharpDeveloper != null) {  
 return false;  
 }  
   
 Query<DatabaseAnalyst> dbQuery = session.createQuery("FROM DatabaseAnalyst WHERE login = :login", DatabaseAnalyst.class);  
 dbQuery.setParameter("login", login);  
 DatabaseAnalyst dbAnalyst = dbQuery.uniqueResult();  
 if (dbAnalyst != null) {  
 return false;  
 }  
   
 Query<FrontEndDeveloper> frontEndQuery = session.createQuery("FROM FrontEndDeveloper WHERE login = :login", FrontEndDeveloper.class);  
 frontEndQuery.setParameter("login", login);  
 FrontEndDeveloper frontEndDeveloper = frontEndQuery.uniqueResult();  
 if (frontEndDeveloper != null) {  
 return false;  
 }  
   
 Query<PythonDeveloper> pythonQuery = session.createQuery("FROM PythonDeveloper WHERE login = :login", PythonDeveloper.class);  
 pythonQuery.setParameter("login", login);  
 PythonDeveloper pythonDeveloper = pythonQuery.uniqueResult();  
 if (pythonDeveloper != null) {  
 return false;  
 }  
  
 transaction.commit();  
 session.close();  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 sessionFactory.close();  
 }  
  
 return true;  
 }  
  
}

**Opis:**

Klasa **PaySlipGeneratorService** jest odpowiedzialna za obsługę logiki biznesowej związanej z generowaniem wypłat dla pracowników oraz zarządzaniem bazą danych. W ramach tej klasy znajdują się metody umożliwiające inicjowanie danych testowych w bazie, generowanie wypłat dla pracowników na podstawie ich danych, sprawdzanie poprawności logowania oraz sprawdzanie unikalności loginów pracowników.

Metody:

1. **setDataInDb()**:
   * Metoda statyczna, która inicjuje połączenie z bazą danych, ustawia dane testowe dla różnych typów pracowników i zapisuje je do bazy. Dane testowe obejmują informacje takie jak imię, nazwisko, login, hasło, narzędzie pracy oraz doświadczenie zawodowe.
   * Wykorzystuje obiekty klas pracowników (np. **JavaDeveloper**, **CppDeveloper**) do utworzenia danych testowych.
   * Po zakończeniu operacji zamyka sesję i zamyka fabrykę sesji Hibernate'a.
2. **generatePaySlipForEmployee(String login)**:
   * Metoda statyczna, która generuje wypłatę dla pracownika na podstawie podanego loginu. Na podstawie loginu pobiera dane pracownika z bazy danych.
   * Wykorzystuje zapytania Hibernate'a do pobrania danych pracownika na podstawie loginu.
   * Zwraca obiekt klasy **Employee**, która jest klasą nadrzędną dla wszystkich rodzajów pracowników.
3. **checkSignIn(String username, String password)**:
   * Metoda statyczna, która sprawdza, czy użytkownik może się zalogować na podstawie podanej nazwy użytkownika i hasła. Sprawdza, czy podane dane logowania są zgodne z danymi w bazie.
   * Pobiera wszystkich użytkowników z bazy danych, a następnie sprawdza, czy podana nazwa użytkownika i hasło pasują do któregoś z użytkowników.
   * Zwraca **true**, jeśli logowanie powiodło się, w przeciwnym razie zwraca **false**.
4. **checkLoginDuplicates(String login)**:
   * Metoda statyczna, która sprawdza, czy podany login już istnieje w bazie danych. Sprawdza unikalność loginu w bazie, aby zapobiec duplikatom.
   * Wykorzystuje zapytania Hibernate'a, aby sprawdzić, czy istnieje użytkownik o podanym loginie dla różnych typów pracowników.
   * Zwraca **true**, jeśli login jest unikatowy, w przeciwnym razie zwraca **false**.

3.5. Klasy encje oraz powiązane z nimi interfejsy oraz enumy

3.5.1. Employee:

package com.example.payslipgenerator.employees;  
  
import lombok.Data;  
  
import javax.persistence.\*;  
  
@Entity  
@Data  
@Table(name = "employees")  
@Inheritance(strategy = InheritanceType.*TABLE\_PER\_CLASS*)  
public abstract class Employee {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*SEQUENCE*)  
 private Long id;  
  
 @Column(name = "name")  
 private String name;  
 @Column(name = "surname")  
 private String surname;  
 @Column(name = "login")  
 private String login;  
 @Column(name = "password")  
 private String password;  
  
 public Employee(Long id, String name, String surname, String login, String password) {  
 this.id = id;  
 this.name = name;  
 this.surname = surname;  
 this.login = login;  
 this.password = password;  
 }  
  
 public Employee() {  
  
 }  
  
 public abstract String[] generatePaySlip();  
}

Opis:

* **@Entity**: Oznacza klasę jako encję, co umożliwia mapowanie jej pól do kolumn w bazie danych.
* **@Data**: Lombokowa adnotacja generująca metody **toString()**, **equals()**, **hashCode()**, **getter**y i **setter**y dla pól klasy.
* **@Table(name = "employees")**: Określa nazwę tabeli w bazie danych, do której zostanie zmapowana ta encja.
* **@Inheritance(strategy = InheritanceType.TABLE\_PER\_CLASS)**: Określa strategię dziedziczenia, w tym przypadku każda podklasa będzie miała swoją własną tabelę w bazie danych.
* **@Id**: Oznacza pole jako identyfikator (klucz główny) encji.
* **@GeneratedValue(strategy = GenerationType.SEQUENCE)**: Określa, że wartość identyfikatora będzie generowana automatycznie przy użyciu sekwencji.
* **@Column(name = "name")**, **@Column(name = "surname")**, **@Column(name = "login")**, **@Column(name = "password")**: Określają nazwy kolumn w bazie danych, do których zostaną zmapowane odpowiednie pola klasy.
* **abstract class Employee**: Jest to klasa abstrakcyjna, która będzie służyć jako nadklasa dla wszystkich rodzajów pracowników.
* Konstruktor **Employee(Long id, String name, String surname, String login, String password)**: Inicjalizuje obiekt klasy Employee z podanymi wartościami.
* Pusta metoda **Employee()**: Jest to domyślny konstruktor bezargumentowy.
* **public abstract String[] generatePaySlip()**: Jest to abstrakcyjna metoda, która zostanie zaimplementowana w klasach podrzędnych. Ma za zadanie generować wypłatę dla danego pracownika.

Początek formularza

3.5.2. DataToDbHandler (interface):

package com.example.payslipgenerator.employees;  
  
public interface DataToDbHandler {  
  
 void addDataToDb(String name, String surname, String login, String password, String toolName, String experience);  
  
}

Opis:

* **interface DataToDbHandler**: Interfejs jest deklaracją metod, które muszą być zaimplementowane przez klasy implementujące ten interfejs.
* **void addDataToDb(String name, String surname, String login, String password, String toolName, String experience)**: Jest to metoda, która będzie odpowiedzialna za dodawanie danych do bazy danych. Parametry metody to informacje o pracowniku, które będą dodawane do bazy danych, takie jak imię, nazwisko, login, hasło, narzędzie (np. edytor kodu), oraz doświadczenie pracownika. Implementacje tego interfejsu będą miały własną logikę dodawania danych do bazy danych w zależności od rodzaju pracownika.

3.5.3. Experience (enum):

package com.example.payslipgenerator.employees;  
  
public enum Experience {  
 *JUNIOR*,  
 *MID*,  
 *SENIOR*}

Opis:

* **enum Experience**: Jest to typ wyliczeniowy reprezentujący poziom doświadczenia pracownika. Posiada trzy wartości: **JUNIOR**, **MID** oraz **SENIOR**, które opisują różne poziomy doświadczenia pracownika w kontekście generowania wypłat. W zależności od tego, czy pracownik jest początkujący, średnio doświadczony czy doświadczony, jego poziom doświadczenia będzie odpowiednio oznaczony jako **JUNIOR**, **MID** lub **SENIOR**.

3.5.4. PythonDeveloper:

package com.example.payslipgenerator.employees;  
  
import lombok.Getter;  
import lombok.Setter;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
import org.hibernate.cfg.Configuration;  
  
import javax.persistence.Entity;  
import javax.persistence.EnumType;  
import javax.persistence.Enumerated;  
import javax.persistence.Table;  
  
@Entity  
@Getter  
@Setter  
@Table(name = "python\_developers")  
public class PythonDeveloper extends Employee implements DataToDbHandler {  
 private String toolName;  
  
 @Enumerated(EnumType.*STRING*)  
 private Experience experience;  
 private double baseSalary = 6500;  
  
 public PythonDeveloper(Long id, String name, String surname, String login, String password, String toolName, Experience experience) {  
 super(id, name, surname, login, password);  
 this.toolName = toolName;  
 this.experience = experience;  
 switch (experience) {  
 case *JUNIOR*:  
 this.baseSalary = baseSalary;  
 break;  
 case *MID*:  
 this.baseSalary = baseSalary \* 1.5;  
 break;  
 case *SENIOR*:  
 this.baseSalary = baseSalary \* 2;  
 break;  
 default:  
 this.baseSalary = baseSalary;  
 }  
 }  
  
 public PythonDeveloper() {  
 super(null, null, null, null, null);  
 }  
  
 @Override  
 public String[] generatePaySlip() {  
 String[] paySlip = new String[5];  
 paySlip[0] = getName();  
 paySlip[1] = getSurname();  
 paySlip[2] = "Python Developer";  
 paySlip[3] = String.*valueOf*(getExperience());  
 paySlip[4] = String.*valueOf*(getBaseSalary());  
 return paySlip;  
 }  
  
 @Override  
 public void addDataToDb(String name, String surname, String login, String password, String toolName, String experience) {  
 SessionFactory sessionFactory = new Configuration()  
 .configure()  
 .buildSessionFactory();  
 try (Session session = sessionFactory.openSession()) {  
 Transaction transaction = session.beginTransaction();  
  
 PythonDeveloper pythonDeveloper = new PythonDeveloper(null, name, surname, login, password, toolName, Experience.*valueOf*(experience.toUpperCase()));  
 session.save(pythonDeveloper);  
  
 transaction.commit();  
 session.close();  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 sessionFactory.close();  
 }  
 }  
  
}

Opis:

* **class PythonDeveloper extends Employee implements DataToDbHandler**: Jest to klasa reprezentująca programistę Pythona. Rozszerza klasę **Employee** i implementuje interfejs **DataToDbHandler**, co oznacza, że musi zaimplementować metodę **addDataToDb** z interfejsu **DataToDbHandler**.
* **@Entity**: Adnotacja oznaczająca, że ta klasa jest encją JPA, co oznacza, że dane obiekty tej klasy będą mapowane do rekordów w tabeli bazy danych.
* **@Table(name = "python\_developers")**: Określa nazwę tabeli w bazie danych, do której będą mapowane obiekty tej klasy.
* **@Getter** i **@Setter**: Adnotacje Lomboka generujące automatycznie gettery i settery dla pól klasy.
* **private String toolName**: Pole przechowujące nazwę narzędzia używanego przez programistę Pythona.
* **@Enumerated(EnumType.STRING) private Experience experience;**: Pole **experience** reprezentujące poziom doświadczenia programisty Pythona, które jest typem wyliczeniowym **Experience**.
* **private double baseSalary = 6500;**: Pole przechowujące bazową pensję programisty Pythona, która wynosi 6500.
* Konstruktory:
  + **public PythonDeveloper(Long id, String name, String surname, String login, String password, String toolName, Experience experience)**: Konstruktor inicjujący obiekt PythonDeveloper z danymi takimi jak id, imię, nazwisko, login, hasło, nazwa narzędzia i poziom doświadczenia. Na podstawie poziomu doświadczenia obliczana jest pensja, która zależy od bazowej pensji i mnożnika.
  + **public PythonDeveloper()**: Konstruktor domyślny.
* Metoda **generatePaySlip()**: Metoda implementowana z interfejsu **Employee**. Zwraca tablicę z danymi dotyczącymi wypłaty programisty Pythona, takimi jak imię, nazwisko, stanowisko, poziom doświadczenia i pensja.
* Metoda **addDataToDb()**: Implementacja metody interfejsu **DataToDbHandler**, która dodaje dane programisty Pythona do bazy danych. Tworzy nowy obiekt PythonDeveloper na podstawie przekazanych danych i zapisuje go do bazy danych.

Początek formularza

3.5.5. JavaDeveloper:

package com.example.payslipgenerator.employees;  
  
import lombok.\*;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
import org.hibernate.cfg.Configuration;  
  
import javax.persistence.Entity;  
import javax.persistence.EnumType;  
import javax.persistence.Enumerated;  
import javax.persistence.Table;  
  
@Entity  
@Getter  
@Setter  
@Table(name = "java\_developers")  
public class JavaDeveloper extends Employee implements DataToDbHandler {  
 private String toolName;  
  
 @Enumerated(EnumType.*STRING*)  
 private Experience experience;  
 private double baseSalary = 6000;  
  
 public JavaDeveloper(Long id, String name, String surname, String login, String password, String toolname, Experience experience) {  
 super(id, name, surname, login, password);  
 this.toolName = toolName;  
 this.experience = experience;  
 switch (this.experience) {  
 case *JUNIOR*:  
 this.baseSalary = baseSalary;  
 break;  
 case *MID*:  
 this.baseSalary = baseSalary \* 1.5;  
 break;  
 case *SENIOR*:  
 this.baseSalary = baseSalary \* 2;  
 break;  
 default:  
 this.baseSalary = baseSalary;  
 }  
 }  
  
 public JavaDeveloper() {  
 super(null, null, null, null, null);  
 }  
  
  
 @Override  
 public String[] generatePaySlip() {  
 String[] paySlip = new String[5];  
 paySlip[0] = getName();  
 paySlip[1] = getSurname();  
 paySlip[2] = "Java Developer";  
 paySlip[3] = String.*valueOf*(getExperience());  
 paySlip[4] = String.*valueOf*(getBaseSalary());  
 return paySlip;  
 }  
  
 @Override  
 public void addDataToDb(String name, String surname, String login, String password, String toolName, String experience) {  
 SessionFactory sessionFactory = new Configuration()  
 .configure()  
 .buildSessionFactory();  
 try (Session session = sessionFactory.openSession()) {  
 Transaction transaction = session.beginTransaction();  
  
 JavaDeveloper javaDeveloper = new JavaDeveloper(null, name, surname, login, password, toolName, Experience.*valueOf*(experience.toUpperCase()));  
 session.save(javaDeveloper);  
  
 transaction.commit();  
 session.close();  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 sessionFactory.close();  
 }  
 }  
}

Opis:

* **class JavaDeveloper extends Employee implements DataToDbHandler**: Jest to klasa reprezentująca programistę Javy. Rozszerza klasę **Employee** i implementuje interfejs **DataToDbHandler**, co oznacza, że musi zaimplementować metodę **addDataToDb** z interfejsu **DataToDbHandler**.
* **@Entity**: Adnotacja oznaczająca, że ta klasa jest encją JPA, co oznacza, że dane obiekty tej klasy będą mapowane do rekordów w tabeli bazy danych.
* **@Table(name = "java\_developers")**: Określa nazwę tabeli w bazie danych, do której będą mapowane obiekty tej klasy.
* **@Getter** i **@Setter**: Adnotacje Lomboka generujące automatycznie gettery i settery dla pól klasy.
* **private String toolName**: Pole przechowujące nazwę narzędzia używanego przez programistę Javy.
* **@Enumerated(EnumType.STRING) private Experience experience;**: Pole **experience** reprezentujące poziom doświadczenia programisty Javy, które jest typem wyliczeniowym **Experience**.
* **private double baseSalary = 6000;**: Pole przechowujące bazową pensję programisty Javy, która wynosi 6000.
* Konstruktory:
  + **public JavaDeveloper(Long id, String name, String surname, String login, String password, String toolname, Experience experience)**: Konstruktor inicjujący obiekt JavaDeveloper z danymi takimi jak id, imię, nazwisko, login, hasło, nazwa narzędzia i poziom doświadczenia. Na podstawie poziomu doświadczenia obliczana jest pensja, która zależy od bazowej pensji i mnożnika.
  + **public JavaDeveloper()**: Konstruktor domyślny.
* Metoda **generatePaySlip()**: Metoda implementowana z interfejsu **Employee**. Zwraca tablicę z danymi dotyczącymi wypłaty programisty Javy, takimi jak imię, nazwisko, stanowisko, poziom doświadczenia i pensja.
* Metoda **addDataToDb()**: Implementacja metody interfejsu **DataToDbHandler**, która dodaje dane programisty Javy do bazy danych. Tworzy nowy obiekt JavaDeveloper na podstawie przekazanych danych i zapisuje go do bazy danych.

3.5.6. FrontendDeveloper:

package com.example.payslipgenerator.employees;  
  
import lombok.Getter;  
import lombok.Setter;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
import org.hibernate.cfg.Configuration;  
  
import javax.persistence.Entity;  
import javax.persistence.EnumType;  
import javax.persistence.Enumerated;  
import javax.persistence.Table;  
  
@Entity  
@Getter  
@Setter  
@Table(name = "frontend\_developers")  
public class FrontEndDeveloper extends Employee implements DataToDbHandler {  
 private String toolName;  
  
 @Enumerated(EnumType.*STRING*)  
 private Experience experience;  
 private double baseSalary = 3500;  
  
 public FrontEndDeveloper(Long id, String name, String surname, String login, String password, String toolName, Experience experience) {  
 super(id, name, surname, login, password);  
 this.toolName = toolName;  
 this.experience = experience;  
 switch (experience) {  
 case *JUNIOR*:  
 this.baseSalary = baseSalary;  
 break;  
 case *MID*:  
 this.baseSalary = baseSalary \* 1.5;  
 break;  
 case *SENIOR*:  
 this.baseSalary = baseSalary \* 2;  
 break;  
 default:  
 this.baseSalary = baseSalary;  
 }  
 }  
  
 public FrontEndDeveloper() {  
 super(null, null, null, null, null);  
 }  
  
 @Override  
 public String[] generatePaySlip() {  
 String[] paySlip = new String[5];  
 paySlip[0] = getName();  
 paySlip[1] = getSurname();  
 paySlip[2] = "Front-end Developer";  
 paySlip[3] = String.*valueOf*(getExperience());  
 paySlip[4] = String.*valueOf*(getBaseSalary());  
 return paySlip;  
 }  
  
 @Override  
 public void addDataToDb(String name, String surname, String login, String password, String toolName, String experience) {  
 SessionFactory sessionFactory = new Configuration()  
 .configure()  
 .buildSessionFactory();  
 try (Session session = sessionFactory.openSession()) {  
 Transaction transaction = session.beginTransaction();  
  
 FrontEndDeveloper frontEndDeveloper = new FrontEndDeveloper(null, name, surname, login, password, toolName, Experience.*valueOf*(experience.toUpperCase()));  
 session.save(frontEndDeveloper);  
  
 transaction.commit();  
 session.close();  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 sessionFactory.close();  
 }  
 }  
}

Opis:

* **class FrontEndDeveloper extends Employee implements DataToDbHandler**: Jest to klasa reprezentująca front-end developera. Rozszerza klasę **Employee** i implementuje interfejs **DataToDbHandler**, co oznacza, że musi zaimplementować metodę **addDataToDb** z interfejsu **DataToDbHandler**.
* **@Entity**: Adnotacja oznaczająca, że ta klasa jest encją JPA, co oznacza, że dane obiekty tej klasy będą mapowane do rekordów w tabeli bazy danych.
* **@Table(name = "frontend\_developers")**: Określa nazwę tabeli w bazie danych, do której będą mapowane obiekty tej klasy.
* **@Getter** i **@Setter**: Adnotacje Lomboka generujące automatycznie gettery i settery dla pól klasy.
* **private String toolName**: Pole przechowujące nazwę narzędzia używanego przez front-end developera.
* **@Enumerated(EnumType.STRING) private Experience experience;**: Pole **experience** reprezentujące poziom doświadczenia front-end developera, które jest typem wyliczeniowym **Experience**.
* **private double baseSalary = 3500;**: Pole przechowujące bazową pensję front-end developera, która wynosi 3500.
* Konstruktory:
  + **public FrontEndDeveloper(Long id, String name, String surname, String login, String password, String toolName, Experience experience)**: Konstruktor inicjujący obiekt FrontEndDeveloper z danymi takimi jak id, imię, nazwisko, login, hasło, nazwa narzędzia i poziom doświadczenia. Na podstawie poziomu doświadczenia obliczana jest pensja, która zależy od bazowej pensji i mnożnika.
  + **public FrontEndDeveloper()**: Konstruktor domyślny.
* Metoda **generatePaySlip()**: Metoda implementowana z interfejsu **Employee**. Zwraca tablicę z danymi dotyczącymi wypłaty front-end developera, takimi jak imię, nazwisko, stanowisko, poziom doświadczenia i pensja.
* Metoda **addDataToDb()**: Implementacja metody interfejsu **DataToDbHandler**, która dodaje dane front-end developera do bazy danych. Tworzy nowy obiekt FrontEndDeveloper na podstawie przekazanych danych i zapisuje go do bazy danych.

3.5.7. DatabaseDeveloper:

package com.example.payslipgenerator.employees;  
  
import lombok.Getter;  
import lombok.Setter;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
import org.hibernate.cfg.Configuration;  
  
import javax.persistence.Entity;  
import javax.persistence.EnumType;  
import javax.persistence.Enumerated;  
import javax.persistence.Table;  
  
@Entity  
@Getter  
@Setter  
@Table(name = "database\_analysts")  
public class DatabaseAnalyst extends Employee implements DataToDbHandler {  
 private String toolName;  
  
 @Enumerated(EnumType.*STRING*)  
 private Experience experience;  
 private double baseSalary = 5000;  
  
 public DatabaseAnalyst(Long id, String name, String surname, String login, String password, String toolName, Experience experience) {  
 super(id, name, surname, login, password);  
 this.toolName = toolName;  
 this.experience = experience;  
 switch (experience) {  
 case *JUNIOR*:  
 this.baseSalary = baseSalary;  
 break;  
 case *MID*:  
 this.baseSalary = baseSalary \* 1.5;  
 break;  
 case *SENIOR*:  
 this.baseSalary = baseSalary \* 2;  
 break;  
 default:  
 this.baseSalary = baseSalary;  
 }  
 }  
  
 public DatabaseAnalyst() {  
 super(null, null, null, null, null);  
 }  
  
 @Override  
 public String[] generatePaySlip() {  
 String[] paySlip = new String[5];  
 paySlip[0] = getName();  
 paySlip[1] = getSurname();  
 paySlip[2] = "Database Analyst";  
 paySlip[3] = String.*valueOf*(getExperience());  
 paySlip[4] = String.*valueOf*(getBaseSalary());  
 return paySlip;  
 }  
  
 @Override  
 public void addDataToDb(String name, String surname, String login, String password, String toolName, String experience) {  
 SessionFactory sessionFactory = new Configuration()  
 .configure()  
 .buildSessionFactory();  
 try (Session session = sessionFactory.openSession()) {  
 Transaction transaction = session.beginTransaction();  
  
 DatabaseAnalyst databaseAnalyst = new DatabaseAnalyst(null, name, surname, login, password, toolName, Experience.*valueOf*(experience.toUpperCase()));  
 session.save(databaseAnalyst);  
  
 transaction.commit();  
 session.close();  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 sessionFactory.close();  
 }  
 }  
}

Opis:

* **class DatabaseAnalyst extends Employee implements DataToDbHandler**: Klasa **DatabaseAnalyst** reprezentuje analityka baz danych. Rozszerza klasę **Employee** i implementuje interfejs **DataToDbHandler**.
* **@Entity**: Adnotacja oznaczająca, że ta klasa jest encją JPA, co oznacza, że dane obiekty tej klasy będą mapowane do rekordów w tabeli bazy danych.
* **@Table(name = "database\_analysts")**: Określa nazwę tabeli w bazie danych, do której będą mapowane obiekty tej klasy.
* **@Getter** i **@Setter**: Adnotacje Lomboka generujące automatycznie gettery i settery dla pól klasy.
* **private String toolName**: Pole przechowujące nazwę narzędzia używanego przez analityka baz danych.
* **@Enumerated(EnumType.STRING) private Experience experience;**: Pole **experience** reprezentujące poziom doświadczenia analityka baz danych, które jest typem wyliczeniowym **Experience**.
* **private double baseSalary = 5000;**: Pole przechowujące bazową pensję analityka baz danych, która wynosi 5000.
* Konstruktory:
  + **public DatabaseAnalyst(Long id, String name, String surname, String login, String password, String toolName, Experience experience)**: Konstruktor inicjujący obiekt DatabaseAnalyst z danymi takimi jak id, imię, nazwisko, login, hasło, nazwa narzędzia i poziom doświadczenia. Na podstawie poziomu doświadczenia obliczana jest pensja, która zależy od bazowej pensji i mnożnika.
  + **public DatabaseAnalyst()**: Konstruktor domyślny.
* Metoda **generatePaySlip()**: Metoda implementowana z interfejsu **Employee**. Zwraca tablicę z danymi dotyczącymi wypłaty analityka baz danych, takimi jak imię, nazwisko, stanowisko, poziom doświadczenia i pensja.
* Metoda **addDataToDb()**: Implementacja metody interfejsu **DataToDbHandler**, która dodaje dane analityka baz danych do bazy danych. Tworzy nowy obiekt DatabaseAnalyst na podstawie przekazanych danych i zapisuje go do bazy danych.

Początek formularza

3.5.8. CSharpDeveloper:

package com.example.payslipgenerator.employees;  
  
import lombok.Getter;  
import lombok.Setter;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
import org.hibernate.cfg.Configuration;  
  
import javax.persistence.Entity;  
import javax.persistence.EnumType;  
import javax.persistence.Enumerated;  
import javax.persistence.Table;  
  
@Entity  
@Getter  
@Setter  
@Table(name = "csharp\_developers")  
public class CSharpDeveloper extends Employee implements DataToDbHandler {  
 private String toolName;  
  
 @Enumerated(EnumType.*STRING*)  
 private Experience experience;  
 private double baseSalary = 4500;  
  
 public CSharpDeveloper(Long id, String name, String surname, String login, String password, String toolName, Experience experience) {  
 super(id, name, surname, login, password);  
 this.toolName = toolName;  
 this.experience = experience;  
 switch (experience) {  
 case *JUNIOR*:  
 this.baseSalary = baseSalary;  
 break;  
 case *MID*:  
 this.baseSalary = baseSalary \* 1.5;  
 break;  
 case *SENIOR*:  
 this.baseSalary = baseSalary \* 2;  
 break;  
 default:  
 this.baseSalary = baseSalary;  
 }  
 }  
  
 public CSharpDeveloper() {  
 super(null, null, null, null, null);  
 }  
  
 @Override  
 public String[] generatePaySlip() {  
 String[] paySlip = new String[5];  
 paySlip[0] = getName();  
 paySlip[1] = getSurname();  
 paySlip[2] = "C# Developer";  
 paySlip[3] = String.*valueOf*(getExperience());  
 paySlip[4] = String.*valueOf*(getBaseSalary());  
 return paySlip;  
 }  
  
 @Override  
 public void addDataToDb(String name, String surname, String login, String password, String toolName, String experience) {  
 SessionFactory sessionFactory = new Configuration()  
 .configure()  
 .buildSessionFactory();  
 try (Session session = sessionFactory.openSession()) {  
 Transaction transaction = session.beginTransaction();  
  
 CSharpDeveloper cSharpDeveloper = new CSharpDeveloper(null, name, surname, login, password, toolName, Experience.*valueOf*(experience.toUpperCase()));  
 session.save(cSharpDeveloper);  
  
 transaction.commit();  
 session.close();  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 sessionFactory.close();  
 }  
 }  
}

Opis:

* **class CSharpDeveloper extends Employee implements DataToDbHandler**: Klasa **CSharpDeveloper** reprezentuje programistę C#. Rozszerza klasę **Employee** i implementuje interfejs **DataToDbHandler**.
* **@Entity**: Adnotacja oznaczająca, że ta klasa jest encją JPA, co oznacza, że dane obiekty tej klasy będą mapowane do rekordów w tabeli bazy danych.
* **@Table(name = "csharp\_developers")**: Określa nazwę tabeli w bazie danych, do której będą mapowane obiekty tej klasy.
* **@Getter** i **@Setter**: Adnotacje Lomboka generujące automatycznie gettery i settery dla pól klasy.
* **private String toolName**: Pole przechowujące nazwę narzędzia używanego przez programistę C#.
* **@Enumerated(EnumType.STRING) private Experience experience;**: Pole **experience** reprezentujące poziom doświadczenia programisty C#, które jest typem wyliczeniowym **Experience**.
* **private double baseSalary = 4500;**: Pole przechowujące bazową pensję programisty C#, która wynosi 4500.
* Konstruktory:
  + **public CSharpDeveloper(Long id, String name, String surname, String login, String password, String toolName, Experience experience)**: Konstruktor inicjujący obiekt CSharpDeveloper z danymi takimi jak id, imię, nazwisko, login, hasło, nazwa narzędzia i poziom doświadczenia. Na podstawie poziomu doświadczenia obliczana jest pensja, która zależy od bazowej pensji i mnożnika.
  + **public CSharpDeveloper()**: Konstruktor domyślny.
* Metoda **generatePaySlip()**: Metoda implementowana z interfejsu **Employee**. Zwraca tablicę z danymi dotyczącymi wypłaty programisty C#, takimi jak imię, nazwisko, stanowisko, poziom doświadczenia i pensja.
* Metoda **addDataToDb()**: Implementacja metody interfejsu **DataToDbHandler**, która dodaje dane programisty C# do bazy danych. Tworzy nowy obiekt CSharpDeveloper na podstawie przekazanych danych i zapisuje go do bazy danych.

3.5.9. CppDeveloper:

package com.example.payslipgenerator.employees;  
  
import lombok.Getter;  
import lombok.Setter;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
import org.hibernate.cfg.Configuration;  
  
import javax.persistence.Entity;  
import javax.persistence.EnumType;  
import javax.persistence.Enumerated;  
import javax.persistence.Table;  
  
@Entity  
@Getter  
@Setter  
@Table(name = "cpp\_developers")  
public class CppDeveloper extends Employee implements DataToDbHandler {  
 private String toolName;  
  
 @Enumerated(EnumType.*STRING*)  
 private Experience experience;  
 private double baseSalary = 4000;  
  
 public CppDeveloper(Long id, String name, String surname, String login, String password, String toolName, Experience experience) {  
 super(id, name, surname, login, password);  
 this.toolName = toolName;  
 this.experience = experience;  
 switch (experience) {  
 case *JUNIOR*:  
 this.baseSalary = baseSalary;  
 break;  
 case *MID*:  
 this.baseSalary = baseSalary \* 1.5;  
 break;  
 case *SENIOR*:  
 this.baseSalary = baseSalary \* 2;  
 break;  
 default:  
 this.baseSalary = baseSalary;  
 }  
 }  
  
 public CppDeveloper() {  
 super(null, null, null, null, null);  
 }  
  
 @Override  
 public String[] generatePaySlip() {  
 String[] paySlip = new String[5];  
 paySlip[0] = getName();  
 paySlip[1] = getSurname();  
 paySlip[2] = "C++ Developer";  
 paySlip[3] = String.*valueOf*(getExperience());  
 paySlip[4] = String.*valueOf*(getBaseSalary());  
 return paySlip;  
 }  
  
 @Override  
 public void addDataToDb(String name, String surname, String login, String password, String toolName, String experience) {  
 SessionFactory sessionFactory = new Configuration()  
 .configure()  
 .buildSessionFactory();  
 try (Session session = sessionFactory.openSession()) {  
 Transaction transaction = session.beginTransaction();  
  
 CppDeveloper cppDeveloper = new CppDeveloper(null, name, surname, login, password, toolName, Experience.*valueOf*(experience.toUpperCase()));  
 session.save(cppDeveloper);  
  
 transaction.commit();  
 session.close();  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 sessionFactory.close();  
 }  
 }  
}

Opis:

* **class CppDeveloper extends Employee implements DataToDbHandler**: Klasa **CppDeveloper** reprezentuje programistę C++. Rozszerza klasę **Employee** i implementuje interfejs **DataToDbHandler**.
* **@Entity**: Adnotacja oznaczająca, że ta klasa jest encją JPA, co oznacza, że dane obiekty tej klasy będą mapowane do rekordów w tabeli bazy danych.
* **@Table(name = "cpp\_developers")**: Określa nazwę tabeli w bazie danych, do której będą mapowane obiekty tej klasy.
* **@Getter** i **@Setter**: Adnotacje Lomboka generujące automatycznie gettery i settery dla pól klasy.
* **private String toolName**: Pole przechowujące nazwę narzędzia używanego przez programistę C++.
* **@Enumerated(EnumType.STRING) private Experience experience;**: Pole **experience** reprezentujące poziom doświadczenia programisty C++, które jest typem wyliczeniowym **Experience**.
* **private double baseSalary = 4000;**: Pole przechowujące bazową pensję programisty C++, która wynosi 4000.
* Konstruktory:
  + **public CppDeveloper(Long id, String name, String surname, String login, String password, String toolName, Experience experience)**: Konstruktor inicjujący obiekt CppDeveloper z danymi takimi jak id, imię, nazwisko, login, hasło, nazwa narzędzia i poziom doświadczenia. Na podstawie poziomu doświadczenia obliczana jest pensja, która zależy od bazowej pensji i mnożnika.
  + **public CppDeveloper()**: Konstruktor domyślny.
* Metoda **generatePaySlip()**: Metoda implementowana z interfejsu **Employee**. Zwraca tablicę z danymi dotyczącymi wypłaty programisty C++, takimi jak imię, nazwisko, stanowisko, poziom doświadczenia i pensja.
* Metoda **addDataToDb()**: Implementacja metody interfejsu **DataToDbHandler**, która dodaje dane programisty C++ do bazy danych. Tworzy nowy obiekt CppDeveloper na podstawie przekazanych danych i zapisuje go do bazy danych.

4. **Baza danych**

Baza danych jest w MySQL Workbench. By aplikacja działała poprawnie należy pobrać to środowisko, utworzyć połączenie oraz przystosować (port, login, hasło etc.) plik konfiguracyjny hibernate.cfg.xml do swoich potrzeb.

Defaultowo baza danych aplikacji jest pusta. Po uruchomieniu klasy Main z pakietu service, która używa statycznej metody setDataInDb() z klasy PaySlipGeneratorService() wypełnia się ona następującymi tabelami oraz pracownikami firmy informatycznej:

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, oprogramowanie, zrzut ekranu, Ikona komputerowa

Opis wygenerowany automatycznie

Obraz zawierający tekst, oprogramowanie, numer, Ikona komputerowa

Opis wygenerowany automatycznie

Obraz zawierający tekst, oprogramowanie, zrzut ekranu

Opis wygenerowany automatycznie

Obraz zawierający tekst, oprogramowanie, Ikona komputerowa, numer

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, linia

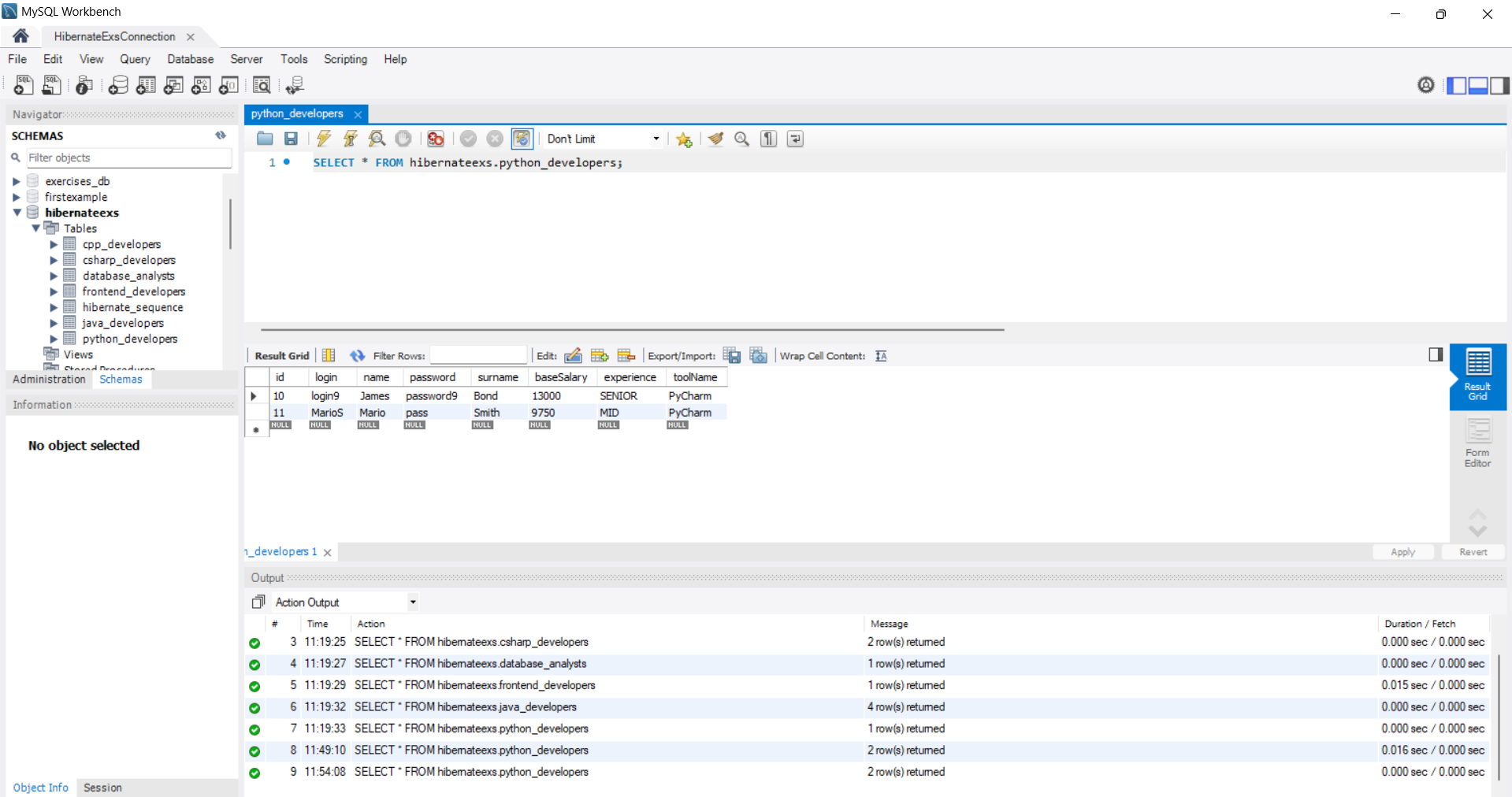
Opis wygenerowany automatycznie

Obraz zawierający tekst, oprogramowanie, Ikona komputerowa, numer

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, linia

Opis wygenerowany automatycznie

Baza jest następnie uzupełniana o kolejnych pracowników w przypadku rejestracji nowego developera lub analityka poprzez stronę http://localhost:8080/signup.  


**5. Aplikacja z poziomu użytkownika**

Strona główna daje możliwość logowania lub rejestracji pracownika firmy komputerowej:

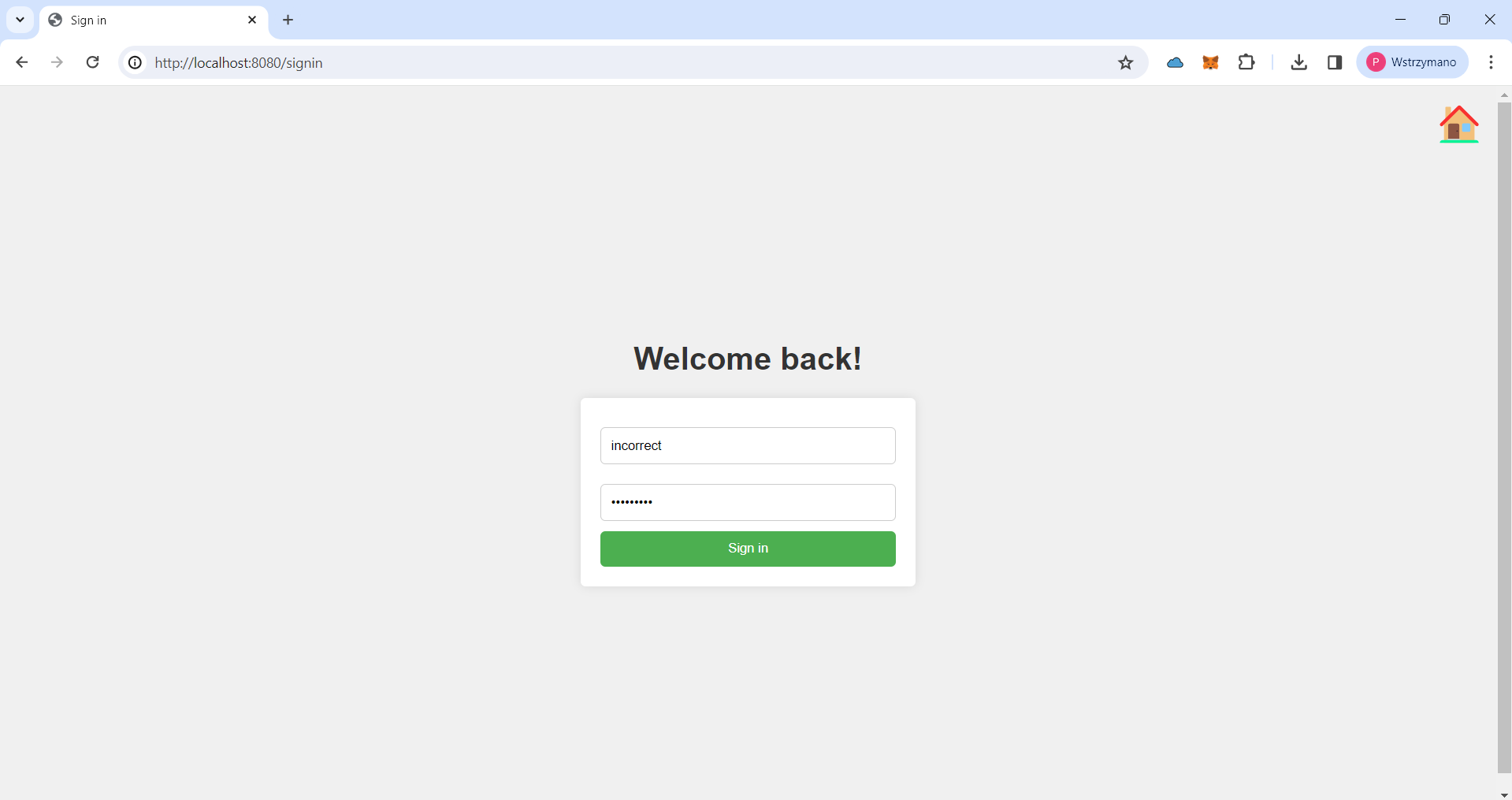
Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

W rezultacie wyboru opcji logowania użytkownik przeniesiony jest na poniższą stronę:

Obraz zawierający tekst, oprogramowanie, Ikona komputerowa, System operacyjny

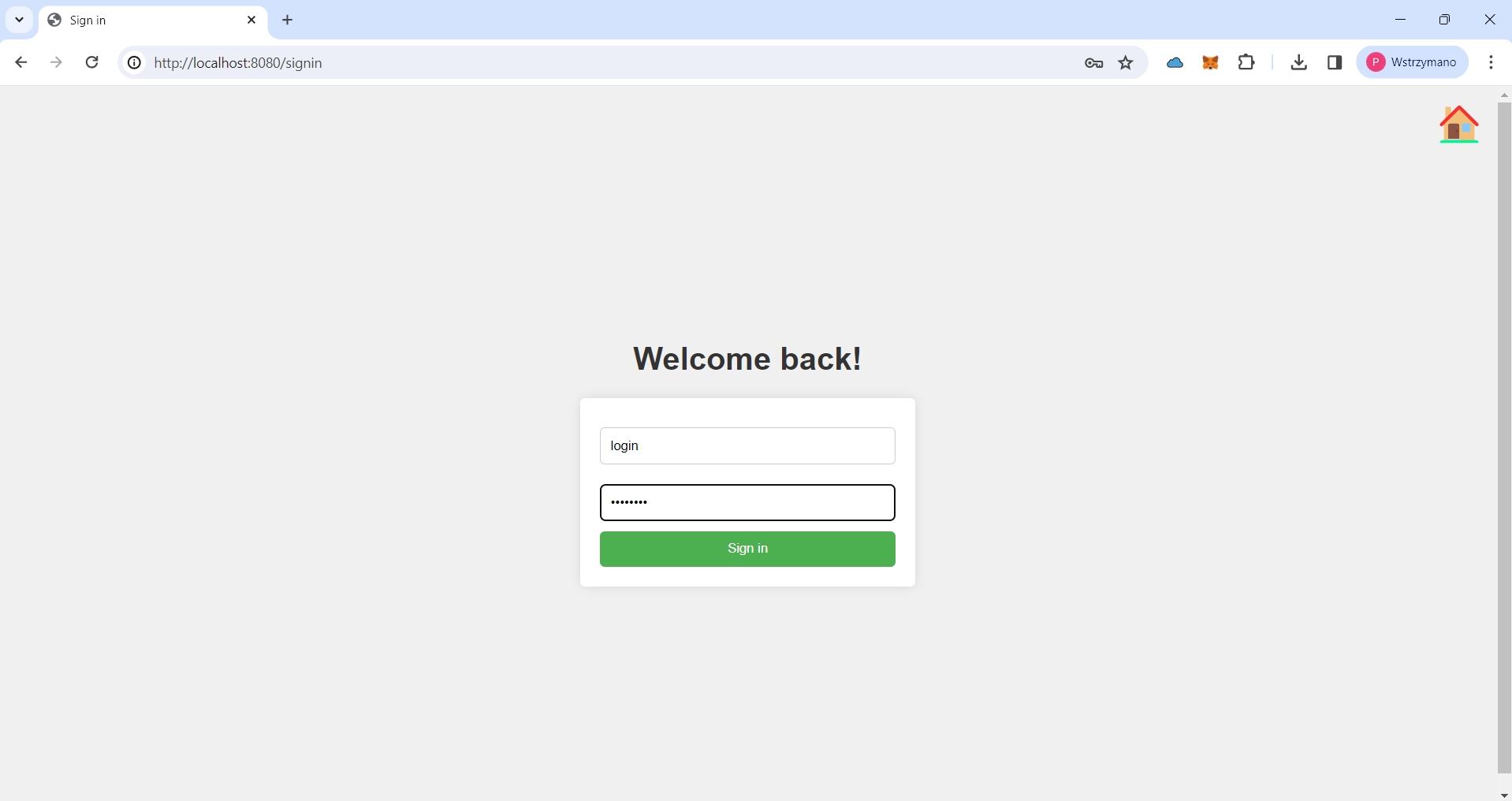
Opis wygenerowany automatycznie

Przy wprowadzeniu błędnych danych pracownika (brak pracownika w bazie) i kliknięciu Sign in użytkownik przeniesiony jest na poniższą stronę z komunikatem o błędnych danych logowania:  


Obraz zawierający tekst, oprogramowanie, Ikona komputerowa, System operacyjny

Opis wygenerowany automatycznie

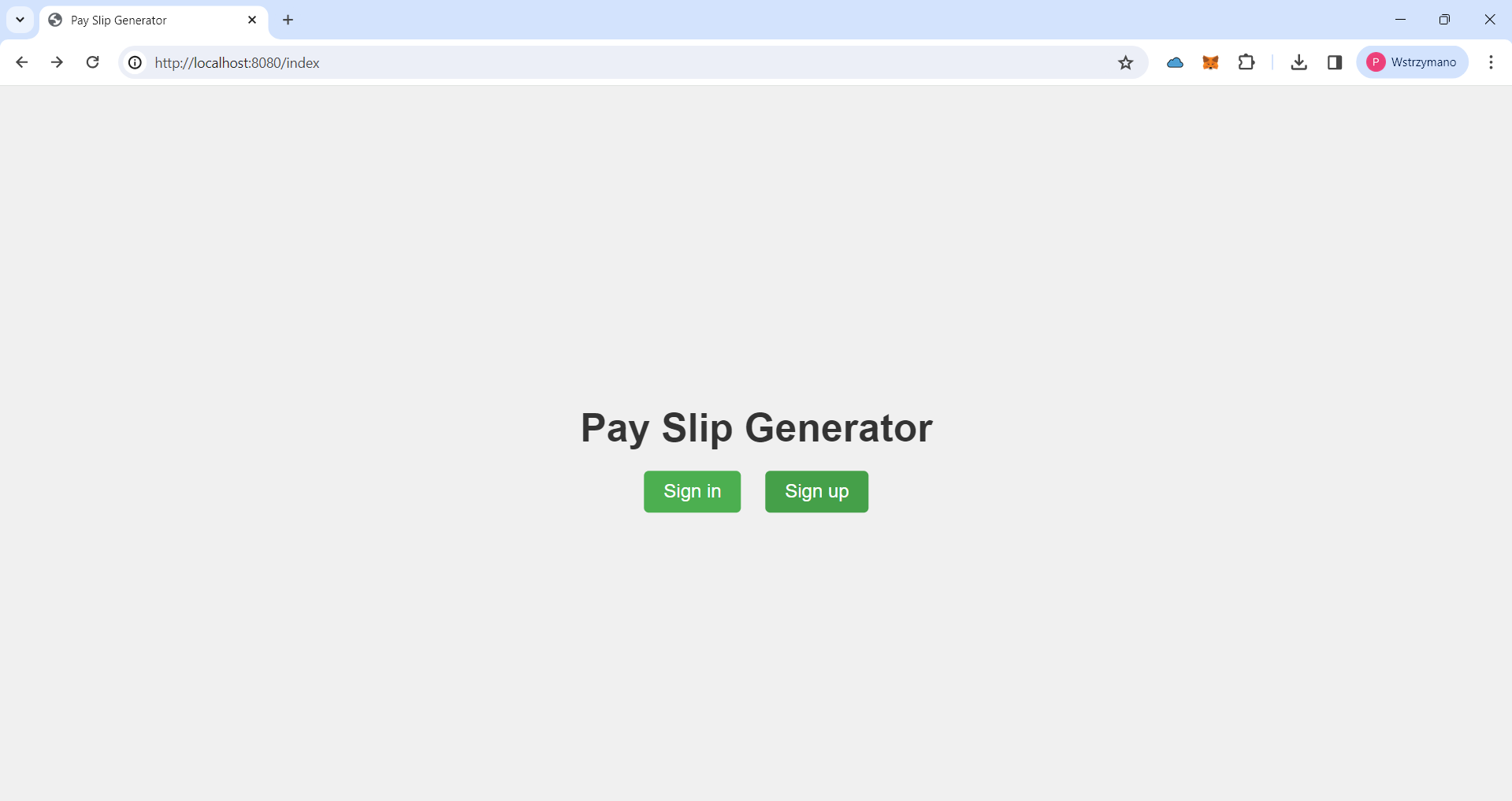
Po kliknięciu Try Again użytkownik jest przeniesiony z powrotem na stronę logowania po czym po wprowadzeniu poprawnych danych (pracownik jest w bazie) zostaje przeniesiony na stronę z swoim paskiem wypłaty:



Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

Po kliknięciu ikonki w prawym górnym rogu zostaje wylogowany i przeniesiony na stronę główną:

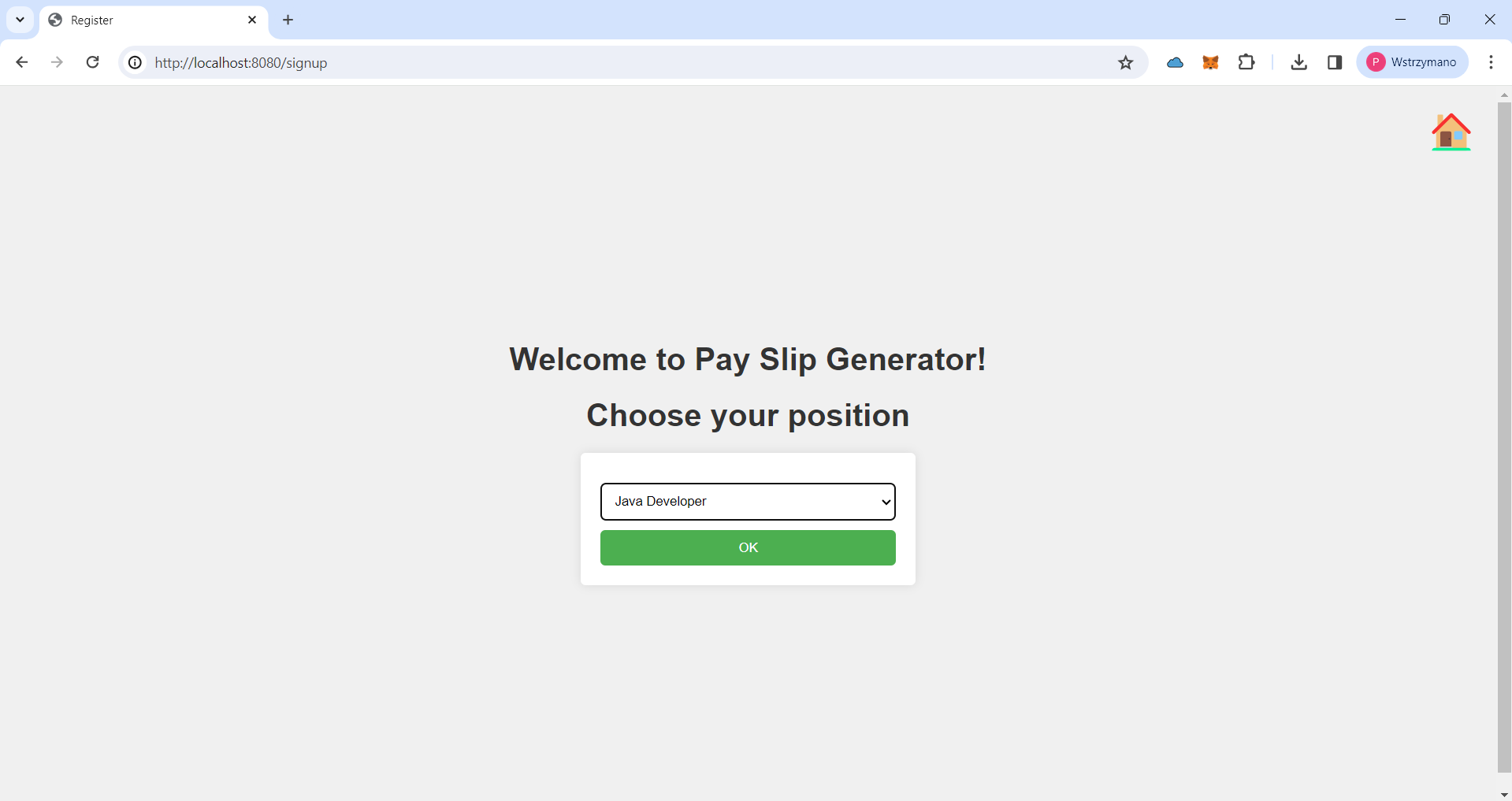


W rezultacie wyboru opcji rejestracji użytkownik przeniesiony jest na poniższą stronę z możliwością wyboru specjalizacji pracownika:

Obraz zawierający tekst, oprogramowanie, Ikona komputerowa, System operacyjny

Opis wygenerowany automatycznie

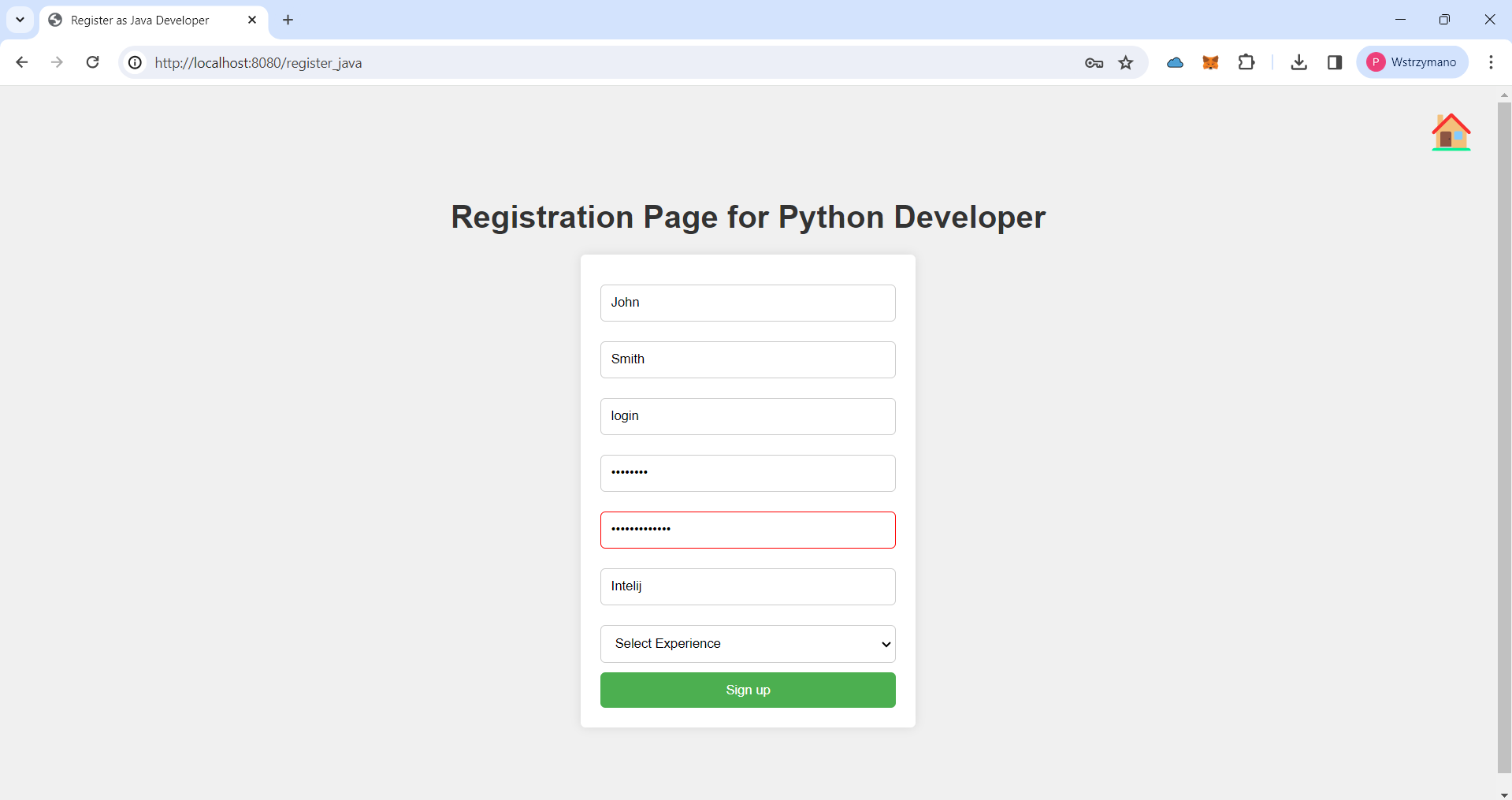
Po wybraniu jednej z opcji i kliknięciu OK użytkownik jest przeniesiony na stronę rejestracji:



Obraz zawierający tekst, oprogramowanie, Ikona komputerowa, Strona internetowa

Opis wygenerowany automatycznie

Niezgodność haseł powoduje blokadę możliwości rejestracji:



Strona daje też możliwość wyboru doświadczenia rejestrowanego pracownika:  
Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

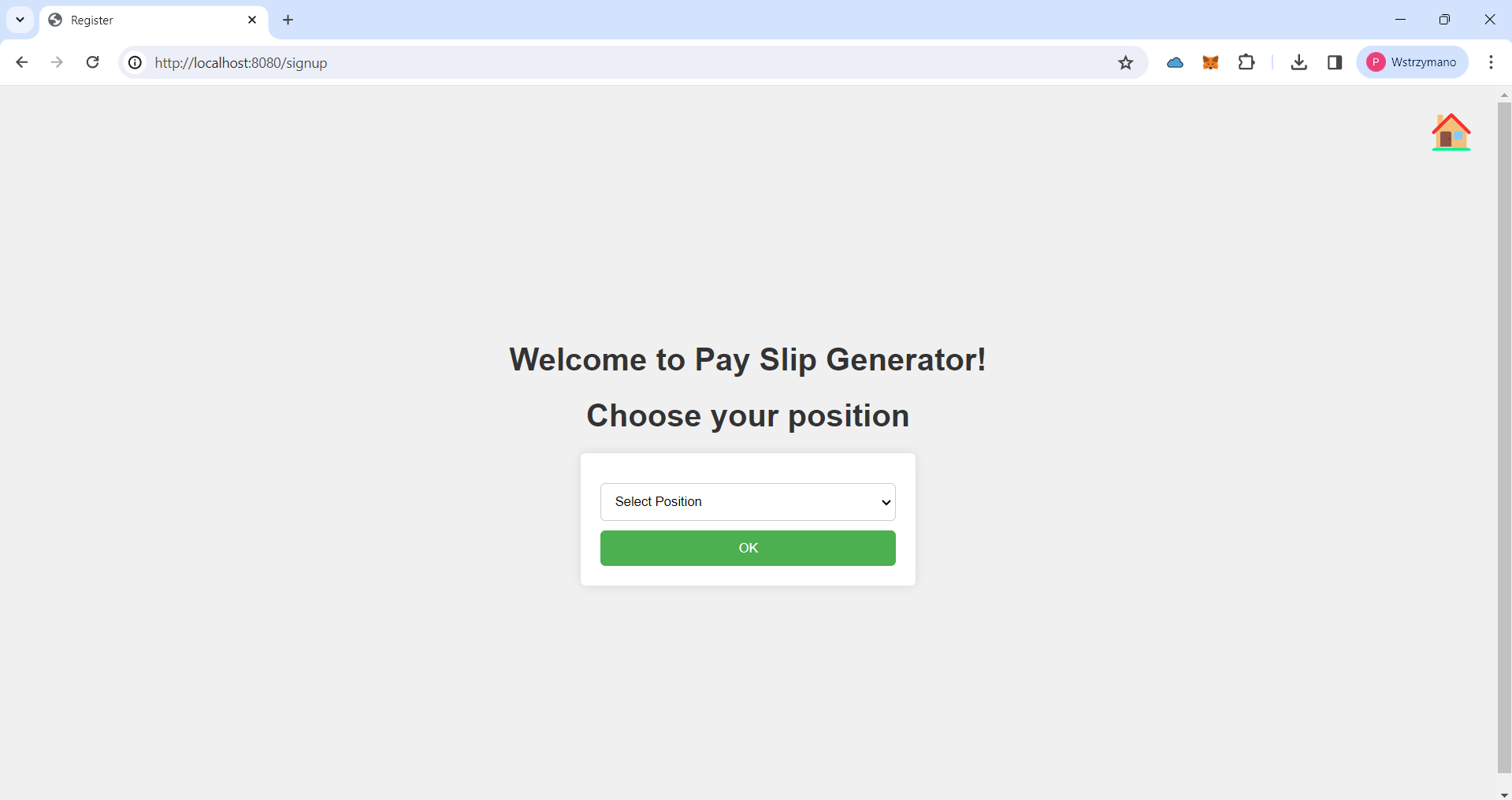
Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

Jeśli login nie jest unikalny (w bazie istnieje już pracownik o takim loginie) użytkownik jest przekierowywany na stronę z komunikatem o błędnych loginie podanym w czasie rejestracji:

Obraz zawierający tekst, oprogramowanie, Ikona komputerowa, System operacyjny

Opis wygenerowany automatycznie

Po kliknięciu Try Again jest przeniesiony z powrotem na stronę rejestracji pracownika:  


Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

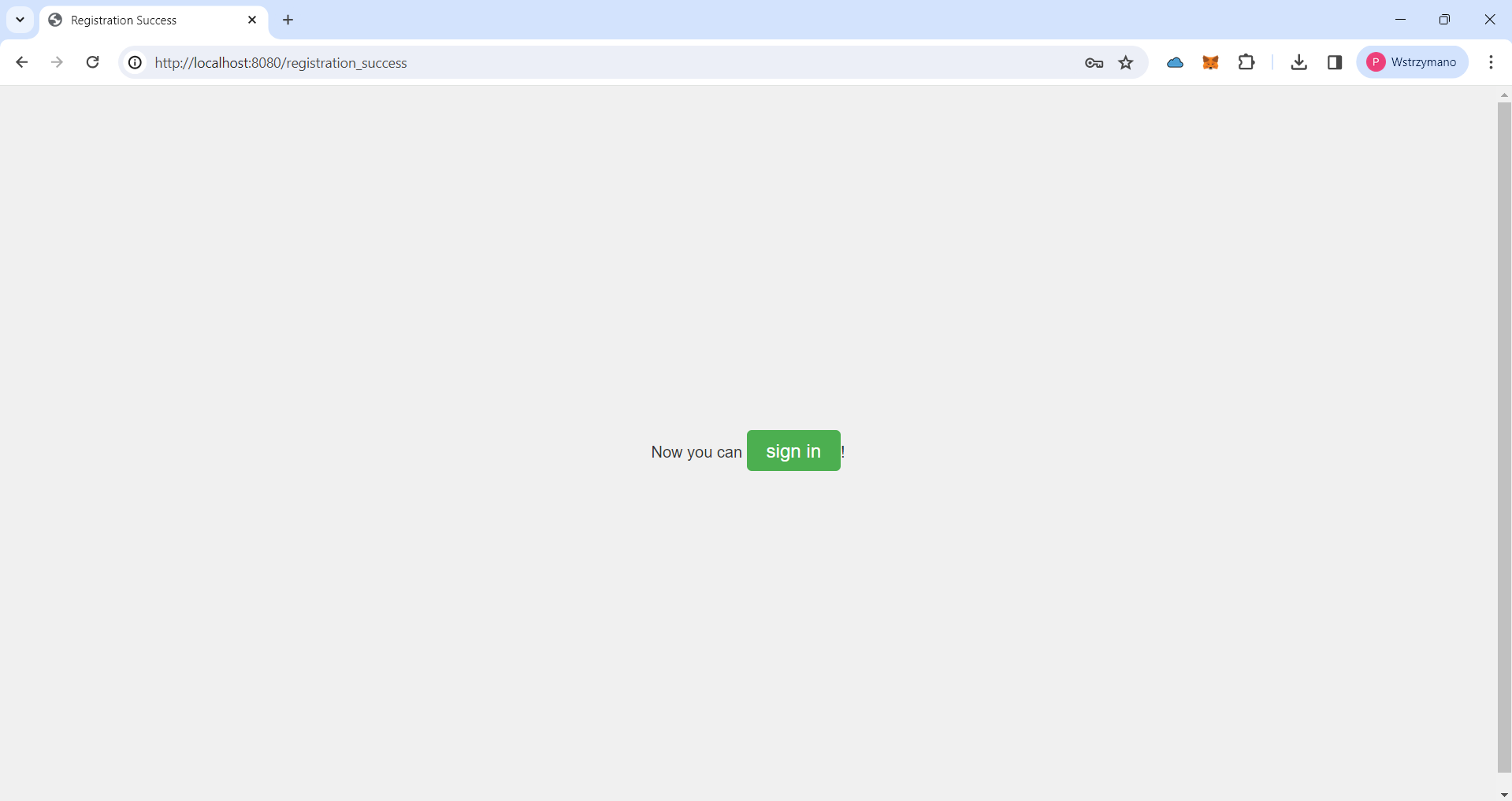
Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

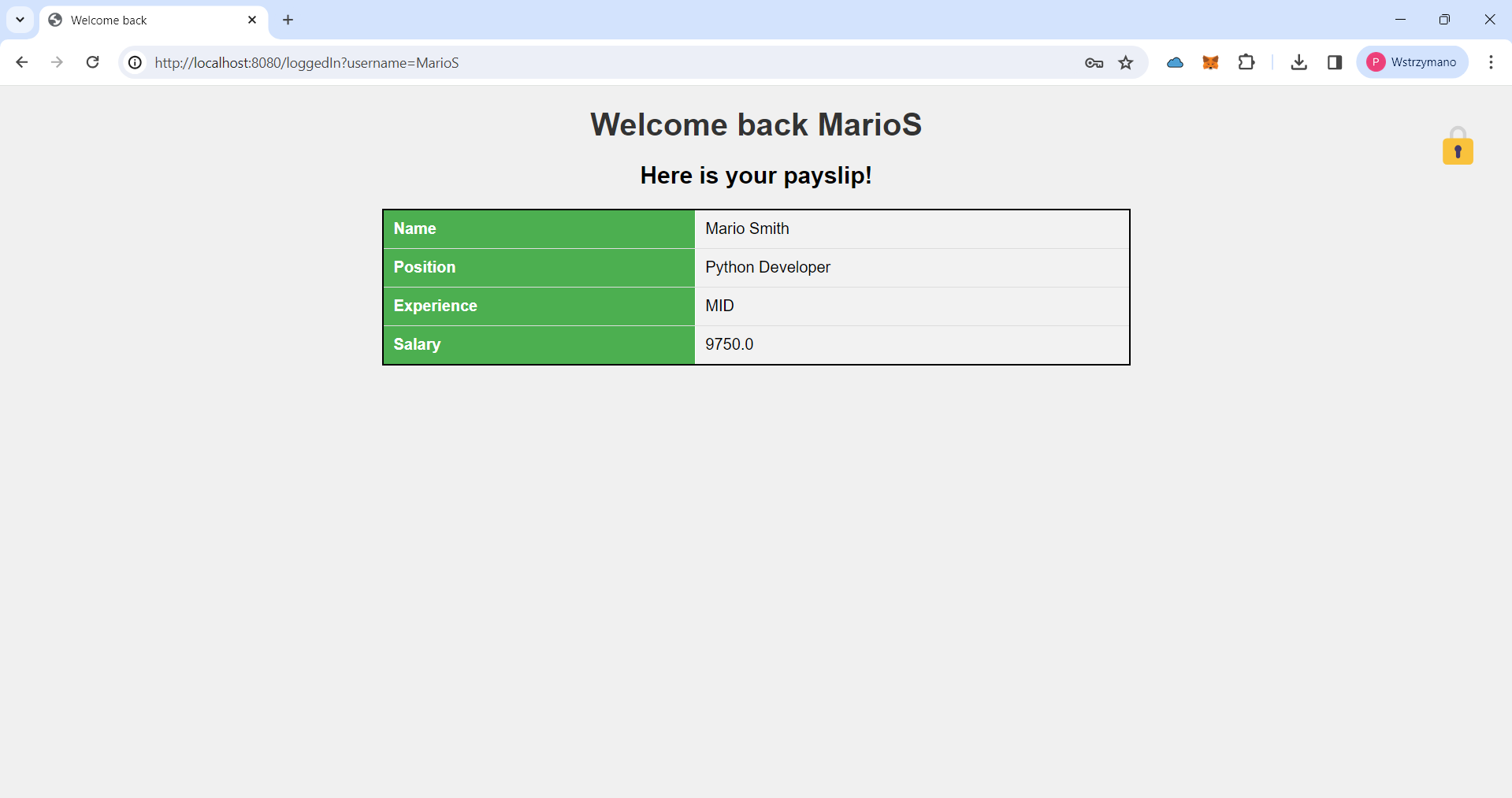
Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

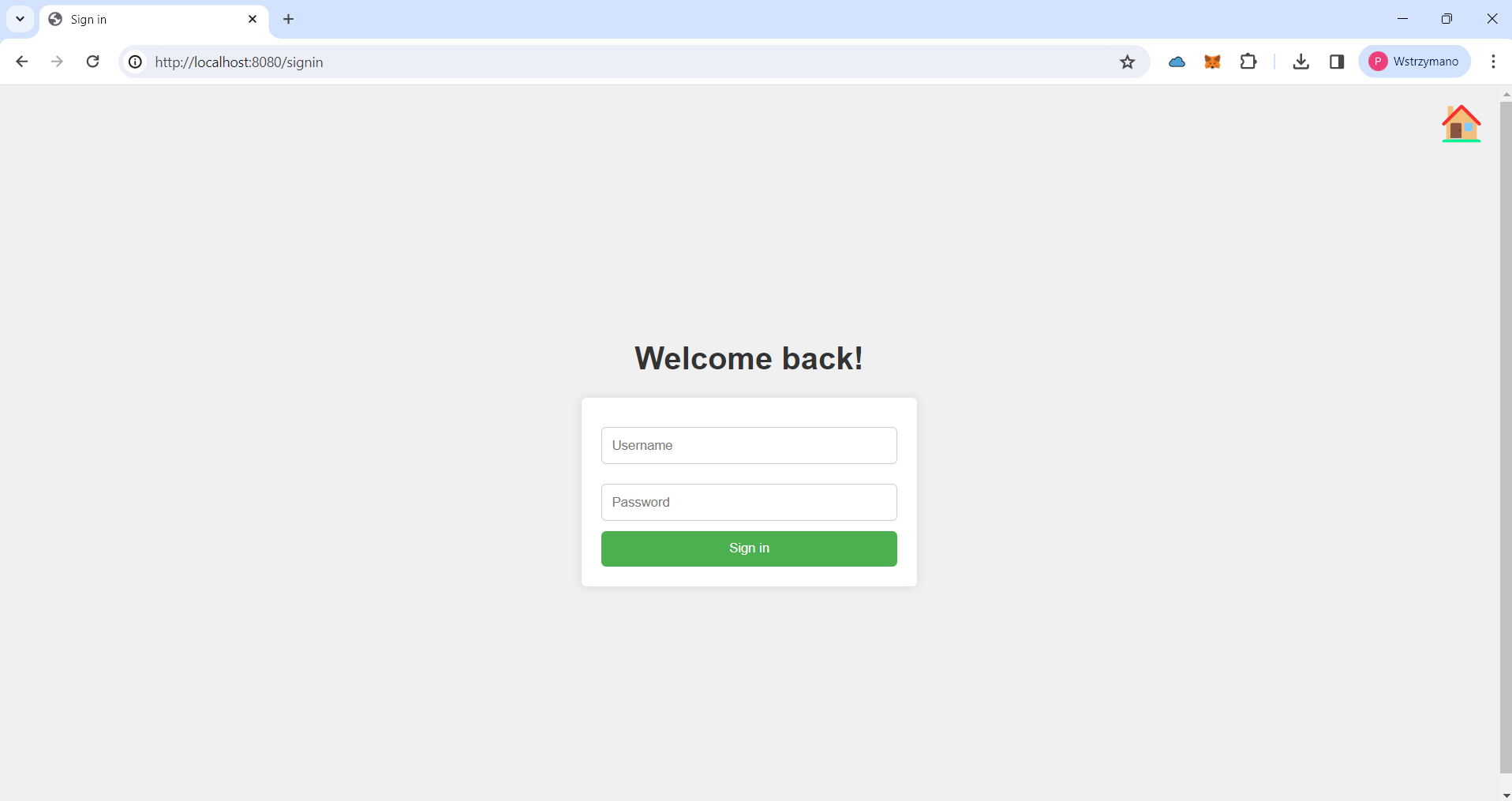
Po podaniu akceptowalnych danych i kliknięciu Sign up rejestracja zostaje potwierdzona:  


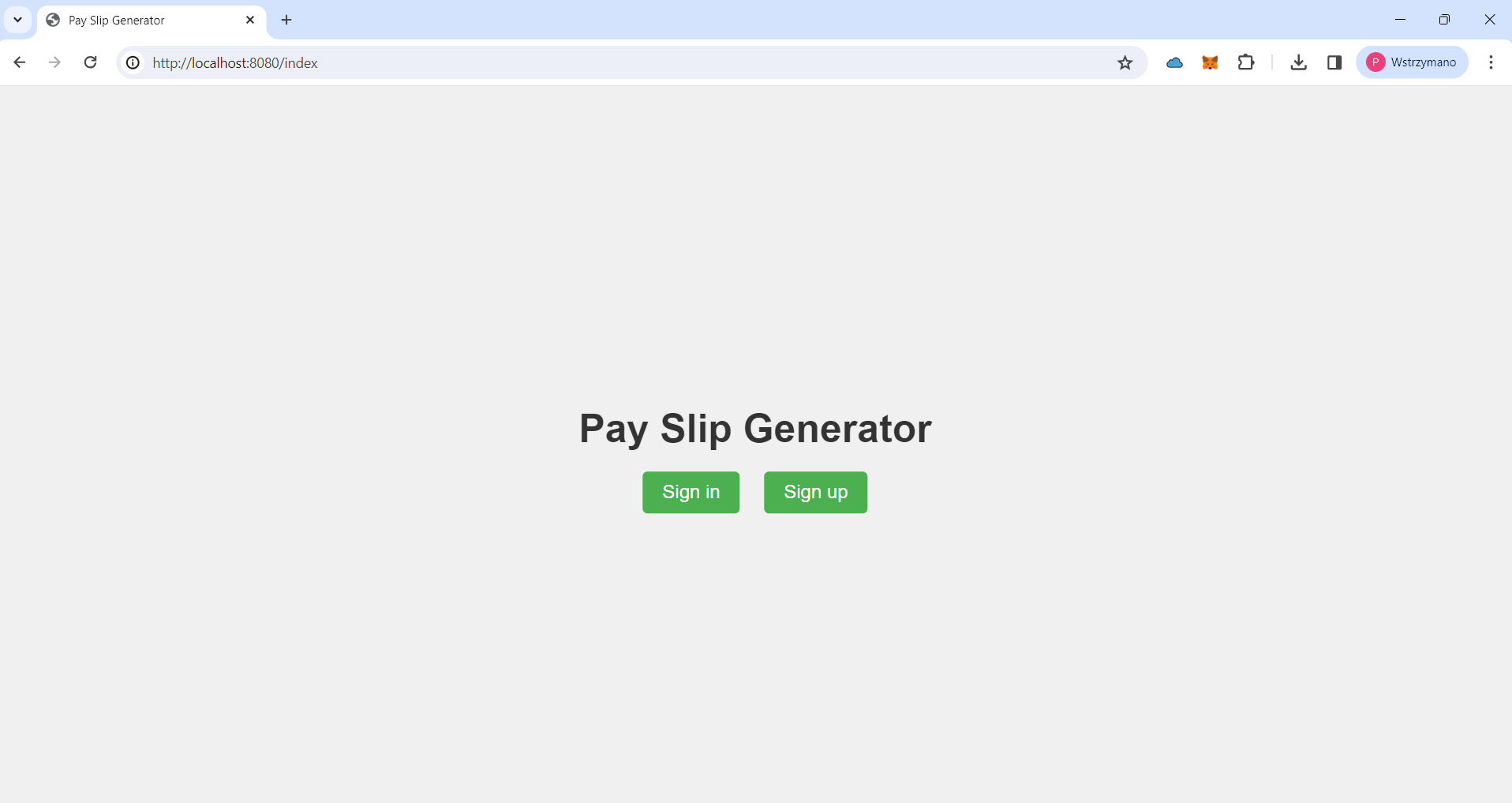
Po kliknięciu sign in użytkownik jest przeniesiony na stronę logowania:  
Obraz zawierający tekst, oprogramowanie, Ikona komputerowa, Oprogramowanie multimedialne

Opis wygenerowany automatycznie  
Po kliknięciu Sing in zostaje zalogowany i przeniesiony na stronę ze swoim paskiem wypłaty:



Jeśli natomiast na stronie logowani lub innej kliknie ikonkę domu zostanie przeniesiony na stronę główną aplikacji:





**6. Rozwinięcia**

Projekt może być dalej rozwijany. Na poziomie frontendu można dopracować style poszczególnych stron. Można uczynić też profil pracownika bardziej przyjaznym poprzez dodanie opcji modyfikacji konta. Dodatkowo można dodać takie opcje jak dodawanie nowego typu pracownika itp.