Instrukcja 11

Laboratorium 14-15
Testy funkcjonalne

Cel laboratorium:
Nabycie umiejętności tworzenia testów funkcjonalnych za pomocą narzędzia Selenium IDE.

1. Wg wskazówek podanych w p.1 **Dodatku 2**, należy zainstalować narzędzie ***Selenium IDE*** w przeglądarce ***FireFox*** firmy ***Mozilla***.
2. Na podstawie p. 2 z **Dodatku 1**, należy w środowisku ***Apache NetBeans 18***, z zainstalowaną platformą ***Java EE 8.0***, wykonać aplikację wygenerowaną na podstawie schematu relacyjnej bazy danych wg zasady „database-first". Wykonanie pliku zawierającego skrypt SQL z instrukcjami tworzącymi tabele należy oprzeć na projektach w ramach przedmiotu **Bazy danych 2**. Dalszy ciąg działań wykonać zgodnie z tutorialem z **Dodatku 1**. Należy zdefiniować przykładowe scenariusze testowanych przypadków użycia aplikacji w zakresie dodawania, edycji i usuwania danych (p.2, **Dodatek 2**)
3. Należy wykonać testy typu ***Test Case*** aplikacji typu ***Java Web Application***, wykonanej w p.2. W tabelce poniżej podano informacje dotyczące wyboru stron takiej aplikacji do testowania oraz przykładów rozwiązań typu ***Test Case***.
   W **Dodatku 2** umieszczono przykłady testów funkcjonalnych, które umożliwiają sprawdzenie działania wygenerowanej aplikacji w przykładzie z **Dodatku 1** w zakresie budowy stron oraz kontrolę poprawnego działania aplikacji za pomocą poleceń z grupy assert… oraz verify…

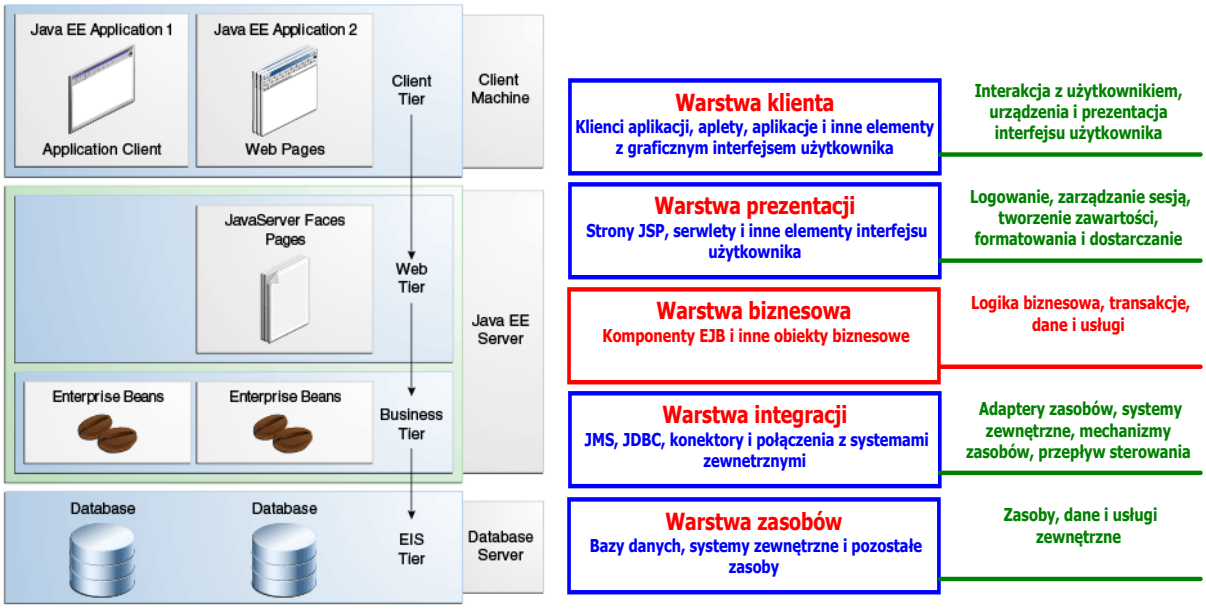
<table>
<tr><td rowspan="6">Grupa
Liczba osób</td><td rowspan="6">Liczba stron do testowania</td><td colspan="4">Przykłady stron</td></tr>
<tr><td colspan="2">1-a para stron</td><td colspan="2">2-a para stron</td></tr>
<tr><td>Create.html</td><td>List.html</td><td>Edit.html
(na podstawie Create.html)</td><td>View.html</td></tr>
<tr><td colspan="4">Przykłady rozwiązań
TestSelenium_WypozyczalniKsiazek1 (str. 22-26 instrukcji)</td></tr>
<tr><td colspan="4">TestSelenium_WypozyczalniKsiazek2 (str. 27-29)</td></tr>
<tr><td>Tabela 3,
Tabela 1</td><td>Weryfikacja-
Tabela 2, Tabela 4</td><td>Tabela 3,
Tabela 1</td><td>Weryfikacja-
Tabela 2, Tabela 4</td></tr>
<tr><td>1</td><td>1 para stron</td><td rowspan="2">Nagranie obsługi dodawana nowej danej i /lub ręczne zdefiniowanie obsługi dodawania nowej danej.
Weryfikacja elementów UI strony</td><td rowspan="2">Nagranie obsługi wyświetlania danych i /lub ręczne zdefiniowanie obsługi wyświetlania.
Dokonanie weryfikacji zawartości wyświetlanych danych</td><td rowspan="2">Nagranie obsługi edycji wybranej danej i /lub ręczne zdefiniowanie obsługi edycji wybranej danej.
Weryfikacja elementów UI strony</td><td rowspan="2">Nagranie obsługi wyświetlania wybranej danej i /lub ręczne zdefiniowanie obsługi wyświetlania takiej danej.
Dokonanie weryfikacji zawartości wyświetlonej danej</td></tr>
<tr><td>2</td><td>2 pary stron</td></tr>
</table>

4. Wykonanie zestawu testów typu ***Test Suite***, który powinien zawierać testy typu ***Test Case*** wykonane w p.3. – na podstawie przykładu budowy zestawu testów ***TestySelenium_WypozyczalniKsiazek*** w **Dodatku 2.**

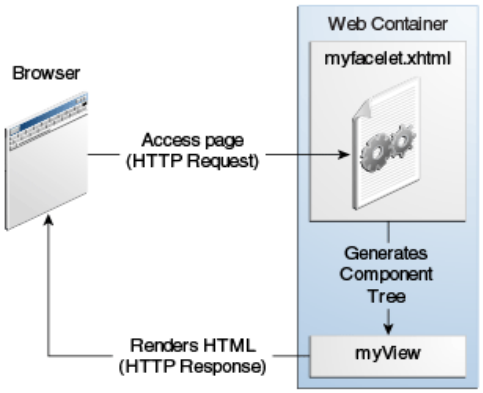
Dodatek 1

Przykład tworzenia aplikacji internetowej wg zasady „database-first development”.

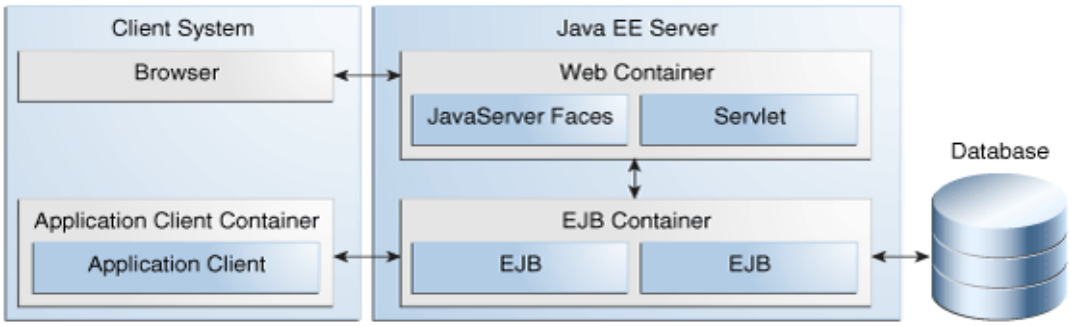
1. **Krótka informacja dotycząca wielowarstwowej aplikacji internetowej na podstawie tutorialu Java EE 8 ze strony** [Java Platform, Enterprise Edition (Java EE) 8. The Java EE Tutorial]



Model wielowarstwowej aplikacji na platformie ***Java EE***, gdzie każda z warstw jest zbudowana z różnych typów komponentów.



Komunikacja “klient – strony ***JavaServer Faces***” aplikacji internetowej typu ***JSF*** platformy ***Java EE***.



Kontenery na platformie ***Java EE*** umożliwiające obsługę różnego typu komponentów przez serwer aplikacji ***Java EE Server***

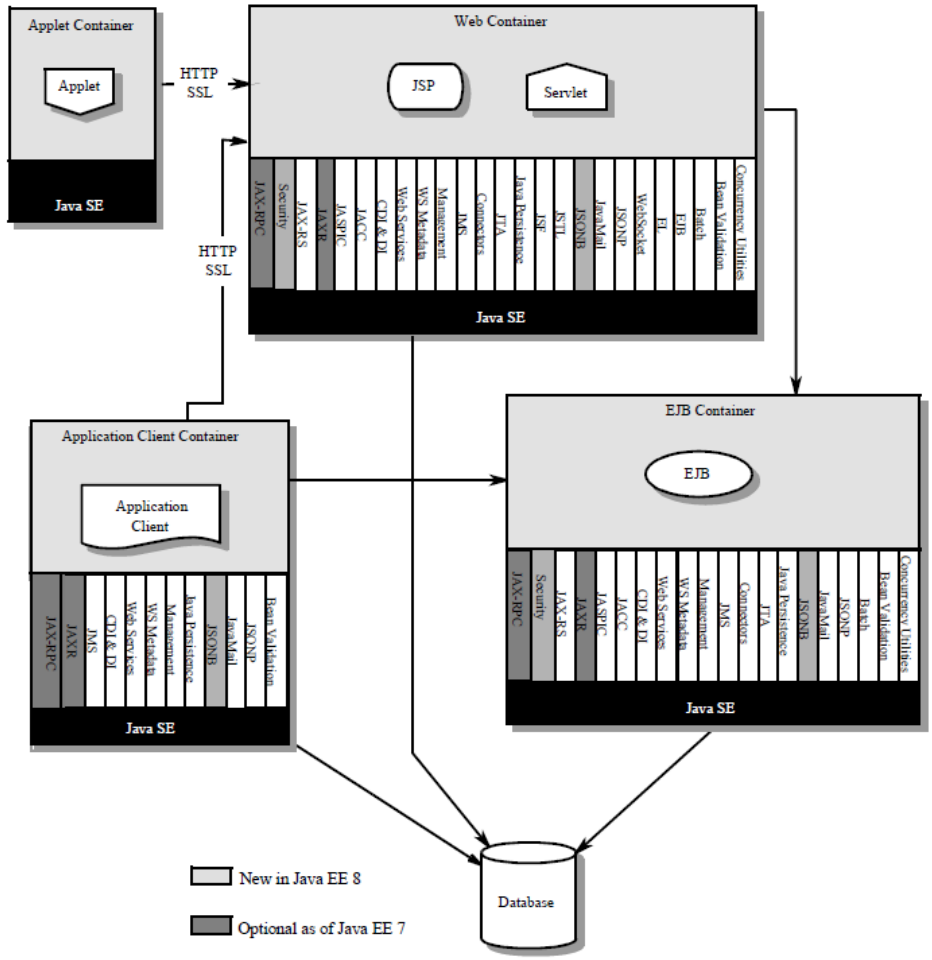


Figure EE.2-1 Java EE Architecture Diagram

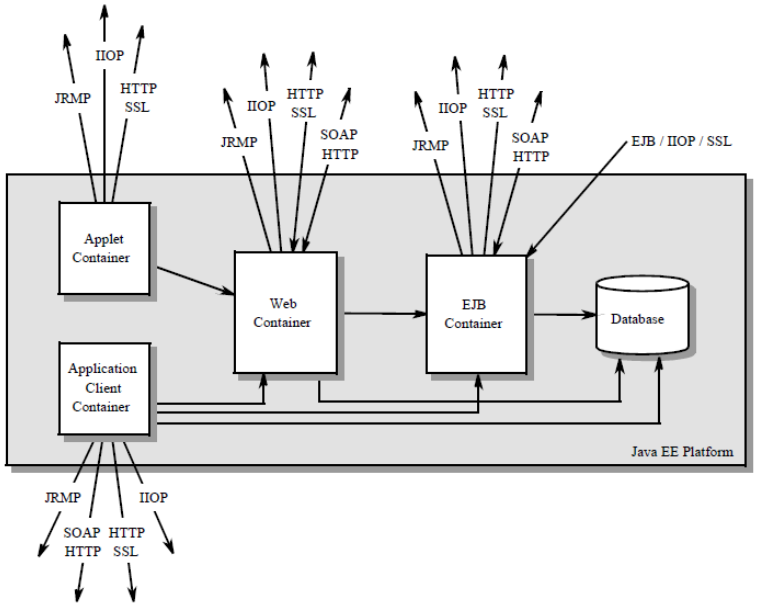


Figure EE.2-2 Java EE Interoperability

Przegląd kontenerów z poszczególnych warstw aplikacji działającej na platformie Java EE 8 wg https://download.oracle.com/otn-pub/jcp/java_ee-8-final-eval-spec/JavaEE_Platform_Spec.pdf

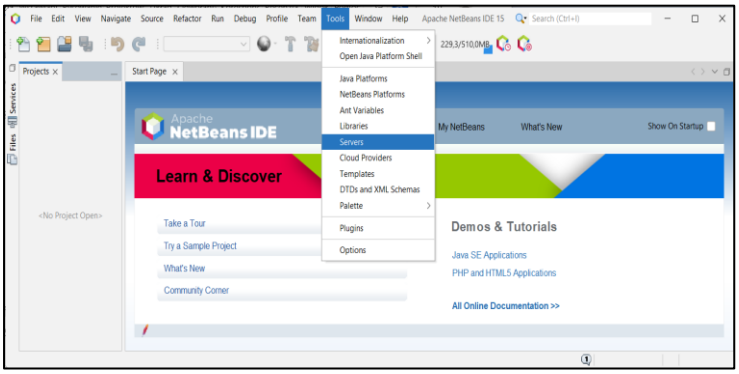
2. Instalacja *Apache Netbeans 18*

Należy zainstalować aktualną wersję narzędzia ***Apache NetBeans 18*** działającego na wybranej platformie ***Java SE 17***. Informacja o instalacji ***Apache NetBeans 18 i Java SE: 8, 17 oraz GlassFish 5.0.1*** została podana na stronie przedmiotu: http://zofia.kruczkiewicz.staff.iiar.pwr.wroc.pl/index.php?id=INEK011.

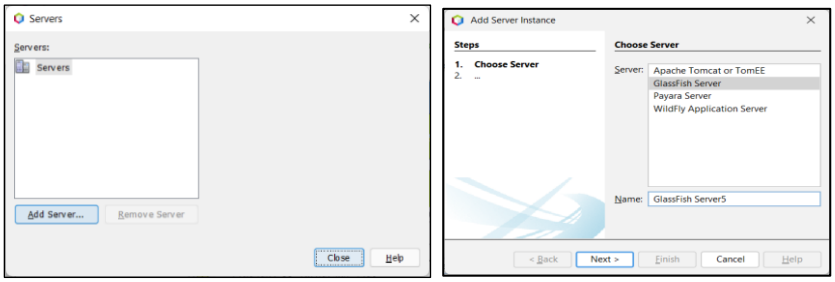
Testowana aplikacja, utworzona w środowisku ***Apache NetBeans 18***, jest aplikacją działającą na platformie ***Java EE 8*** i powinna działać z wykorzystaniem w projekcie ***Javy SE 8,*** niezależnie od domyślnej wersji Javy SE w narzędziu ***Apache Netbeans 18*** (***Java SE 17***) **–** sposób utworzenia tej aplikacji pokazano w p.3.

Po instalacji narzędzia ***Apache Netbeans 18*** należy dodatkowo zainstalować serwer ***GlassFish 5.0.1*** w następujący sposób:

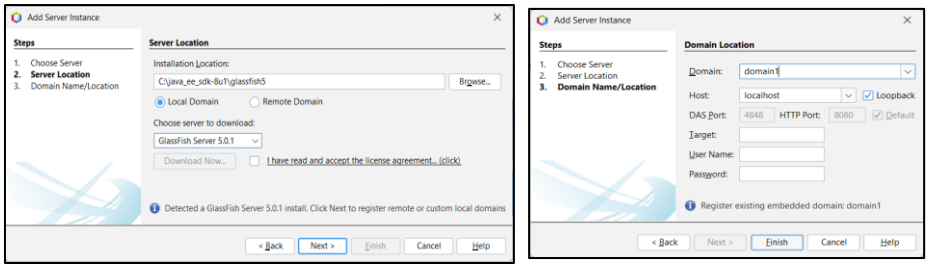
- w zakładce ***Tools*** należy wybrać pozycję ***Servers***



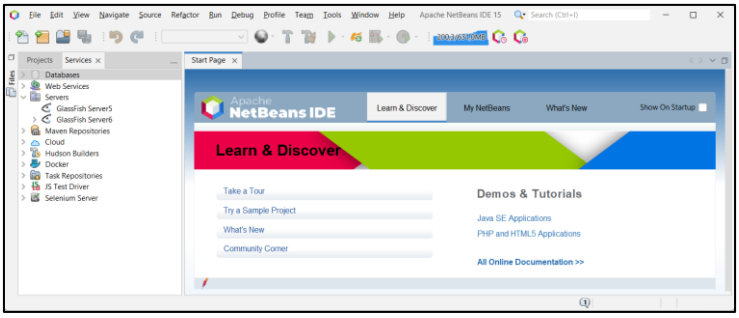
- w formularzu instalacji serwerów należy kliknąć na przycisk: ***Add Server***
- następnie należy wybrać z listy pozycję ***GlassFish Server*** i w polu ***Name*** należy mu nadać nazwę ***GlassFish Server5***.



- Po kliknięciu przycisku ***Next*** należy wybrać katalog z pobranym serwerem za pomocą przycisku ***Browse***, następnie z listy wybrać właściwą wersje instalowanego serwera (***Choose server to download***) i zaaceptować licencję. Po kliknięciu na przycisk ***Next*** pojawi się formularz prezentujący utworzenie katalogu z domeną serwera. Po kliknięciu na przycisk ***Finish*** kończy się instalacja serwera

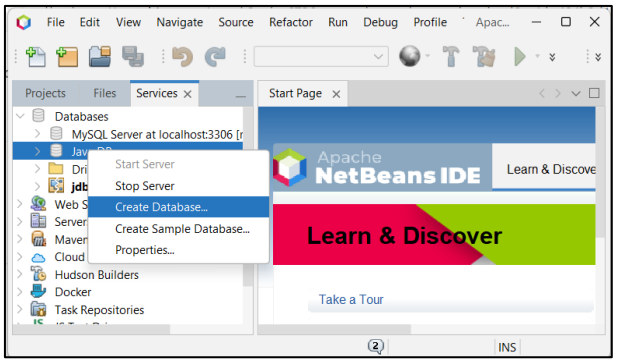


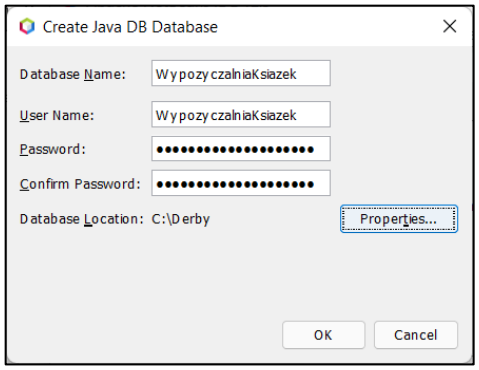
- Po kliknięciu na zakładkę ***Services*** pojawił w pozycji ***Servers*** zainstalowany ***GlassFish Server 5.0.1***.



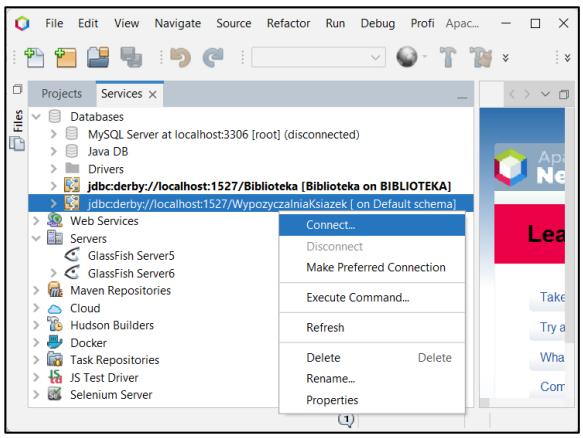
3. Przebieg tworzenia wielowarstwowej aplikacji internetowej w technologii *JavaServer Faces* w środowisku *Apache NetBeans IDE 18* (wg http://netbeans.org/kb/docs/web/jsf20-crud.html) – od 1.09.2023 również dla *Apache NetBeans IDE 19).*

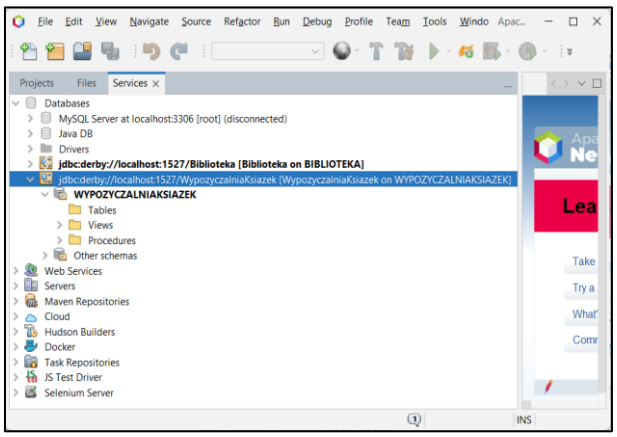
3.1. Zakładanie pustej bazy danych w systemie baz danych ***Derby***: w zakładce ***Services*** w pozycji ***Databases*** kliknąć prawym klawiszem myszy na pozycję ***Java DB*** i następnie wybrać pozycję ***Create Database***. W formularzu ***Create Java DB Database*** należy podać- ***Database Name***: WypozyczalniaKsiazek, ***User Name***: WypozyczalniaKsiazek, ***Password***: WypozyczalniaKsiazek i kliknąć na przycisk ***OK.***





Należy połączyć się z wykonaną, pustą bazą danych (poniżej): w zakładce ***Services/Databases*** kliknąć prawym klawiszem myszy na pozycję reprezentującą połączenie z utworzoną bazą danych i wybrać pozycję ***Connect***.

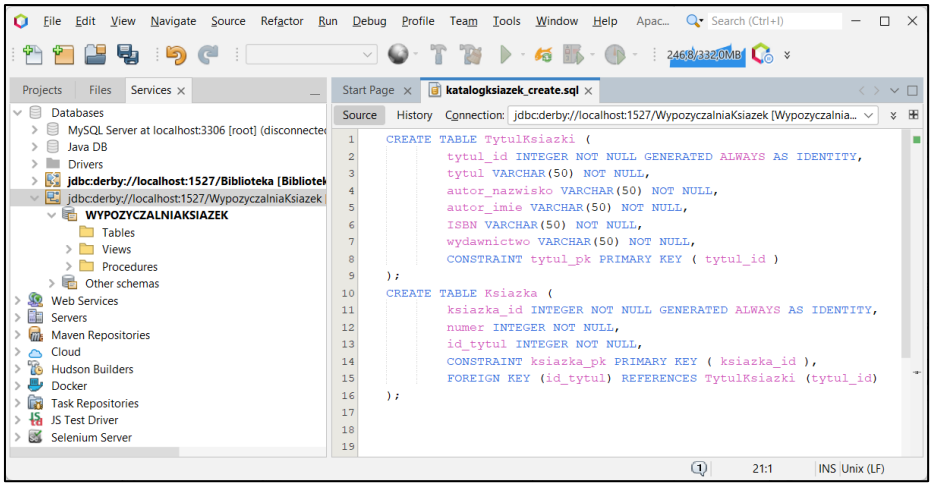




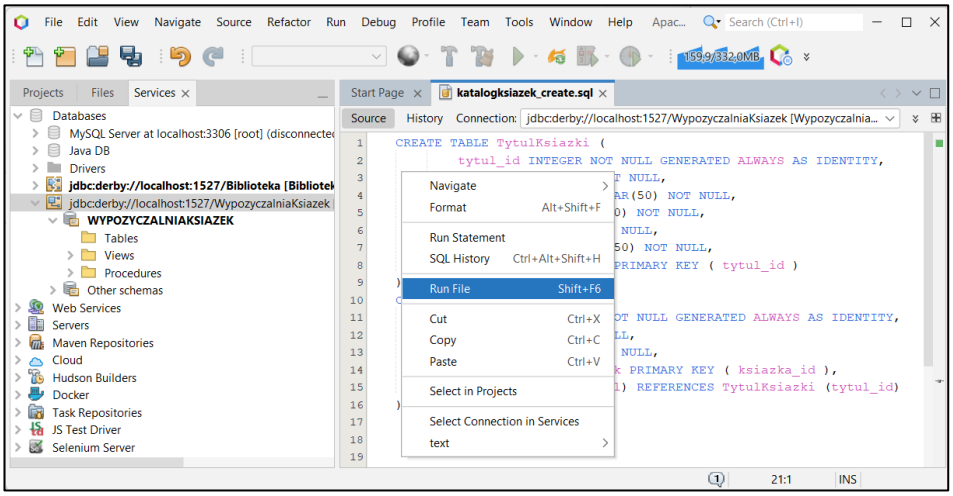
3.2. W celu utworzenia tabel relacyjnej bazy danych zdefiniowano skrypt w języku SQL ***katalogksiazek_create.sql*** – w przykładzie skrypt utworzy dwie tabele: ***TytulKsiazki*** oraz ***Ksiazka*** powiązane relacją 1..*. Przykład ten nawiązuje do przykładu z Instrukcji 1 do lab1 (diagram klas z p.5.2 instrukcji).

```
CREATE TABLE TytulKsiazki (
    tytul_id INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY,
    tytul VARCHAR(50) NOT NULL,
    autor_nazwisko VARCHAR(50) NOT NULL,
    autor_imie VARCHAR(50) NOT NULL,
    ISBN VARCHAR(50) NOT NULL,
    wydawnictwo VARCHAR(50) NOT NULL,
    CONSTRAINT tytul_pk PRIMARY KEY ( tytul_id )
);
CREATE TABLE Ksiazka (
    ksiazka_id INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY,
    numer INTEGER NOT NULL,
     id_tytul INTEGER NOT NULL,
    CONSTRAINT ksiazka_id PRIMARY KEY ( ksiazka_id ),
   FOREIGN KEY (id_tytul) REFERENCES TytulKsiazki (tytul_id)
);
```

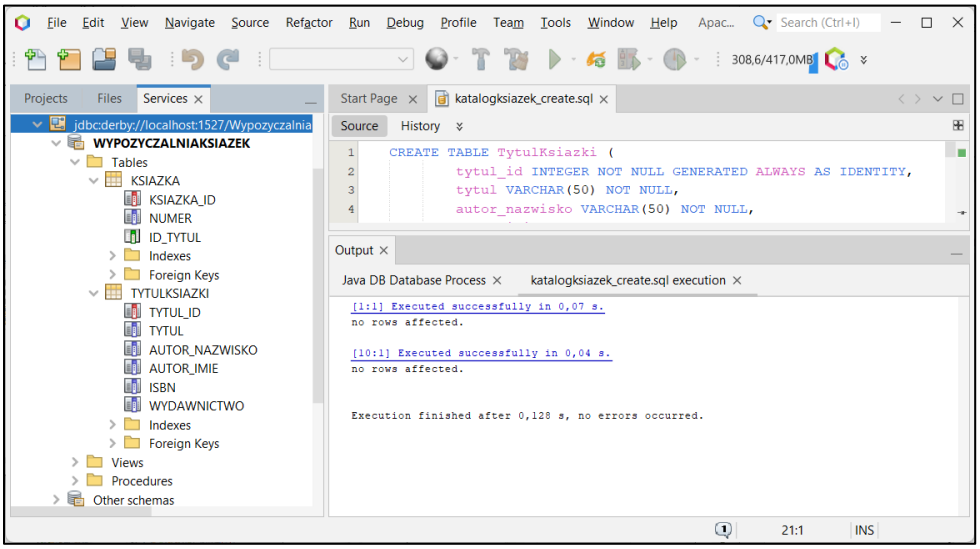
Utworzenie tabel relacyjnej bazy danych - utworzony skrypt należy uruchomić w następujący sposób: w ***Menu Bar*** należy kliknąć na ***File***, a następnie na ***Open File….*** wybrać utworzony wcześniej plik w formacie ***sql***. Następnie, w oknie edytora otworzonego pliku należy w zakładce ***Connection*** wybrać z listy połączeń połączenie do utworzonej bazy danych ***WypozyczalniaKsiazek.***



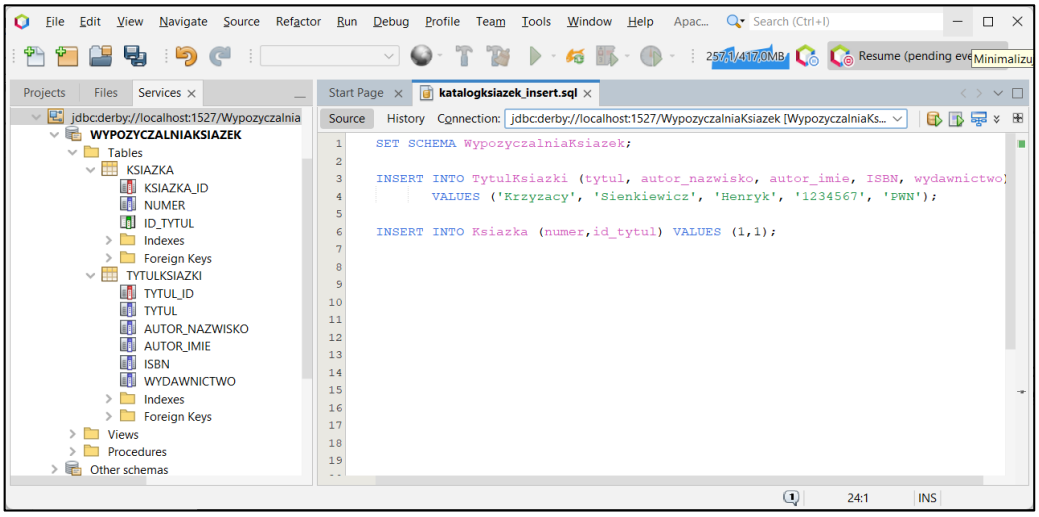
Następnie, należy kliknąć prawym klawiszem myszy na powierzchnię okna edytora pliku i następnie kliknąć na pozycję ***Run File*** w celu uruchomienia skryptu ***katalogksiazek_create.sql***.



Poniżej pokazano utworzone dwie tabele: ***KSIAZKA*** oraz ***TYTULKSIAZKI*** w bazie danych ***WypozyczalniaKsiazek.***



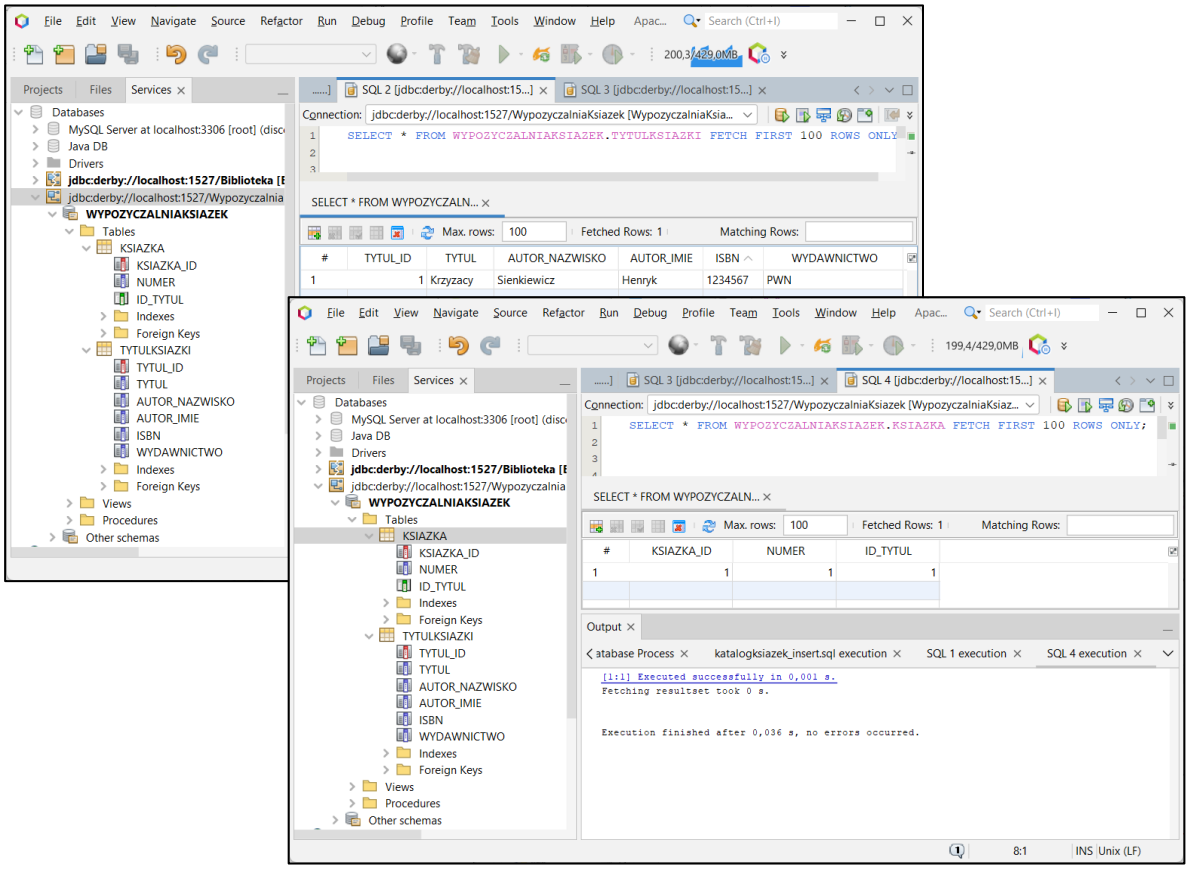
Poniżej pokazano w oknie edytora pliku wczytany drugi skrypt ***katalogksiazek_insert.sql,*** podobnie jak skrypt ***katalogksiazek_create.sql.*** Należy go uruchomić w taki sam sposób jak skrypt ***katalogksiazek_create.sql*** w celu wstawienia po jednej krotce do każdej z dwóch tabel.



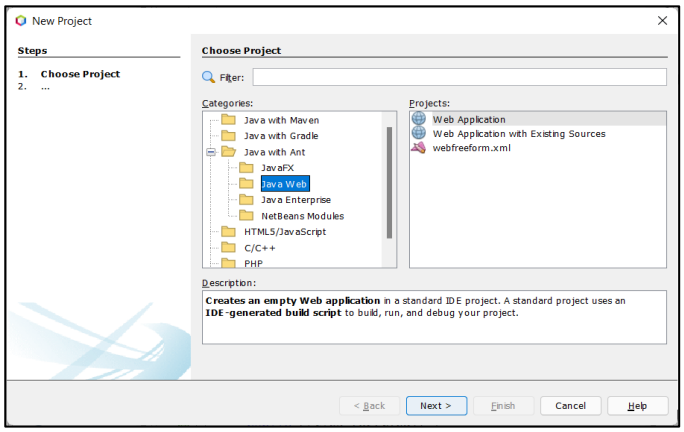
Poniżej opisano utworzone tabele oraz możliwe do utworzenia klasy typu ***Entity.***

Tabela	Entity	Opis
Tytulksiazki	Tytulksiazki	Przechowują dane tytułu książki; relacja 1 do wiele z tabelą (encją) Ksiazka
Ksiazka	Ksiazka	Przechowują numer książki Relacja wiele do jeden z tabelą (encją) Tytulksiazki

W celu odczytania zawartości tabel należy w oknie zakładki ***Service*** (jak poniżej pokazano) kliknąć prawym klawiszem myszy kolejno na każdą z tabel i wybrać pozycję ***View Data…***, co uruchamiania polecenie ***SELECT * FROM*** i pozwala wyświetlić zawartość tabeli.

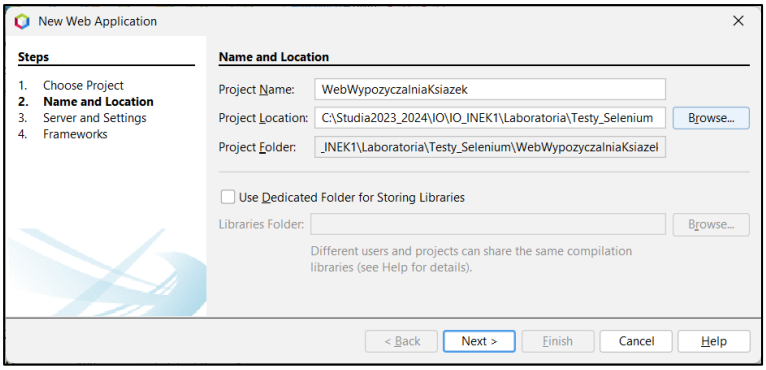


3.3. Tworzenie aplikacji typu ***Web Application*** – wybrać kolejno ***File/New Project/Java Web/Web Application/Next.***

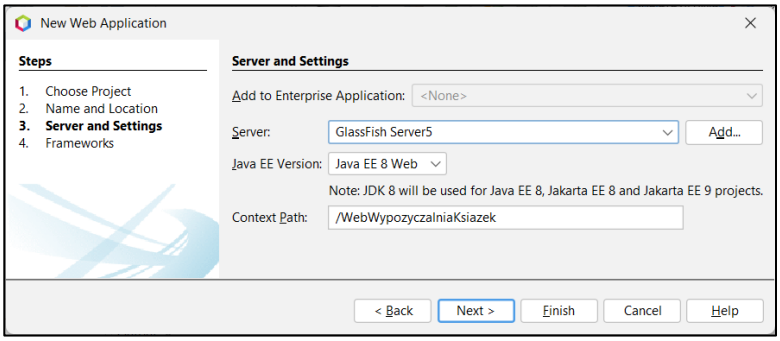


Poniżej pokazano nadanie nazwy projektowi ***WebWypozyczalniaKsiazek*** oraz wybór położenia projektu. W celu kontynuacji należy kliknąć na przycisk ***Next***.

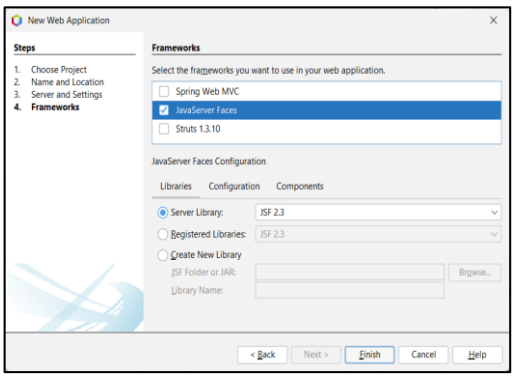
Uwaga: W tym samym katalogu kolejna wersja programu typu *Web Application* musi mieć zmienioną nazwę!

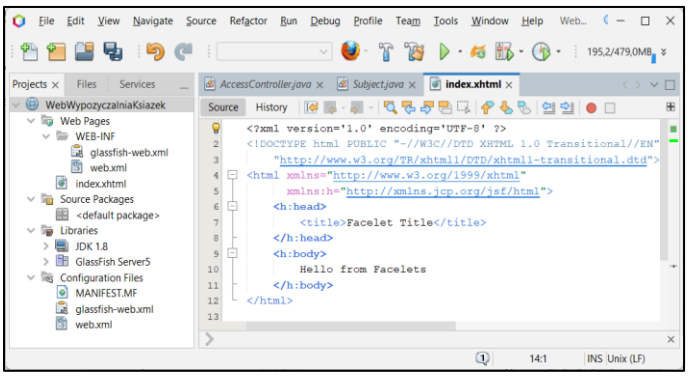


Poniżej pokazano wybór serwera aplikacji, platformy ***Java EE 8 Web*** oraz względnej ścieżki do projektu. W celu kontynuacji należy kliknąć na przycisk ***Next***.

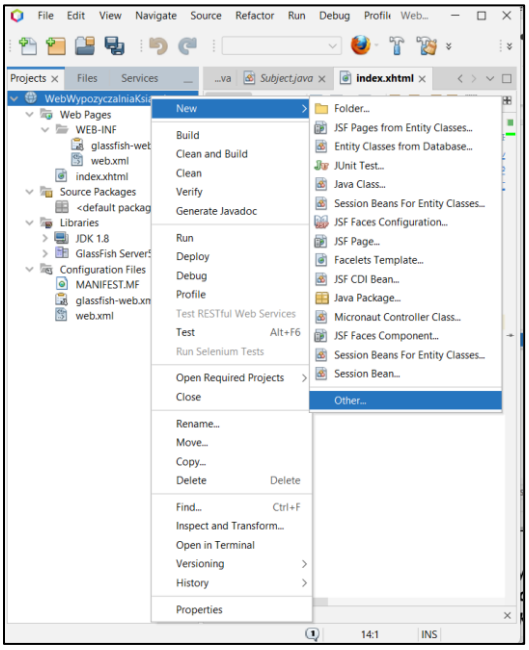


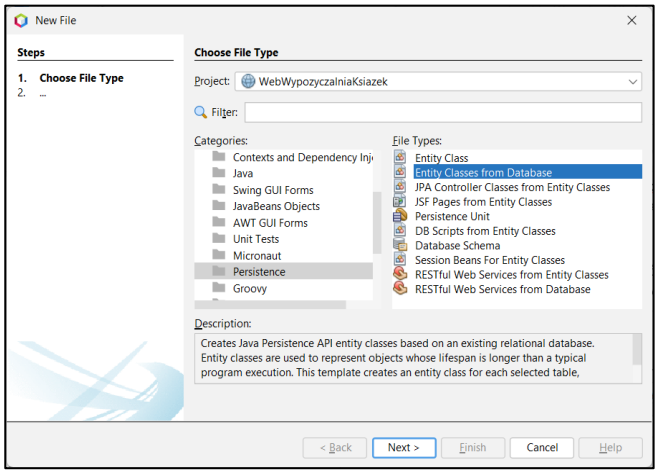
Poniżej pokazano wybór technologii ***Java Server Faces***. Po kliknięciu na przycisk ***Finish*** zostanie wykonany projekt ***Java Web Application*** o nazwie ***WebWypozyczalniaKsiazek***



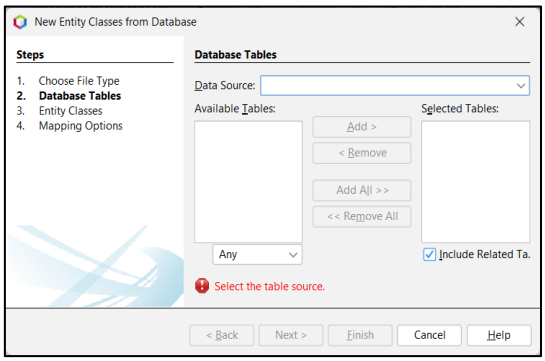


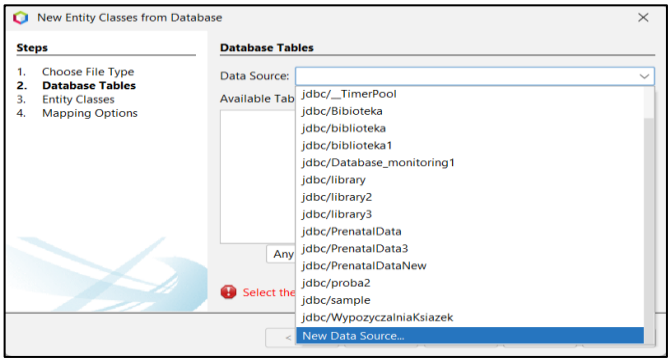
3.4. Wykonanie klas typu ***Entity*** z bazy danych ***WypozyczalniaKsiazek*** – w tym celu należy wybrać pozycję ***New*** z listy po kliknięciu prawym klawiszem myszy na nazwę projektu w zakładce ***Projects***, następnie wybór z kolejnej listy pozycji ***Other.*** W formularzu ***New File*** (poniżej) należy wybrać: ***Persistence/Entity Classes from Database.***

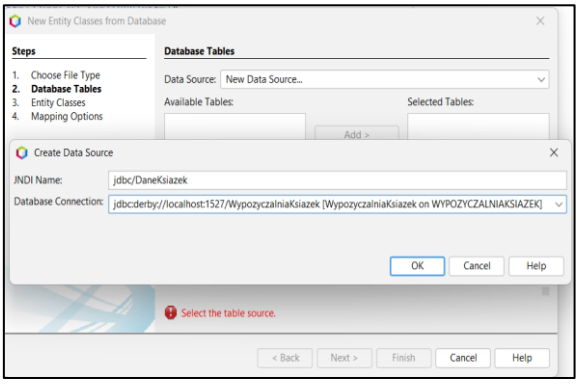


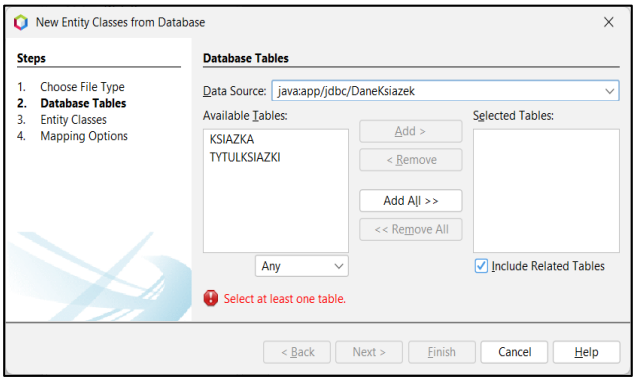


Wykonanie klas typu ***Entity*** z bazy danych – wybór w polu ***Data Source*** pozycji ***New Data Source*** i zdefiniowanie połączenia typu ***JNDI*** w polu ***JNDI Name: jdbc/DaneKsiazek*** do bazy danych ***WypozyczalniaKsiazek.*** W polu ***Database Connection*** należy wybrać połączenie z utworzoną bazą danych: ***jdbc:derby://localhost:1527/WypozyczalniaKsiazek*** i zatwierdzić przyciskiem **OK**. Z listy ***Available Tables*** wybrać obie tabele ***KSIAZKA*** oraz ***TYTULKSIAZKI*** za pomocą przycisku ***Add All>> .*** Następnie należy kliknąć na klawisz ***Next.***

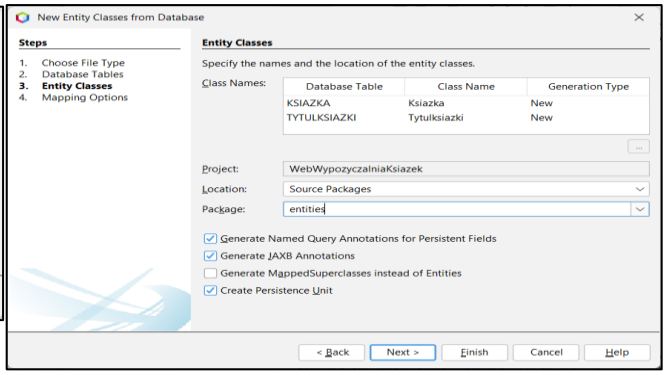


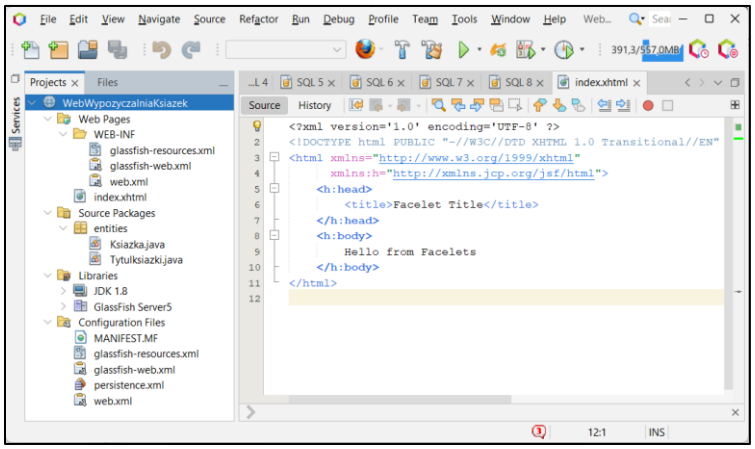




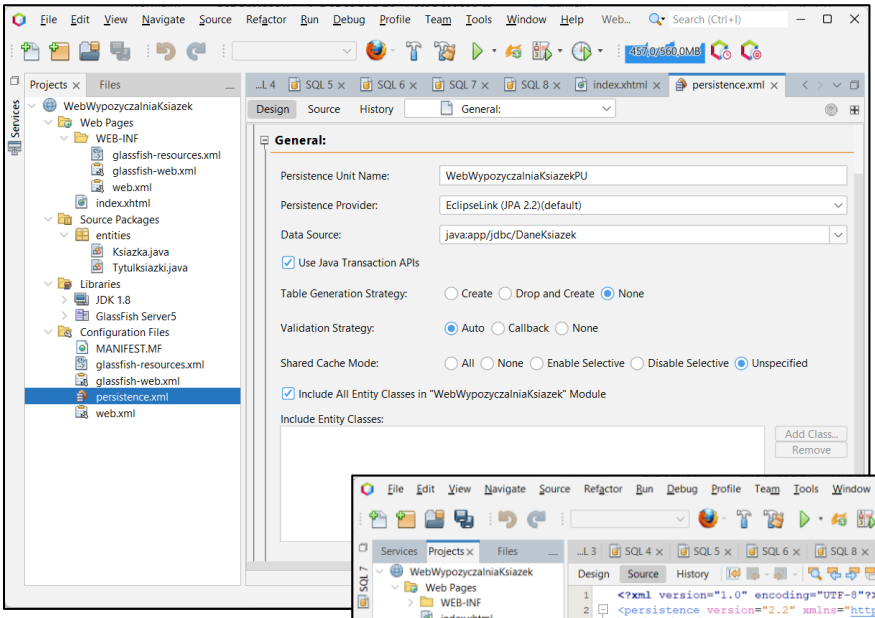


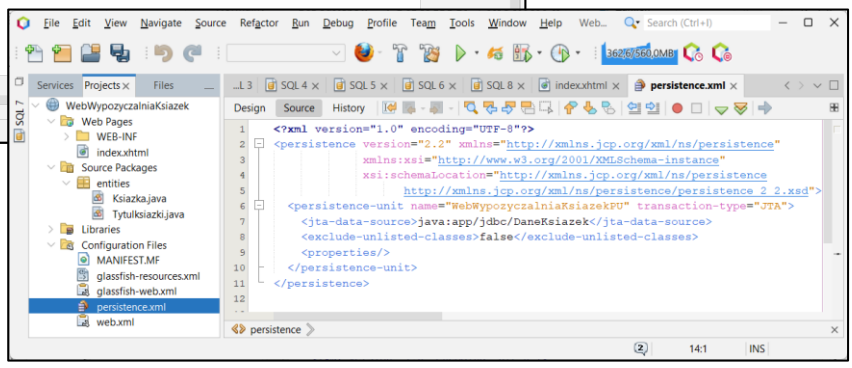
Poniżej kolejne formularze prezentują wykonanie klas typu ***Entity*** na podstawie wybranych wcześnie tabel z bazy danych. Należy w polu ***Package*** wpisać nazwę pakietu (np. ***entities***), w którym będą przechowywane utworzone klasy. Na koniec należy kliknąć na przycisk ***Finish*** – zostaną wygenerowane klasy typu ***Entity*** odwzorowane z podanych tabel.





Plik persistence.xml zawiera informacje dla komponentu ***EntityManager*** dotyczące obsługi obiektów klas typu ***Entity*** w technologii ***JPA***.





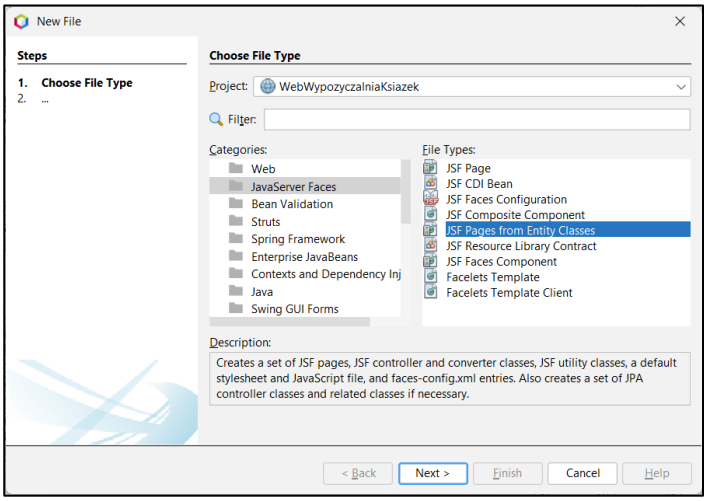
Kod wygenerowanych klas typu ***Entity: Tytulksiazki*** oraz ***Ksiazka***

```
package entities;
import java.io.Serializable;
import java.util.Collection;
import javax.persistence.Basic;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;
@Entity
@Table(name = "TYTULKSIAZKI")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Tytulksiazki.findAll", query = "SELECT t FROM Tytulksiazki t"),
    @NamedQuery(name = "Tytulksiazki.findByTytulId", query = "SELECT t FROM Tytulksiazki t WHERE t.tytulId = :tytulId"),
    @NamedQuery(name = "Tytulksiazki.findByTytul", query = "SELECT t FROM Tytulksiazki t WHERE t.tytul = :tytul"),
    @NamedQuery(name = "Tytulksiazki.findByAutorNazwisko", query = "SELECT t FROM Tytulksiazki t WHERE t.autorNazwisko =
      :autorNazwisko"),
    @NamedQuery(name = "Tytulksiazki.findByAutorImie", query = "SELECT t FROM Tytulksiazki t WHERE t.autorImie = :autorImie"),
    @NamedQuery(name = "Tytulksiazki.findByIsbn", query = "SELECT t FROM Tytulksiazki t WHERE t.isbn = :isbn"),
    @NamedQuery(name = "Tytulksiazki.findByWydawnictwo", query = "SELECT t FROM Tytulksiazki t WHERE t.wydawnictwo =
      :wydawnictwo")})
public class Tytulksiazki implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "TYTUL_ID")
        private Integer tytulId;
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 50)
    @Column(name = "TYTUL")
        private String tytul;
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 50)
    @Column(name = "AUTOR_NAZWISKO")
         private String autorNazwisko;
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 50)
        private String autorImie;
    @Basic(optional = false)
    @NotNull
    @Column(name = "AUTOR_IMIE")
    @Size(min = 1, max = 50)
    @Column(name = "ISBN")
        private String isbn;
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 50)
    @Column(name = "WYDAWNICTWO")
        private String wydawnictwo;
  @OneToMany(cascade = CascadeType.ALL, mappedBy = "idTytul")
```

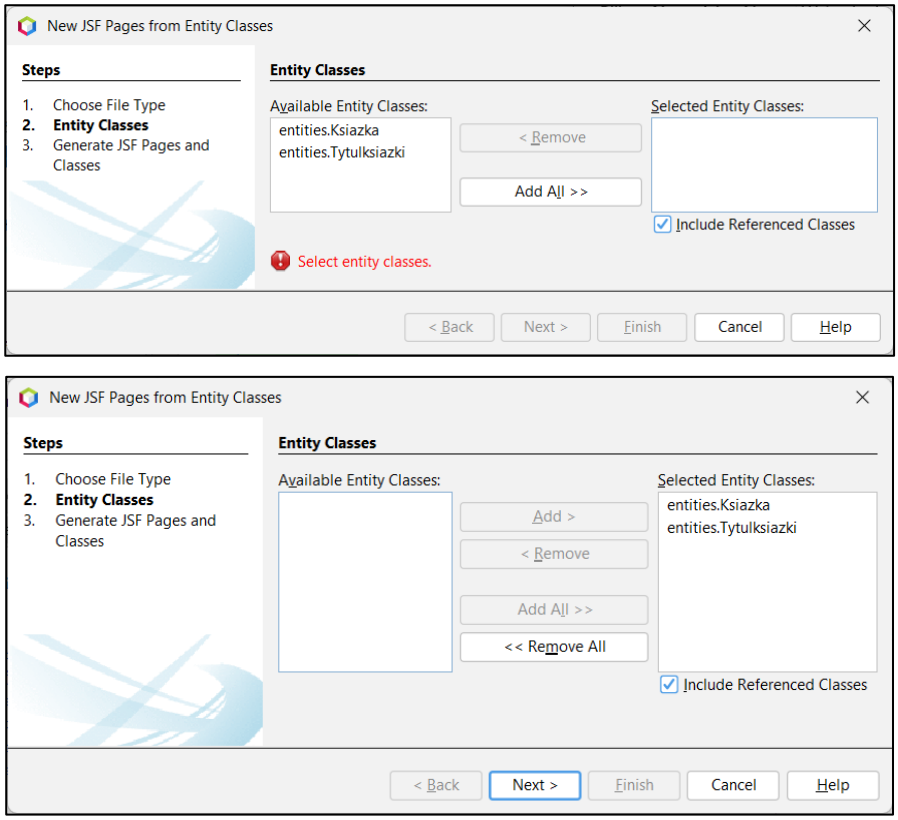
```
        private Collection<Ksiazka> ksiazkaCollection;

        public Tytulksiazki() {    }
        public Tytulksiazki(Integer tytulId) {   this.tytulId = tytulId;    }
        public Tytulksiazki(Integer tytulId, String tytul, String autorNazwisko, String autorImie, String isbn, String wydawnictwo)
                {this.tytulId = tytulId;          this.tytul = tytul;     this.autorNazwisko = autorNazwisko;
                this.autorImie = autorImie;       this.isbn = isbn;       this.wydawnictwo = wydawnictwo;   }
        public Integer getTytulId() {    return tytulId;  }
        public void setTytulId(Integer tytulId) {    this.tytulId = tytulId;  }
        public String getTytul() {  return tytul;}
        public void setTytul(String tytul) { this.tytul = tytul;  }
        public String getAutorNazwisko() { return autorNazwisko;   }
        public void setAutorNazwisko(String autorNazwisko) {  this.autorNazwisko = autorNazwisko; }
        public String getAutorImie() { return autorImie; }
        public void setAutorImie(String autorImie) {  this.autorImie = autorImie;  }
        public String getIsbn() { return isbn;  }
        public void setIsbn(String isbn) {       this.isbn = isbn; }
        public String getWydawnictwo() {  return wydawnictwo;  }
        public void setWydawnictwo(String wydawnictwo) { this.wydawnictwo = wydawnictwo;  }
    @XmlTransient
    public Collection<Ksiazka> getKsiazkaCollection() { return ksiazkaCollection;}
          public void setKsiazkaCollection(Collection<Ksiazka> ksiazkaCollection) {
                this.ksiazkaCollection = ksiazkaCollection; }
    @Override
        public int hashCode() {
                 int hash = 0;
                hash += (tytulId != null ? tytulId.hashCode() : 0);
                return hash; }
    @Override
        public boolean equals(Object object) {
            if (!(object instanceof Tytulksiazki)) {  return false;   }
             Tytulksiazki other = (Tytulksiazki) object;
             if ((this.tytulId == null && other.tytulId != null) || (this.tytulId != null && !this.tytulId.equals(other.tytulId)))
                return false;
            return true; }
    @Override
        public String toString() {
           return "entities.Tytulksiazki[ tytulId=" + tytulId + " ]";   }
}

package entities;
import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.validation.constraints.NotNull;
import javax.xml.bind.annotation.XmlRootElement;
@Entity
@Table(name = "KSIAZKA")
@XmlRootElement
@NamedQueries({
```

```
  @NamedQuery(name = "Ksiazka.findAll", query = "SELECT k FROM Ksiazka k"),
  @NamedQuery(name = "Ksiazka.findByKsiazkaId", query = "SELECT k FROM Ksiazka k WHERE k.ksiazkaId = :ksiazkaId"),
  @NamedQuery(name = "Ksiazka.findByNumer", query = "SELECT k FROM Ksiazka k WHERE k.numer = :numer")})
public class Ksiazka implements Serializable {
  private static final long serialVersionUID = 1L;
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  @Basic(optional = false)
  @Column(name = "KSIAZKA_ID")
        private Integer ksiazkaId;
  @Basic(optional = false)
  @NotNull
  @Column(name = "NUMER")
        private int numer;
  @JoinColumn(name = "ID_TYTUL", referencedColumnName = "TYTUL_ID")
  @ManyToOne(optional = false)
        private Tytulksiazki idTytul;
        public Ksiazka() {  }
        public Ksiazka(Integer ksiazkaId)                    {   this.ksiazkaId = ksiazkaId;  }
        public Ksiazka(Integer ksiazkaId, int numer)         {   this.ksiazkaId = ksiazkaId;  this.numer = numer;  }
        public Integer getKsiazkaId()                        {   return ksiazkaId;  }
        public void setKsiazkaId(Integer ksiazkaId)          {   this.ksiazkaId = ksiazkaId;  }
        public int getNumer()                                {   return numer;  }
        public void setNumer(int numer)                      {   this.numer = numer;  }
        public TytulKsiazki getIdTytul()                     {   return idTytul;  }
        public void setIdTytul(TytulKsiazki idTytul)         {   this.idTytul = idTytul;  }
    @Override
        public int hashCode() {
          int hash = 0;
          hash += (ksiazkaId != null ? ksiazkaId.hashCode() : 0);
          return hash;  }
    @Override
        public boolean equals(Object object) {
          if (!(object instanceof Ksiazka))     return false;
          Ksiazka other = (Ksiazka) object;
          if ((this.ksiazkaId == null && other.ksiazkaId != null)||(this.ksiazkaId != null && !this.ksiazkaId.equals(other.ksiazkaId)))
            return false;
          return true;  }
     @Override
        public String toString() {  return "entities.Ksiazka[ ksiazkaId=" + ksiazkaId + " ]";  }
}
```

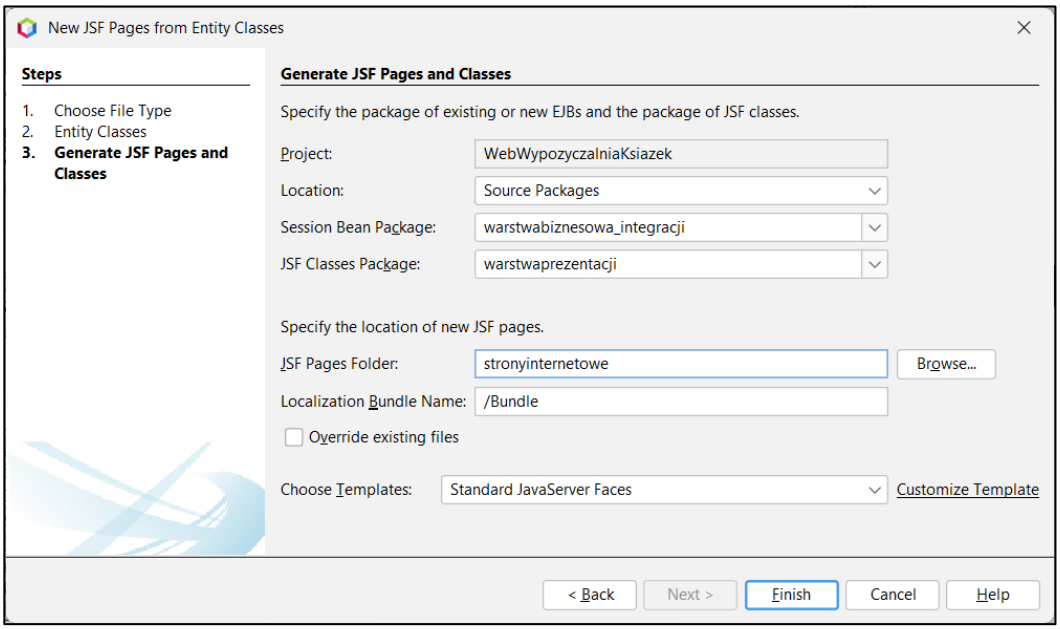
3.5. Generowanie ***stron JSF*** z encji (oraz komponentów EE) – wybór ***New*** z listy po kliknięciu prawym klawiszem myszy na nazwę projektu w zakładce ***Projects***, następnie wybór pozycji ***Other*** z kolejnej listy. W formularzu ***New File*** należy wybrać pozycje ***JavaServer Faces/JSF Pages from Entity Classes*** **i** kliknąć na przycisk ***Next.***



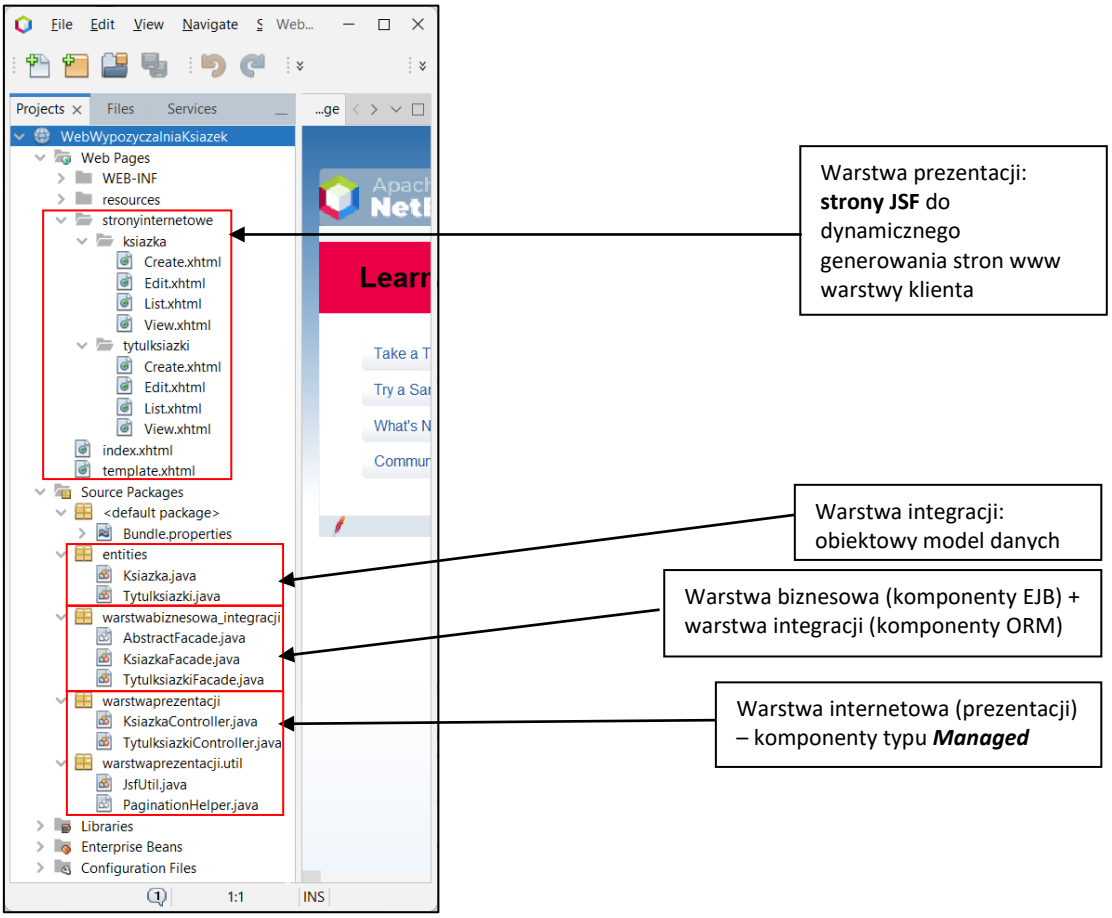
Na kolejnym formularzu należy wybrać z listy ***Available Entity Classes*** wszystkie klasy za pomocą klawisza ***Add All*** i następnie kliknąć na klawisz ***Next.***



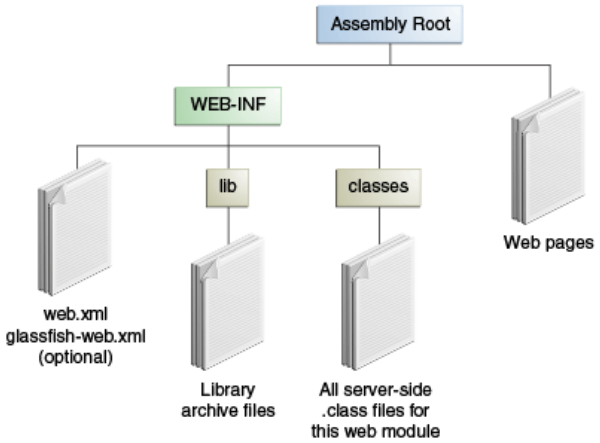
Poniżej przedstawiono widok formularza ***New JSF Pages from Entity Classes*** po wyborze encji i kliknięciu na przycisk ***Next***. Na tym formularzu należy w polu ***Session Bean Package*** podać nazwę pakietu zawierającego kod realizujący funkcje ***Warstwy biznesowej***, który jednocześnie wykonuje funkcje ***Warstwy integracji***, w polu ***JSF Classes Package*** podać nazwę pakietu realizującego funkcje ***Warstwy prezentacji*** lub inaczej ***Warstwy internetowej***, a w polu ***JSF Pages Folder*** nazwę folderu związaną z ***Warstwą internetową*** lub inaczej – z ***Warstwą prezentacji***. Następnie należy kliknąć na przycisk ***Finish***.

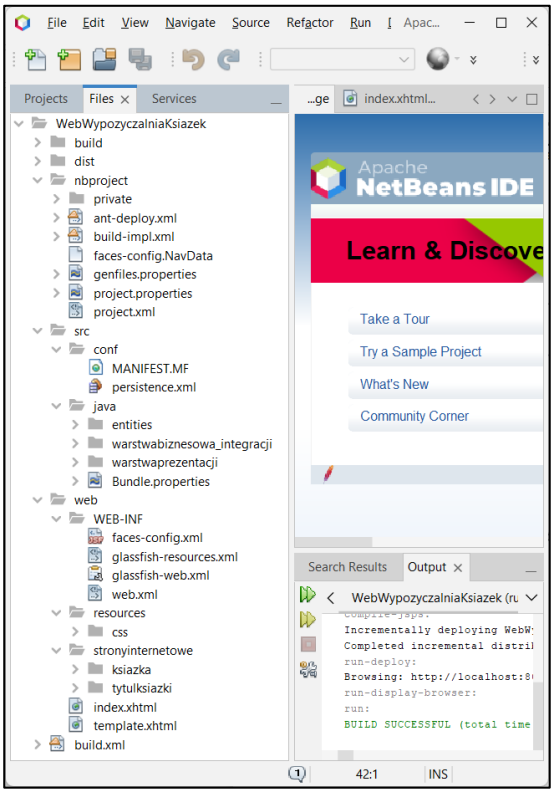


Poniżej pokazano zawartość katalogu wykonanego projektu aplikacji internetowej typu ***Web Application.***

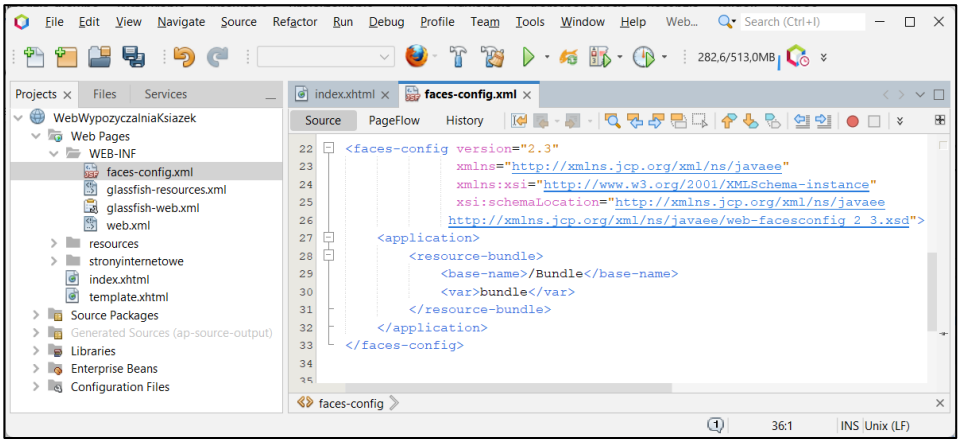


Poniżej przedstawiono schemat struktury aplikacji typu ***Java Web Application.***

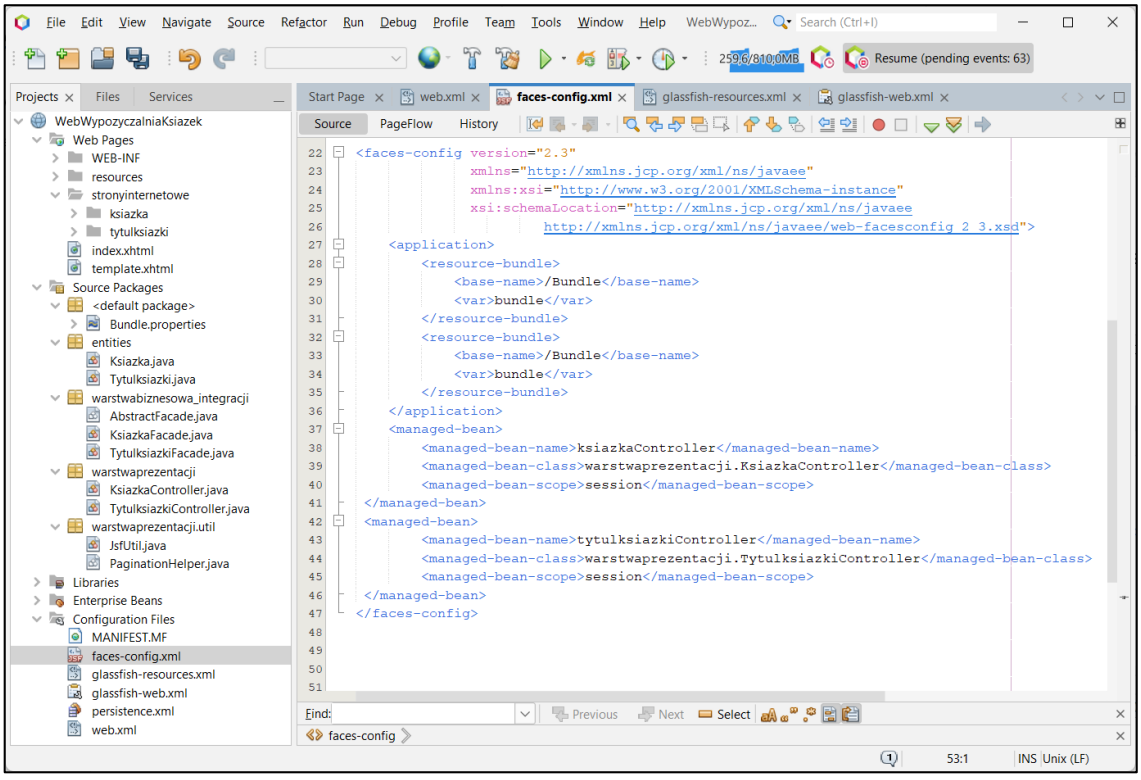




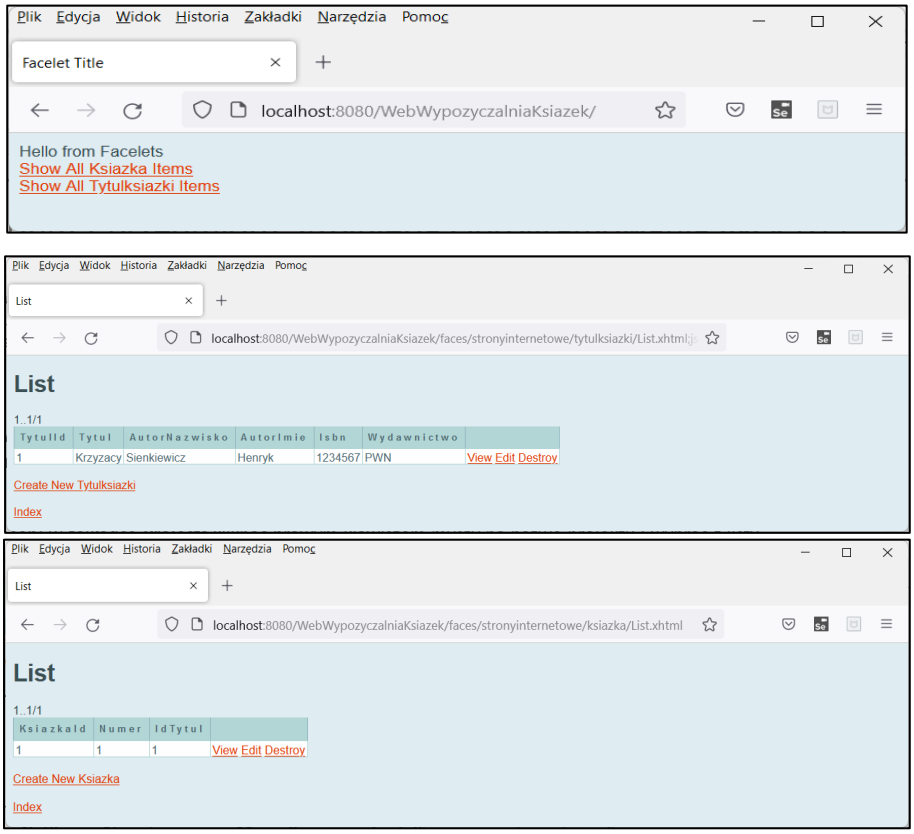
3.6. Należy uzupełnić zawartość pliku konfiguracyjnego typu **faces-config.xml** znacznikami umożliwiającymi konfigurację komponenetów typu **Managed Bean** z *Warstwy prezentacji* wg [Configuring Managed Beans (javaee.github.io)]



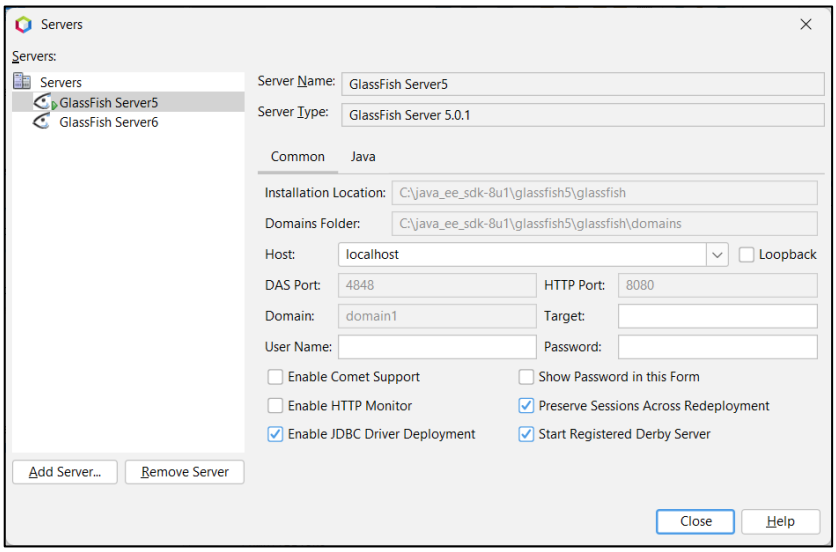

```
<managed-bean>
    <managed-bean-name>ksiazkaController</managed-bean-name>
    <managed-bean-class>warstwaprezentacji.KsiazkaController</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
<managed-bean>
    <managed-bean-name>tytulksiazkiController</managed-bean-name>
    <managed-bean-class>warstwaprezentacji.TytulksiazkiController</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```



3.7. Widok aplikacji internetowej po jej uruchomieniu w następujący sposób: w zakładce ***Projects*** kliknąć prawym klawiszem myszy na nazwę projektu i wybrać z listy kolejno ***Clean and Build/Deploy.*** Następnie należy otworzyć okno w przeglądarce ***FireFox Mozilla*** i wpisać następujący adres strony www: **http://localhost:8080/WebWypozyczalniaKsiazek/**. Innym sposobem uruchomienia aplikacji w oknie przeglądarki typu ***FireFox*** jest przypisanie do niej domyślnego dostępu w środowisku **Apache *NetBeans 18***: należy w oknie ***Tools/Options,*** w polu ***WebBrowser*** przypisać wybraną przeglądarkę ***FireFox*** z listy przeglądarek. Wtedy podczas uruchamiania aplikacji typu ***Web Application*** wykonać ***Clean and Build/Run*** (lub ***Clean and Build/Deploy/Run*** w przypadku problemów z uruchomieniem strony głównej aplikacji*).* Aplikacja zostanie automatycznie uruchomiona w przeglądarce **FireFox**.



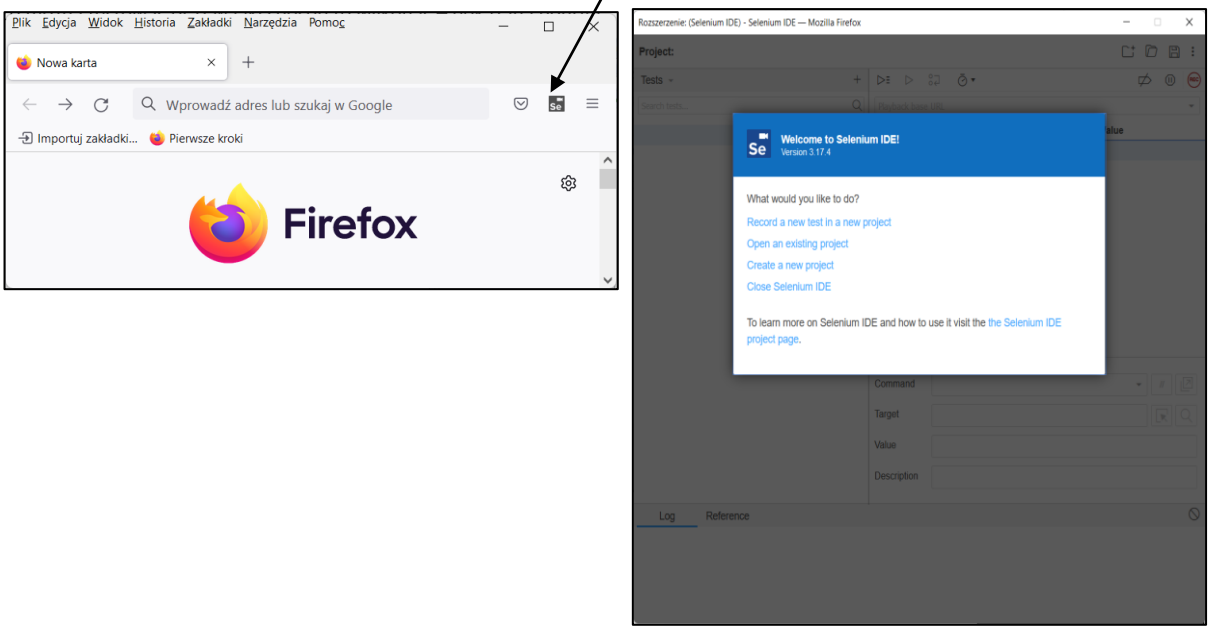
Port 8080 jest portem domyślnym serwera typu **GlassFish**. Należy go sprawdzić, przed uruchomieniem aplikacji, w środowisku **Apache NetBeans *18*** wybierając w ***Menu Bar*** opcje ***Tools/Servers*** i na formularzu ***Servers:*** z pozycji **GlassFish Server *5.0.1*** odczytać port podany w polu ***HTTP Port*** (rysunek poniżej). Wartość tego portu należy wpisać w adresie strony www.



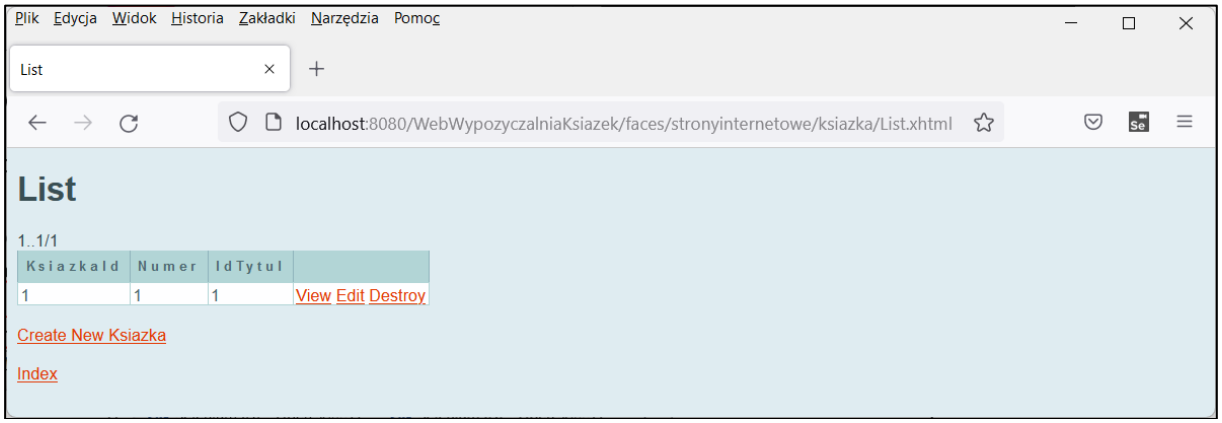
Dodatek 2

Testowanie funkcjonalne aplikacji internetowej za pomocą narzędzia *Selenium IDE.*

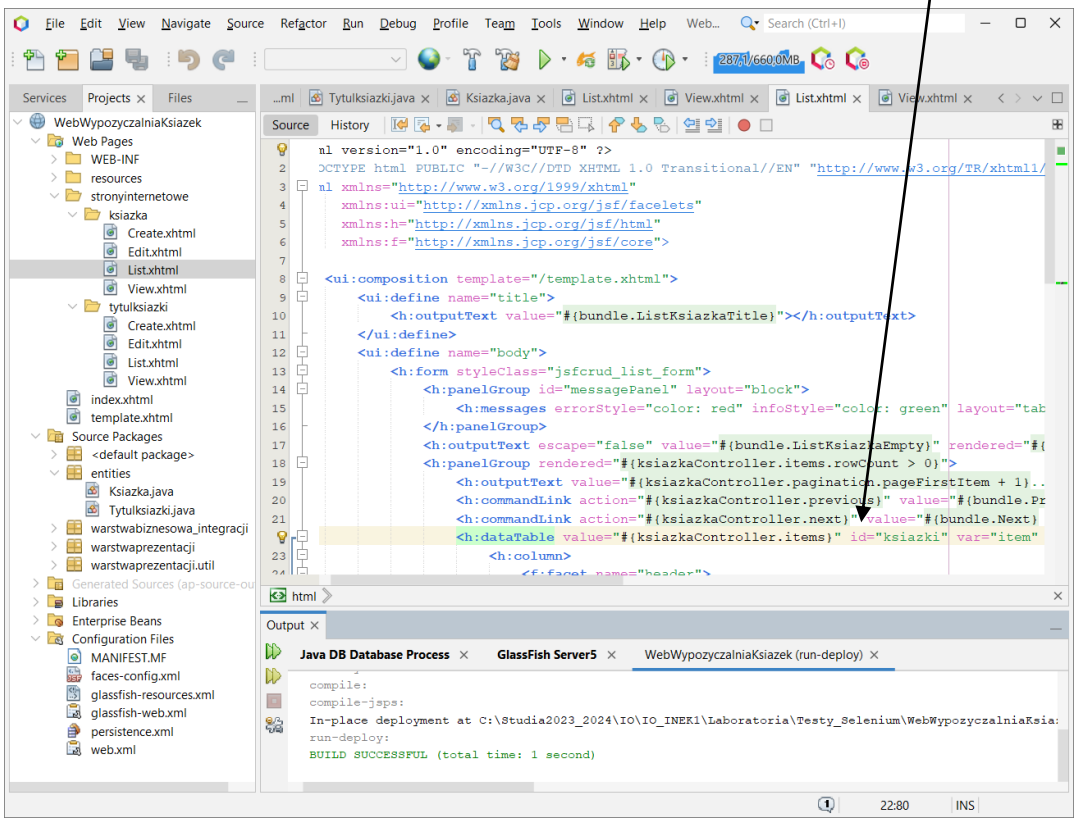
1. W celu zapoznania się ze sposobem instalacji *Selenium IDE*, z programowaniem testów oraz innymi ważnymi informacjami, należy skorzystać z dokumentacji tego środowiska na stronie: Getting Started · Selenium IDE, Selenium IDE · Open source record and playback test automation for the web. Podczas budowy testów dokumentacja jest dostępna ze strony narzędzia ***Selenium IDE***. Poniżej pokazano uruchomioną przeglądarkę ***FireFox***, w której jest już zainstalowane narzędzie ***Selenium IDE 3.17.4***. Narzędzie uruchamia się po kliknięciu na ikonę **Se**. Pojawi się strona startowa narzędzia **Selenium IDE**, gdzie należy kliknąć na pozycję **Create a new project** w celu wykonania testów funkcjonalnych aplikacji internetowej **WebWypozyczalniaKsiazek.** W celu zapoznania się ze sposobem budowy testów należy kliknąć na link the Selenium IDE project page, Getting Started · Selenium IDE.



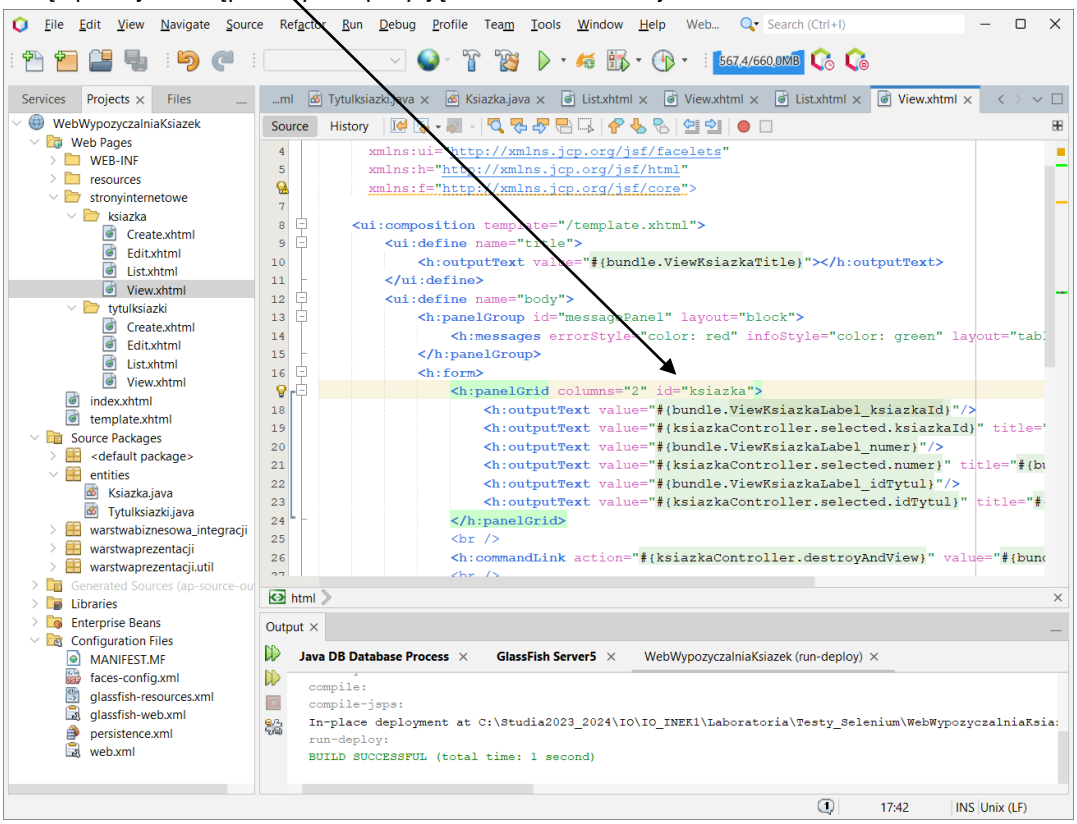
Aby testować zawartość tabeli z książkami na formularzu strony ***List.html***, pokazanego poniżej, potrzebna jest nazwa tabeli jako atrybut ***id*** znacznika **<table>** w języku ***xhtml***. Taki atrybut nie został wygenerowany podczas automatycznego tworzenia aplikacji internetowej w technologii **JSF,** pokazanego w **Dodatku 1.**

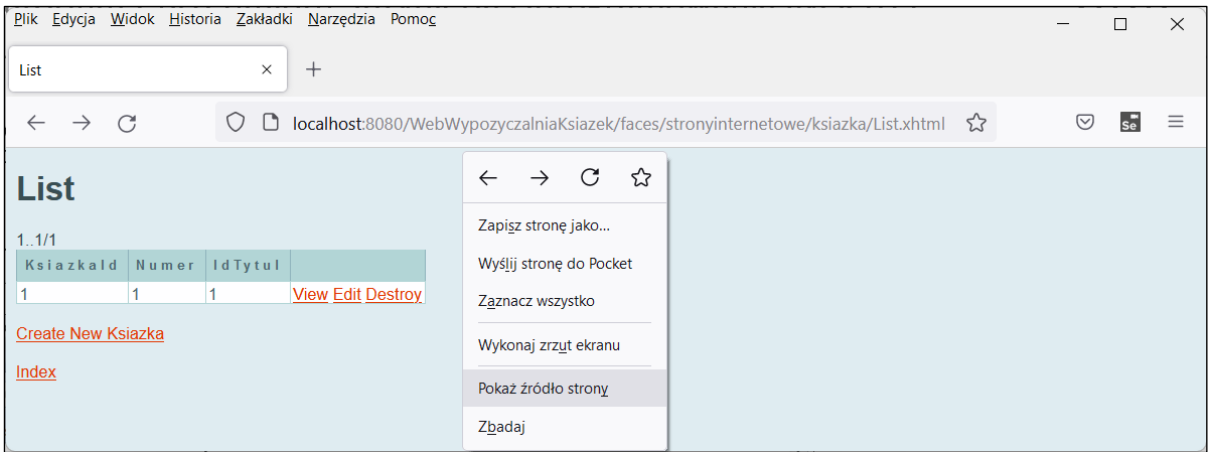


Dalej pokazano stronę **JSF** o nazwie ***List.xhtml,*** która stanowi wzorzec do wygenerowania strony ***List.html*** w fazie ***Response*** i wysłanie jej do przeglądarki (p.1 **Dodatku 1**). Po dodaniu atrybutu **id="książki"** w znaczniku JSF **<h:dataTable>,** nazwa tabeli może być prezentowana przez ten znacznik uzupełniony podczas generowania strony html.

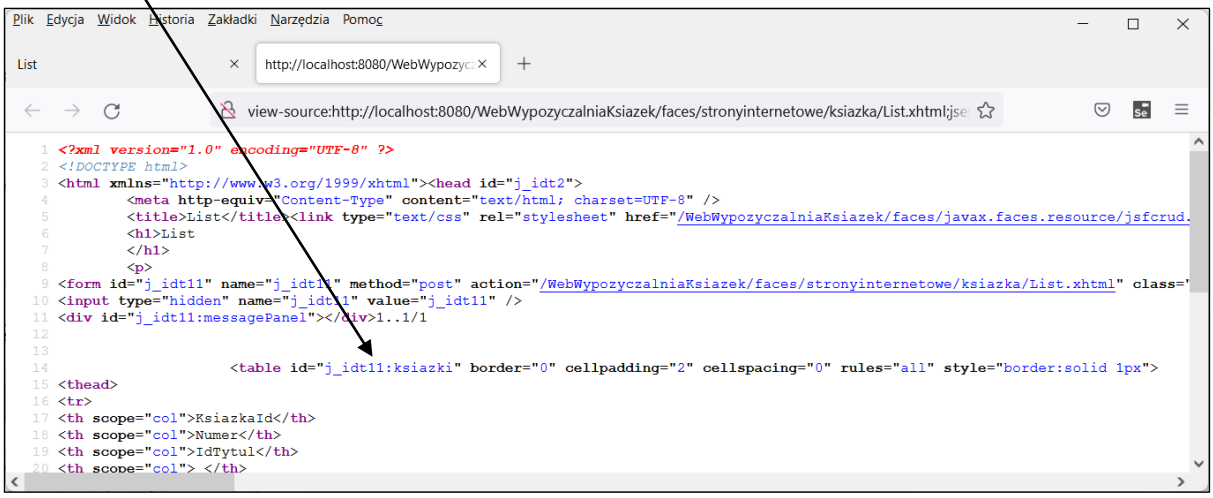


Podobnie nadano ***id="ksiazka"*** w znaczniku **<h:panelGrid>** w pliku ***View.xhtml*** w tym samym katalogu.
Aby poznać dane znaczników wygenerowanej strony ***html***, należy kliknąć prawym klawiszem myszy na daną stronę aplikacji i następnie wybrać pozycję ***Pokaż źródło strony***.

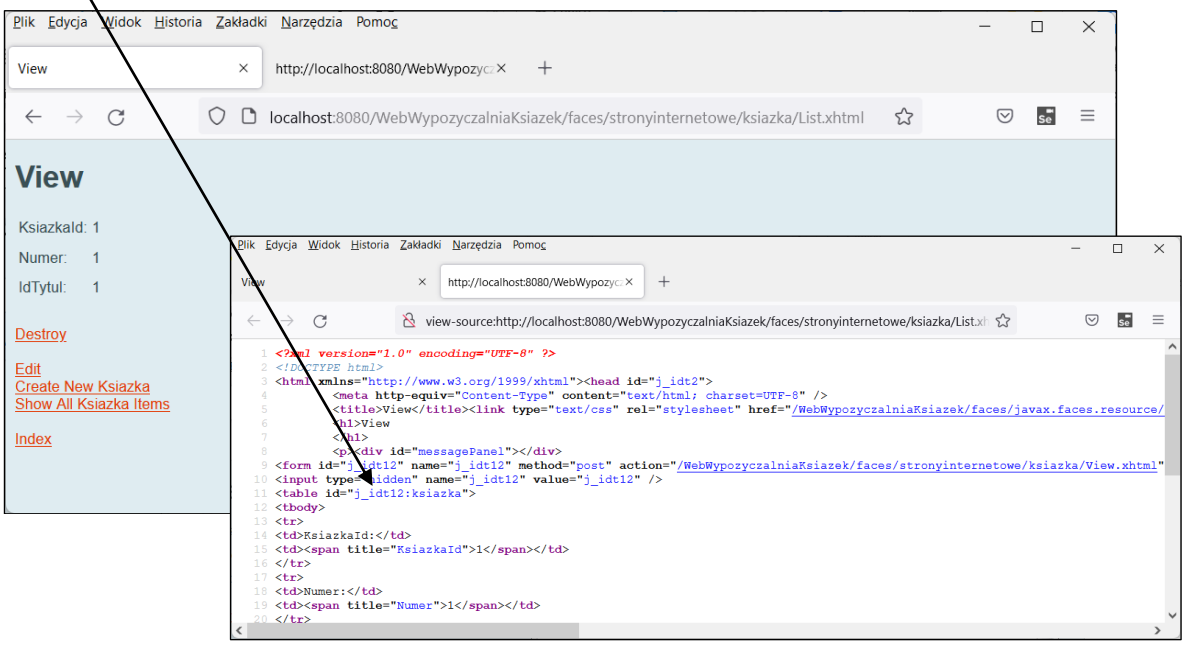




Poniżej pokazano tekst strony ***List.html*** napisanej w języku ***xhtml***. Znacznik ***<table>*** zawiera atrybut ***id="j_idt11:ksiazki",*** który służyć będzie do identyfikacji komórek tabeli podczas testowania.



Poniżej pokazano tekst strony ***View.html*** napisanej w języku ***xhtml***. Znacznik ***<table>*** zawiera atrybut ***id="j_idt12:ksiazka",*** który służyć będzie do identyfikacji komórek tabeli podczas testowania.



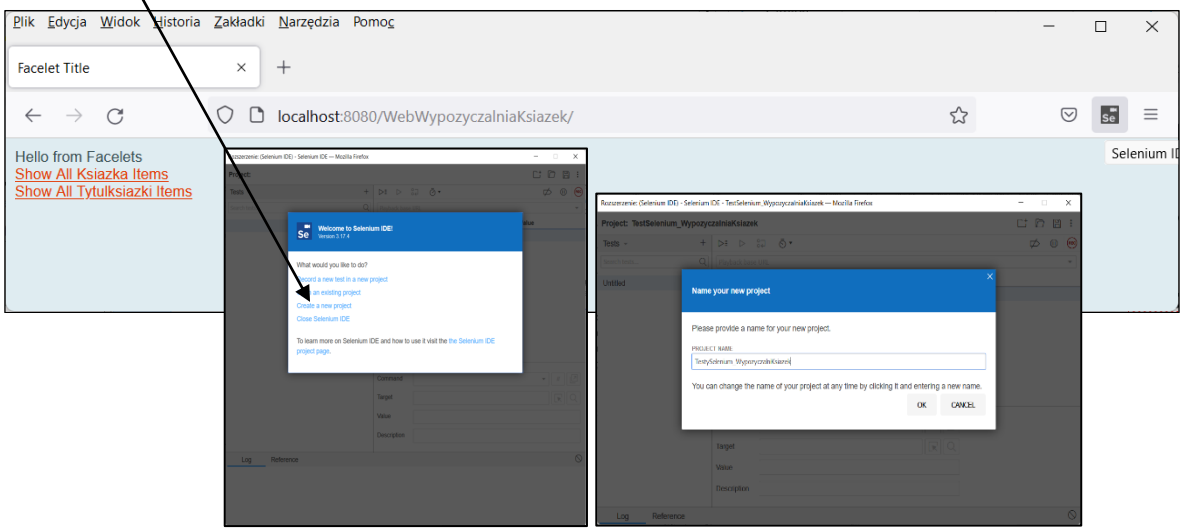
2. Testowanie wybranej funkcji aplikacji, której działanie opisano za pomocą scenariuszy przypadków użycia (instrukcja do lab1, p. 4.2, **Nazwa PU**: Dodaj_ksiazke)

Nazwa PU: Dodaj_ksiazke
Cel: Dodanie nowej książki
Warunki początkowe:
Uruchomienie programu jako aplikacji www lub aplikacji w architekturze typu "klient-serwer„
Warunki końcowe:
Wprowadzenia danych książki o unikalnym numerze egzemplarza w ramach książek o tym samym tytule
Scenariusz:
1. Należy podać atrybuty tytułu: ISBN jako obowiązkowa, dlatego tworzony jest tytuł wzorcowy, w którym istotny jest tylko ISBN do wyszukiwania rzeczywistego tytułu
2. Należy wywołać **PU Sprawdz_czy_jest_tytul**. Należy sprawdzić, czy tytuł o podanym ISBN już istnieje. Jeśli nie, należy zakończyć PU.
3. Należy utworzyć egzemplarz zawierający numer podany do wyszukiwania egzemplarza i należy przekazać go do **PU Sprawdz_czy_jest_ksiazka.** Jeśli nie istnieje egzemplarz o danym numerze, należy wstawić ten egzemplarz, w przeciwnym wypadku należy zakończyć PU.

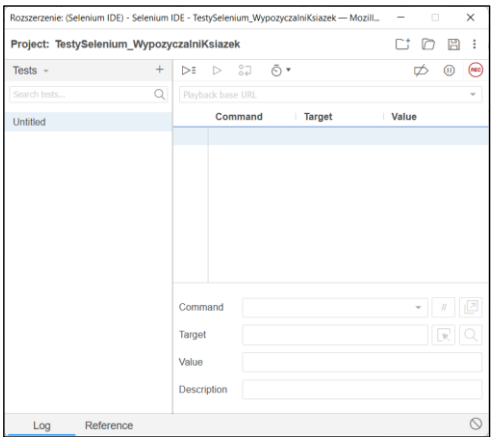
Nazwa PU: Sprawdz_czy_jest_ksiazka
Cel: Wyszukiwanie książki o podanym numerze
Warunki początkowe:
Przypadek użycia jest wywoływany z PU Dodaj_ksiazke
Warunki końcowe:
Zwraca wynik, określający, czy podany numer jest unikatowy lub podaje informację, że dany numer już istnieje
Scenariusz:
1. Szukanie książki przebiega według atrybutu: numer egzemplarza (obowiązkowo) zgodnie z danymi tytułu podanego do przypadku użycia. Przeszukiwane są egzemplarze należące do konkretnego tytułu
2. Jeśli istnieje egzemplarz o podanym numerze, zwracany jest egzemplarz z zasobów wypożyczalni, w przeciwnym wypadku zwracana jest informacja o braku egzemplarza.

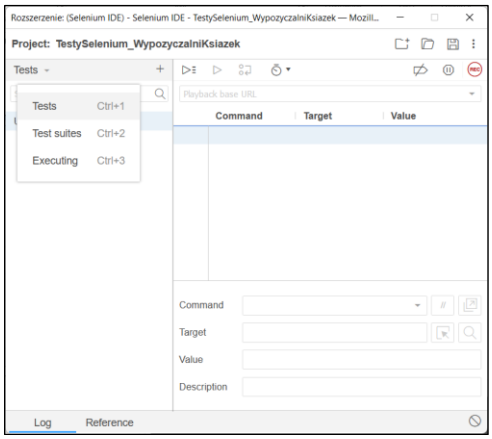
3. Wykonanie testu działania wybranej funkcji aplikacji zgodnie ze scenariuszem przypadku użycia

3.1. Pierwszy etap oparty na nagrywanie testu. Należy uruchomić wykonaną aplikację w **Dodatku 1** tzn. po uruchomieniu narzędzia **Apache NetBeans 18** należy uruchomić aplikację w następujący sposób: w zakładce ***Projects*** kliknąć prawym klawiszem myszy na nazwę projektu i wybrać z listy kolejno ***Clean and Build/Deploy.*** Następnie należy kliknąć na ikonkę **SE** umożliwiającą uruchomienie programu **Selenium IDE**. Po uruchomieniu **Selenium IDE 3.17.4** należy wykonać projekt umożliwiający przechowanie wykonywanych testów funkcjonalnych: **TestySelenium_WypozyczalniKsiazek.**

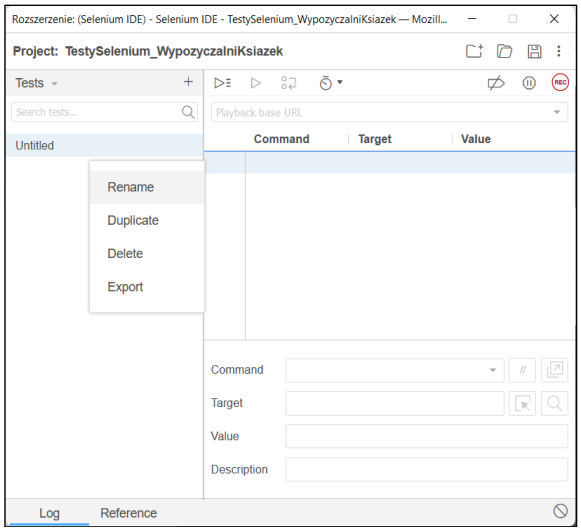


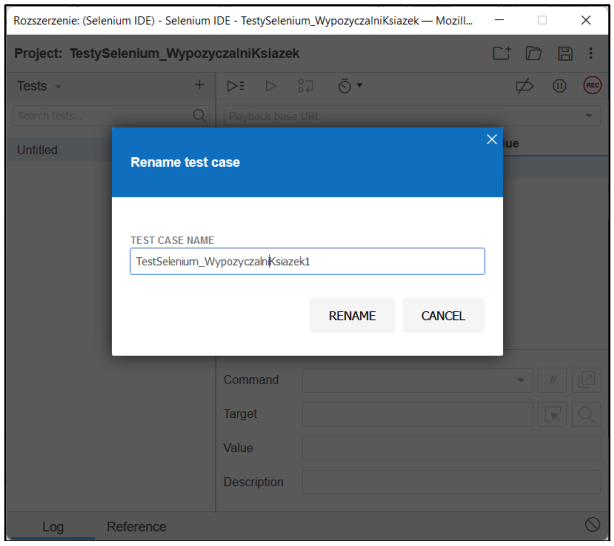
Poniżej pokazano formularz ***Selenium IDE 3.17.4***, gdzie w założonym projekcie **TestySelenium_WypozyczalniKsiazek** znajduje się plik **Untitled** reprezentujący pusty test. Należy wybrać w zakładce **Tests** pozycję **Test**.



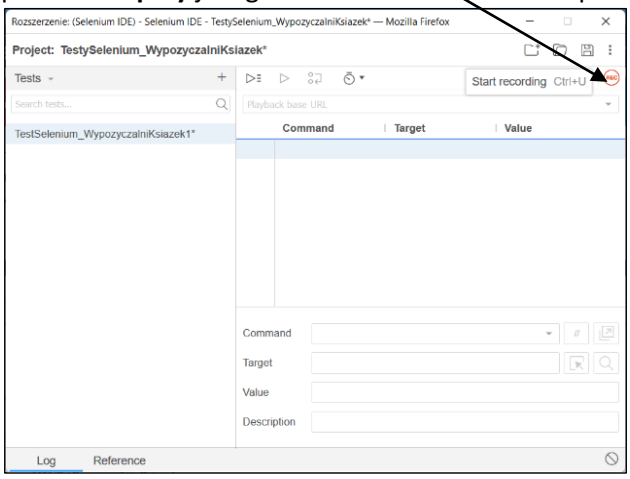


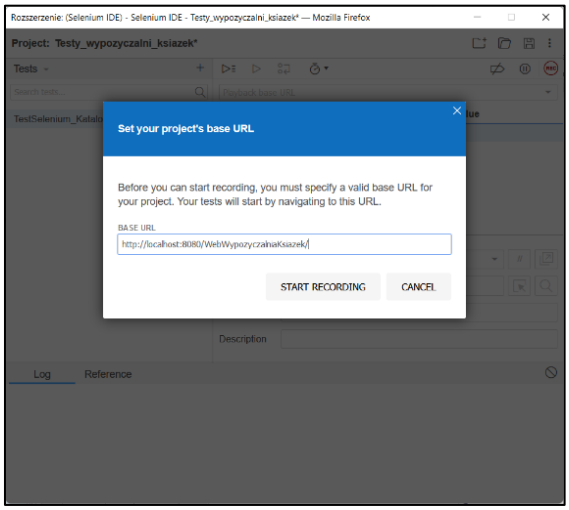
Następnie należy zmienić nazwę testu operacją **Rename** na **TestSelenium_WypozyczalniKsiazek1.**





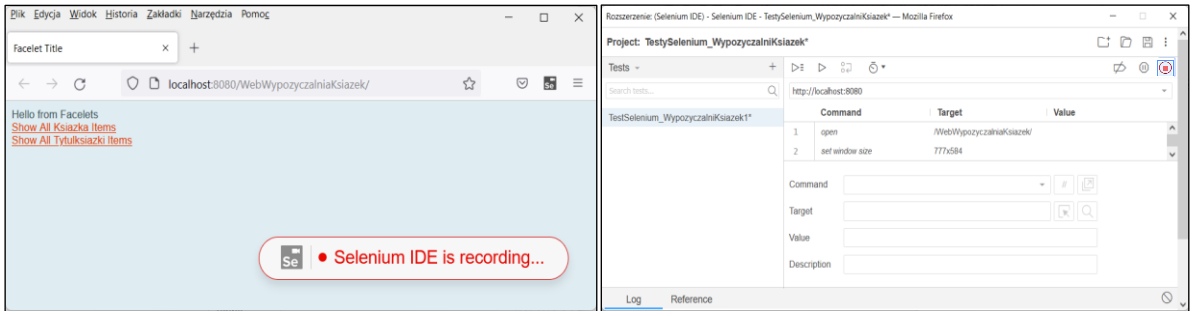
Po zmianie nazwy testu należy uruchomić proces nagrywania testu testującego dodawanie nowej książki klikając na ikonkę Rec w prawej górnej części formularza. Dzięki przekazaniu adresu strony startowej testowanej aplikacji zostanie uruchomiona aplikacja, która po wykonaniu wcześniej procesu **Deploy** jest gotowa do uruchomienia w przeglądarce.

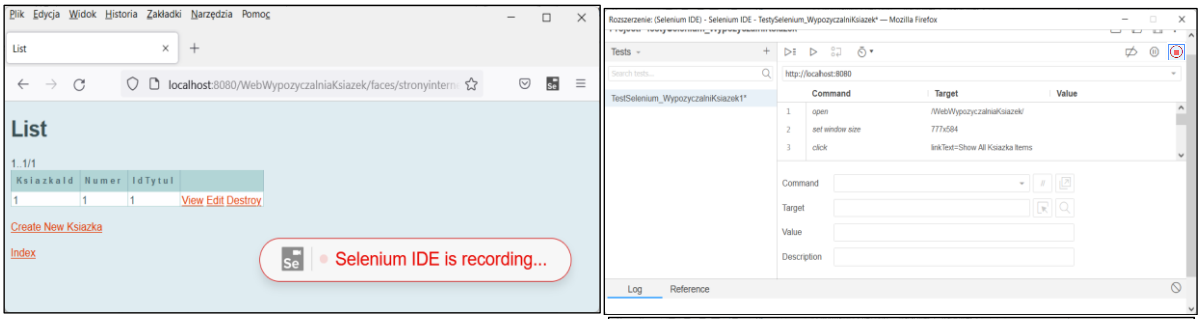


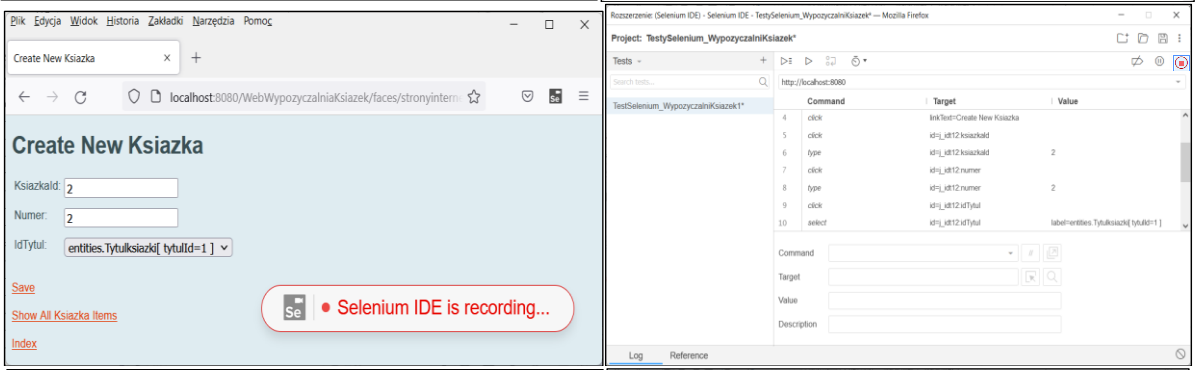


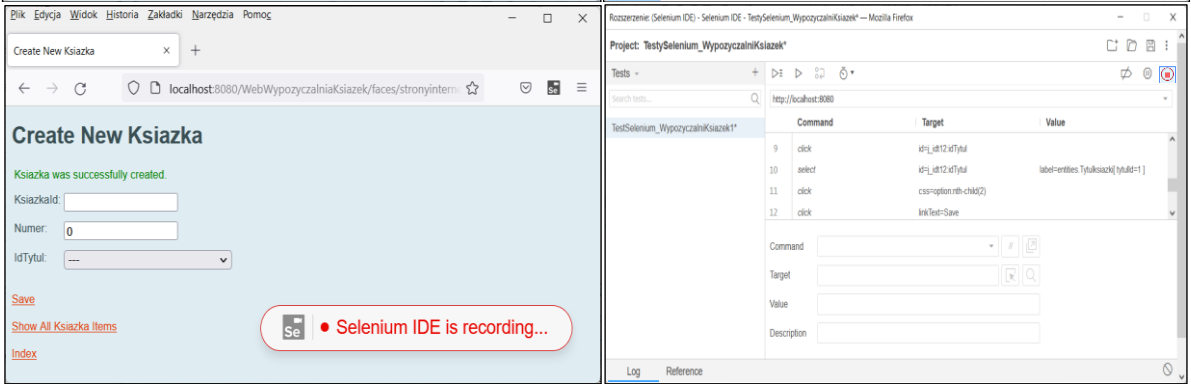
Dalej pokazano przebieg nagrywania testu.

Na kolejnych zrzutach z ekranu znajduje się prezentacja nagrania testu przebiegu wprowadzania nowej książki do tabeli ***Ksiazka***, wykonana przez użytkownika (wybór **Show AllKsiazka Items**).

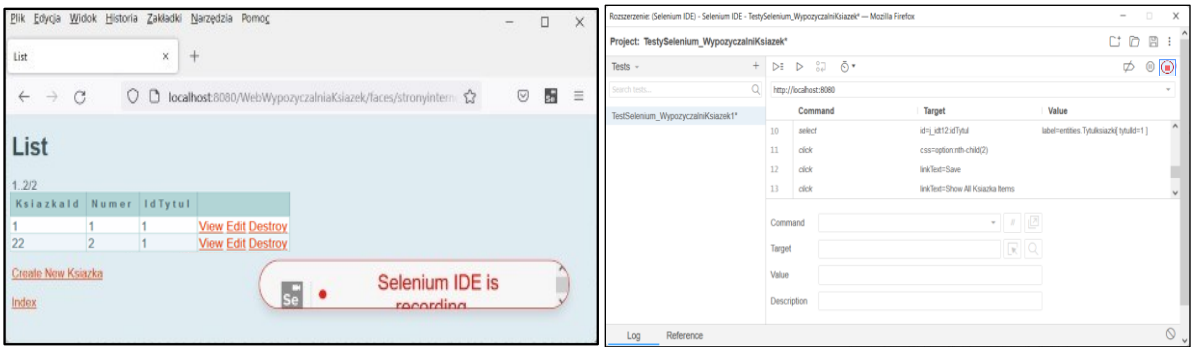


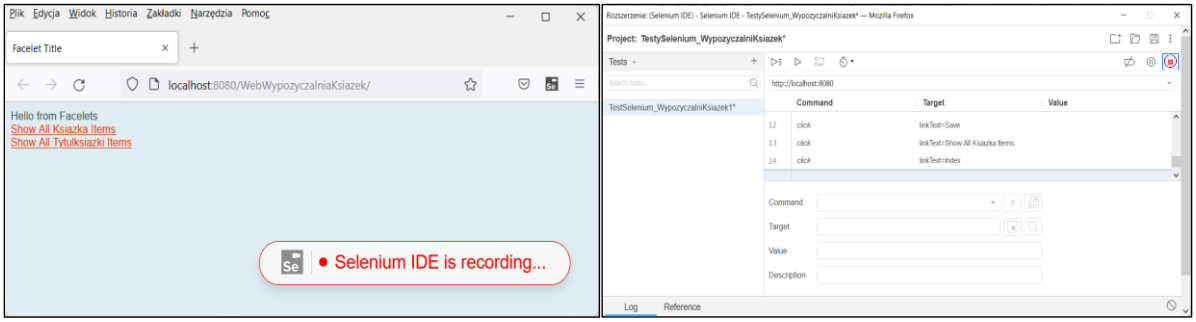




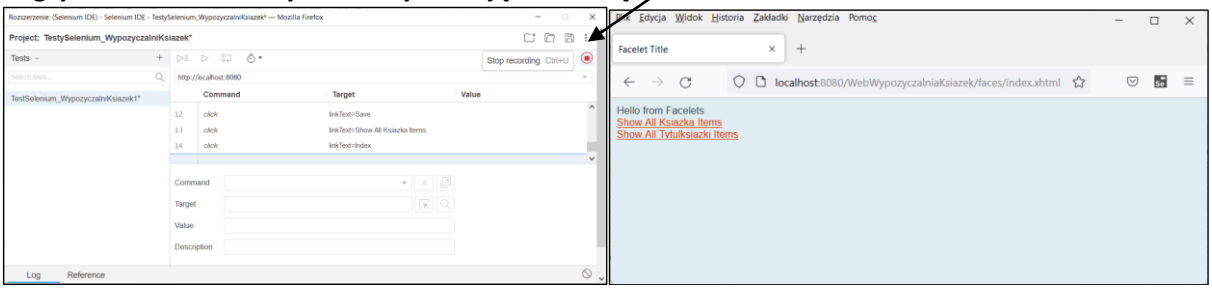


Baza danych zignorowała podaną wartość klucza głównego równą 2 i nadała mu wartość 22.

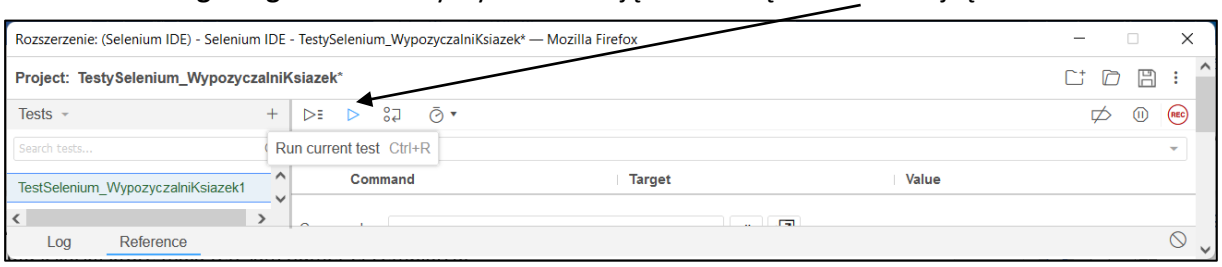




Nagrywanie testu należy zakończyć klikając na ikonę „czerwone kółko”.

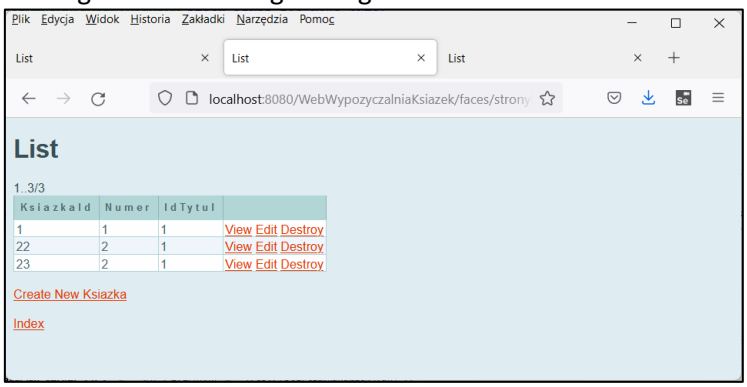


Odtworzenie nagranego testu należy wykonać klikając na ikonkę o kształce trójkąta.



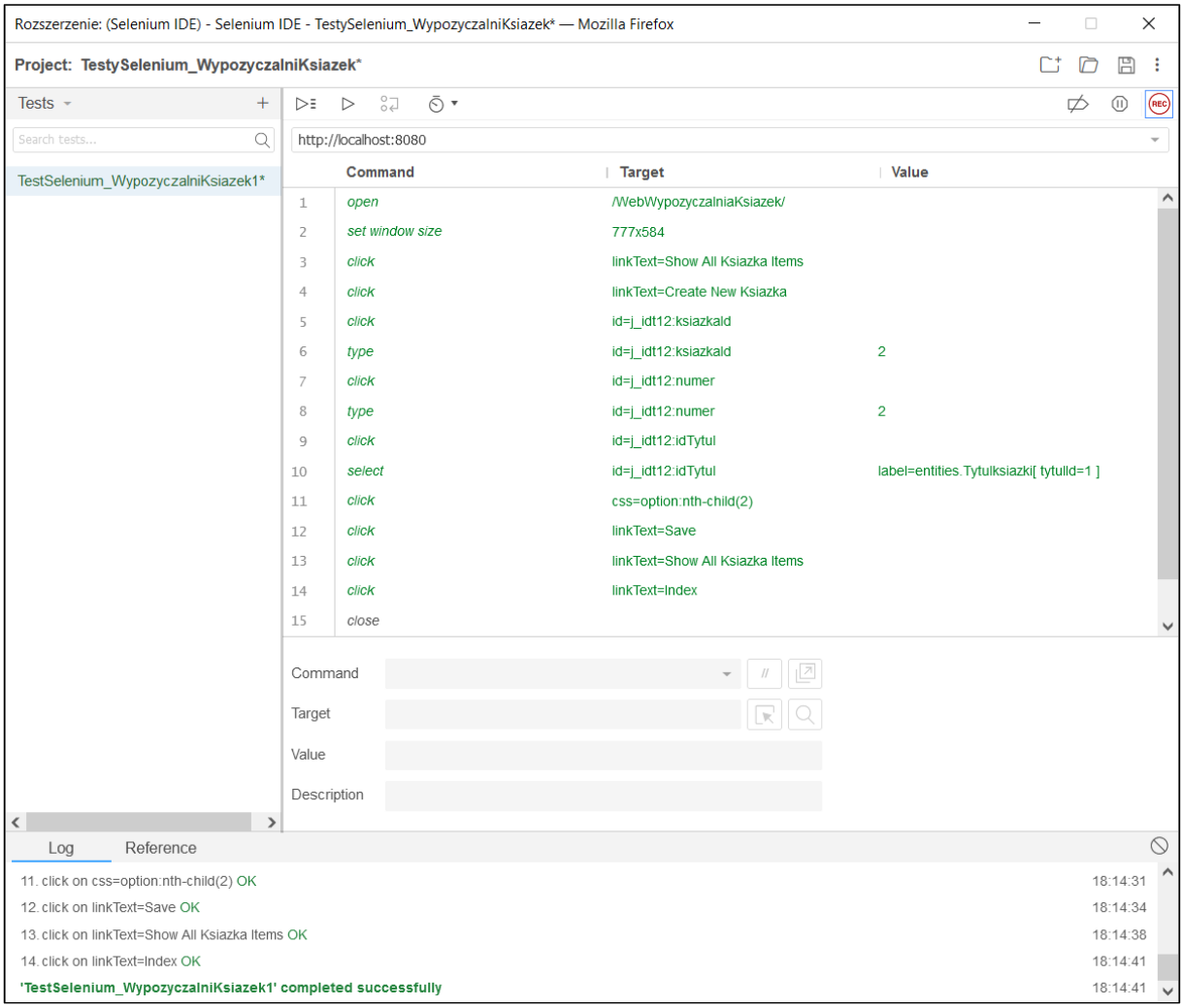
Szybkość odtwarzania testu ustawia się za pomocą suwaka, dostępnego po kliknięciu na ikonkę w kształcie tarczy zegarka.

Po odtworzeniu nagranego testu (gdzie ponownie wprowadzano te same dane do tabeli) widać (rysunek poniżej), że zapewnienie spójności danych jest ograniczone do zmiany klucza głównego w tabeli ***Ksiazka*** przez silnik bazodanowy - w teście wprowadza się klucz główny równy 2, a silnik bazodanowy nadaje kolejnej krotce, zawierającej te same dane, inną wartość klucza głównego dbając jedynie o zachowanie integralności klucza głównego.

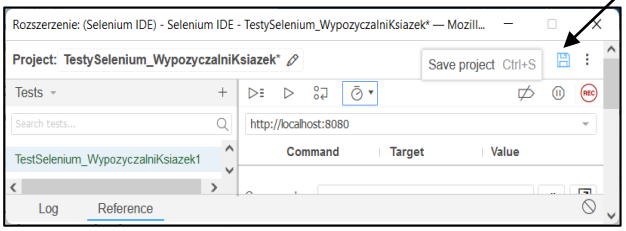


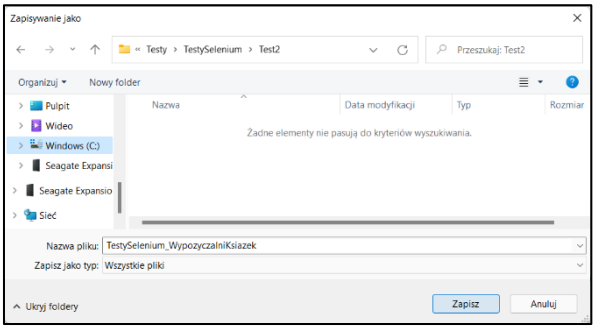
Zostało to ustalone podczas generowania klas typu ***Entity***, gdzie zostały automatycznie nadane następujące adnotacje atrybutowi Id klasom typu ***Entity***: **@Id @GeneratedValue(strategy = GenerationType.IDENTITY)**. Obecnie można zapełnić całą tabelę takimi samymi danymi, które będą się różnić się jedynie wartością klucza głównego. To w przyszłości należy zmienić, gdyż jest to niezgodne ze scenariuszem przypadku użycia dodawania nowej książki (p.2 **Dodatek 2**), który powinien realizować testowany program tzn. każdy egzemplarz książki powinien mieć unikatowy numer, a w tabeli znajdują się dwa egzemplarze należące do tego samego tytułu książki, które mają ten sam numer egzemplarza.

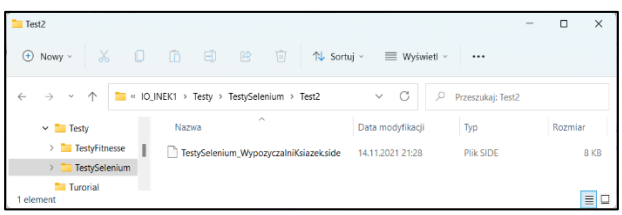
Poniżej przedstawiono kod nagranego testu w formularzu testu w narzędziu **Selenium IDE 3.17.4.**



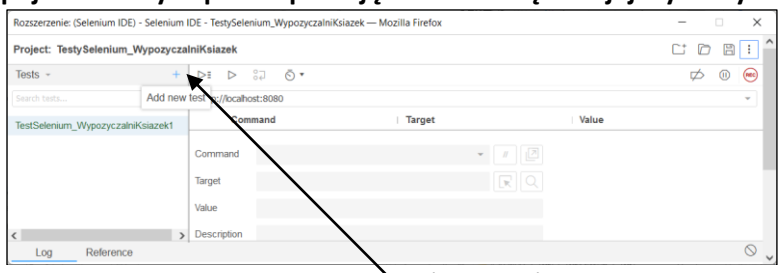
Po zakończeniu 1-go testu zapisano projekt **TestySelenium_WypozyczalniKsiazek.side** z nagranym testem.



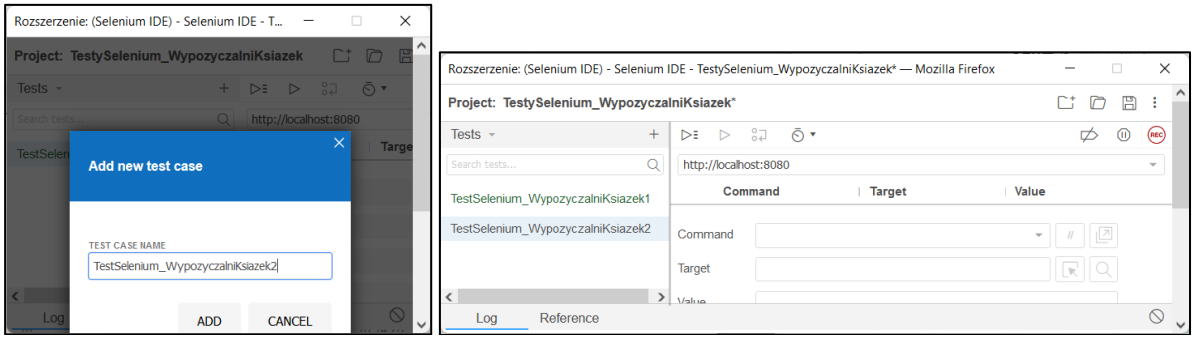




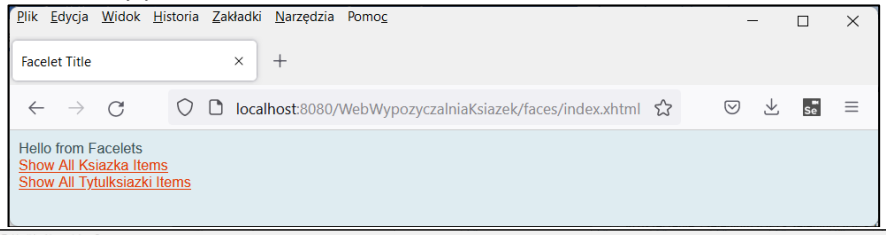
3.2. Drugi etap: tworzenie ręczne testu funkcjonalnego do uproszczonego testowania:
- elementów typu UI wybranej strony aplikacji internetowej oraz
- zachowania spójności danych przez aplikację internetową oraz jej wybranych funkcjonalności.



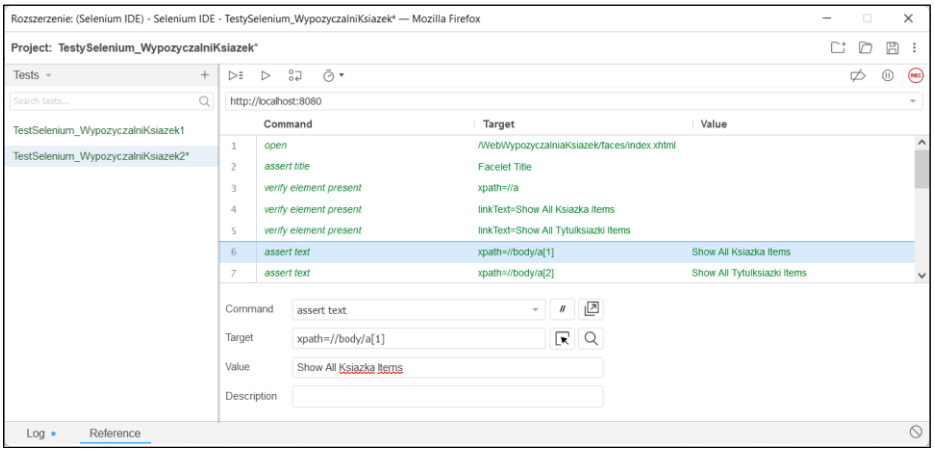
Dodano nowy test, klikając na znak + (powyżej). W przykładzie jest to plik ***TestSelenium_WypozyczalniKsiazek2.***



Następnie ręcznie wpisano polecenia **Selenium IDE** w celu sprawdzenia budowy wybranej głównej strony internetowej aplikacji – poniżej widok testowanej strony i jej kod interpretowany przez przeglądarkę i testowany przez test **Selenium IDE**.



Poniżej zaprezentowano pierwszy fragment testu.



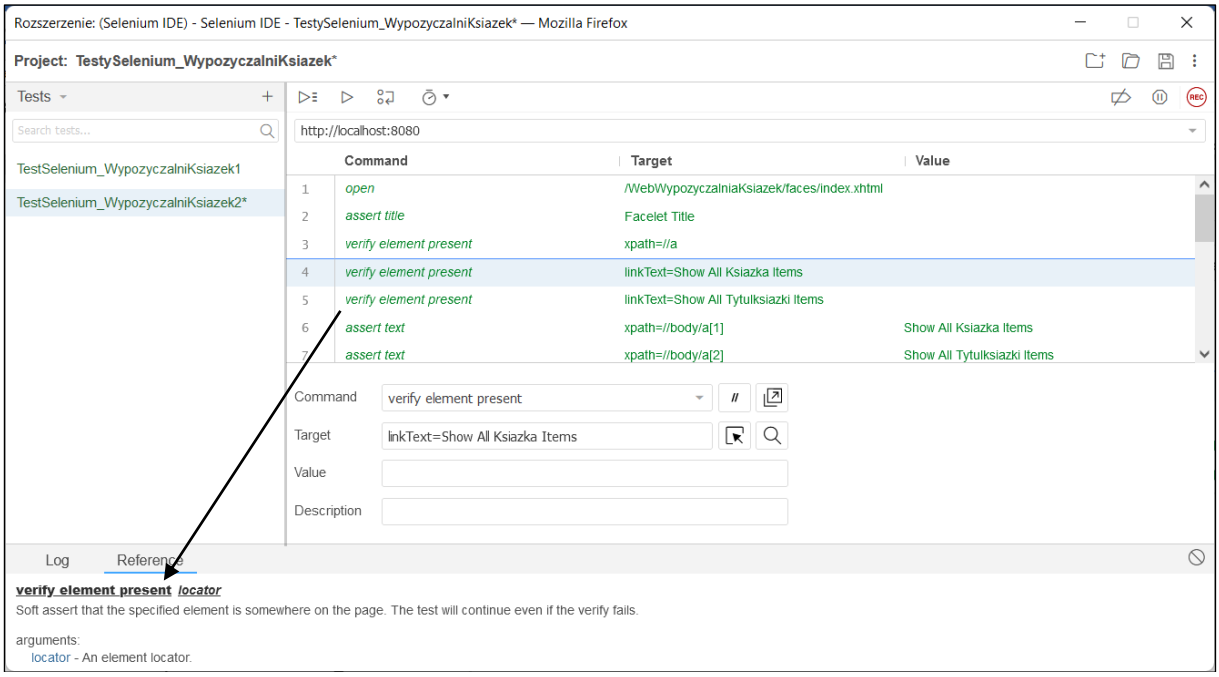
Formularz zawiera pola ***Command, Target***, ***Value,*** które umożliwiają wprowadzanie poleceń, argumentów polecenia oraz wartości argumentów. Pole ***Description*** opisuje znaczenie polecenia.

W **Tabeli 1** przedstawiono testowanie elementów ***UI*** strony internetowej, w której pokazano i skomentowano poszczególne polecenia ***Selenium IDE***.

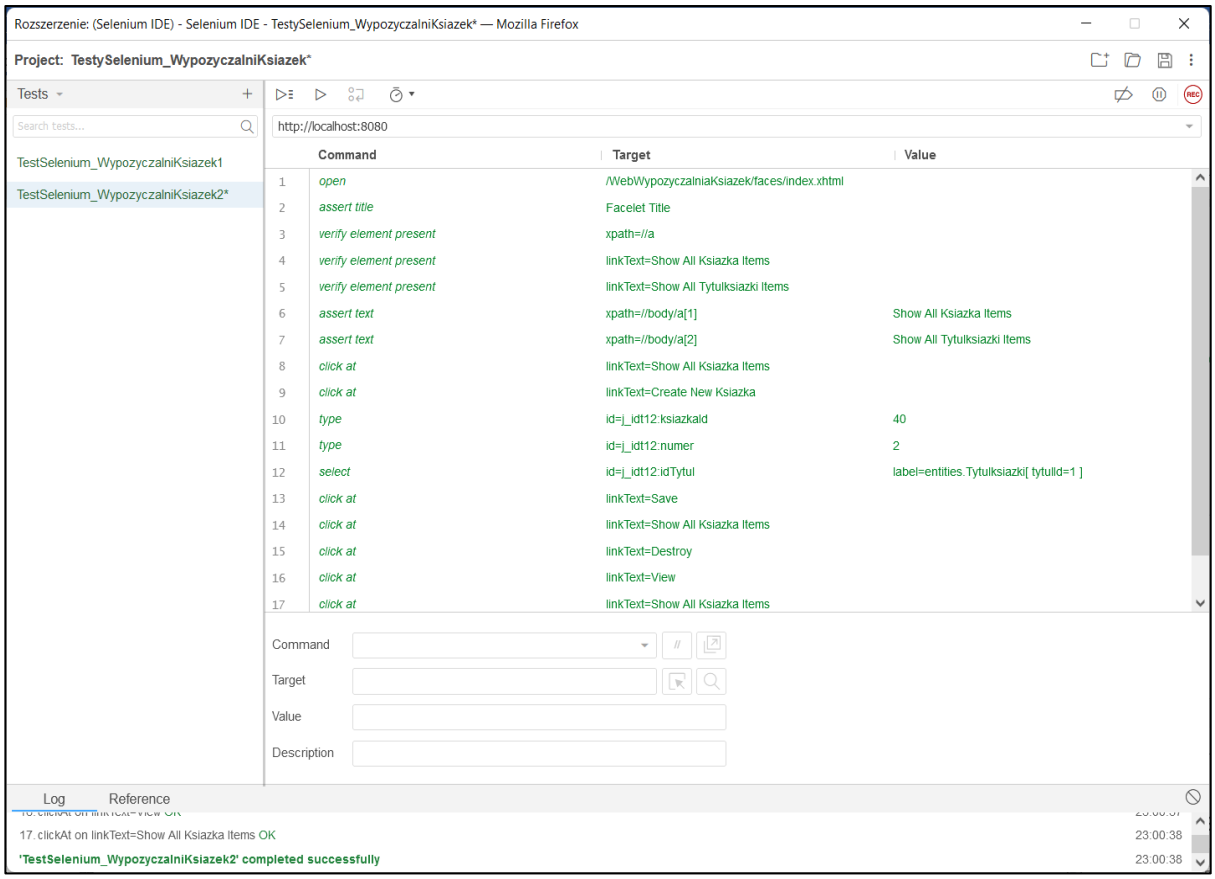
Tabela 1

Command	Target	Value	Opis – na podstawie zakładki **Reference**
open	/WebWypozyczalniaKsiazek/faces /index.xhtml		otwiera aplikację internetową
assert title	Facelet Title		Sprawdzenie tytułu testowanej strony internetowej
verify element present	xpath=//a		Sprawdzenie, czy struktura strony html posiada znacznik ***a***
verify element present	linkText=Show All Ksiazka Items		Weryfikacja tekstu wyświetlanego na stronie
verify element present	linkText=Show All Tytulksiazki Items		Weryfikacja tekstu wyświetlanego na stronie
assert text	xpath=//body/a[1]	Show All Ksiazka Items	Sprawdzenie, czy pierwszy ze znaczników ***a*** zawiera tekst umieszczany na linku: **Show All Ksiazka Items**
assert text	xpath=//body/a[2]	Show All Tytulksiazki Items	Sprawdzenie, czy drugi ze znaczników ***a*** zawiera tekst umieszczany na linku: **Show All Tytulksiazki Items**

W dolnej części formularza znajduje się w zakładce ***Reference*** informacja o znaczeniu i składni każdego z zaznaczonych poleceń. Polecenia można wybierać z listy w polu ***Command.***



Poniżej przedstawiono cały test wykonany ręcznie oraz opis znaczenia kolejnych poleceń wykonywanych przez test



Kolejna część testu (linia 8) służy do prezentacji wartości danych książek pobranych tabeli z książkami ze strony ***List.html*** zapisanych podczas: tworzenia aplikacji (Dodatek 1, p. 3.2), nagrywania i odtwarzania nagranego testu w (Dodatek 2 - bieżący, p.3.1).

Tabela 2

Command	Target	Value
click at	**linkText=Show All Ksiazka Items**	

Kolejna część testu służy do testowania wprowadzenia nowej krotki.

Tabela 3

Command	Target	Value
click at	**linkText=Create New Ksiazka**	
type	id=j_idt12:ksiazkaId	22
type	id=j_idt12:numer	2
select	id=j_idt12:idTytul	label=entities.Tytulksiazki[tytulId=1]
click at	**linkText=Save**	

Kolejna część testu dotyczy testowania zachowania spójności danych przez aplikację internetową.

Tabela 4

Command	Target	Value
click at	**linkText=Show All Ksiazka Items**	
click at	**linkText=Destroy**	
click at	**linkText=View**	
click at	**linkText=Show All Ksiazka Items**	