

POLITECHNIKA KRAKOWSKA
WYDZIAŁ INŻYNIERII ELEKTRYCZNEJ I KOMPUTEROWEJ
Kierunek: Informatyka

Dokumentacja projektowa

Maria Guz, Karol Pątko

GaART

**SYSTEM WSPOMAGAJĄCY PRACĘ
GALERII SZTUKI**

projekt utworzony pod kierunkiem
mgr inż. Anny Sucheni

w ramach przedmiotu
Systemy odporne na błędy

Kraków, 10.06.2020

Tematyka	3
Opis problemu	6
Metoda rozwiązania	7
Metoda ostrzegania	7
Metoda koniecznego kroku	8
Opis działania programu	9
Metoda ostrzegania	9
Metoda koniecznego kroku	9
Implementacja	10
Metoda ostrzegania	10
Dodanie wydarzenia	10
Metoda koniecznego kroku	14
Zakup biletów	14
Rezerwacja biletów	17
Dodawanie biletów do koszyka	19
Przykładowe działanie programu	20
Dodawanie wydarzenia	20
Kupno biletu	22
Rezerwacja biletu	23
Dodawanie biletów do koszyka	23
Testy	25
Otrzymane wyniki i wnioski	30
Wykorzystane materiały, narzędzia, literatura	30
Załącznik: Uruchomienie projektu	31

Niniejszy tekst stanowi dokumentację projektu systemu wspomagającego pracę instytucji kultury jaką jest galeria sztuki współczesnej. Zawiera on m.in. tematykę jakiej dotyczy system, opis problemu, metody rozwiązania problemów, działanie systemu, informacje o implementacji, testy, wyniki oraz wnioski. Dodatkowo załączona została instrukcja uruchamiania systemu.

Tematyka

W dobie digitalizacji wszelkich zasobów różnorodnych instytucji nie sposób pominąć instytucje kultury jakimi są muzea czy galerie sztuki. Możliwość obcowania ze sztuką nie będąc fizycznie w sali wystawowej jest jedną z ogromu możliwości jakie zawdzięczamy informatyce. Dzięki elektronicznej bazie danych dotyczącej eksponatów muzealnych czy dzieł znajdujących się w galerii sztuki osoby, które nie mają możliwości doświadczenia ich na żywo, mogą zrobić to za pomocą systemu informatycznego przeznaczonego do tego celu. Za jego pośrednictwem można nie tylko przeglądać obiekty ale również wyszukać konkretny eksponat czy posortować wyniki ze względu na styl, autora lub cenę. Dodatkowo, w przypadku instytucji takiej jak galeria sztuki, dzięki wprowadzeniu systemu wspomagającego pracę, klienci mają możliwość zapoznania się ze szczegółami dotyczącymi obiektu oraz zakupu dzieła. Często oprócz wyżej wymienionych funkcjonalności za pomocą m.in. strony internetowej klienci danej instytucji mogą zarezerwować lub kupić bilet na organizowane w niej różnego rodzaju wydarzenia. Od strony pracowników instytucji system wspomagający pracę może być narzędziem, które udoskonali przepływ informacji oraz pomoże je uporządkować. Oczywiście kontakt instytucji z potencjalnymi klientami jest bardzo istotny dla poprawnego funkcjonowania firmy, dlatego systemy informatyczne często są wyposażone w opcję szybkiego kontaktu z daną instytucją. Systemy te są również proste i intuicyjne, tak by osoba w każdym wieku i o różnym doświadczeniu z technologicznymi rozwiązaniami wprowadzanymi na rynek mogła z nich swobodnie korzystać.

Celem projektu jest stworzenie systemu służącego do zarządzania zasobami galerii sztuki. Dodatkowo aplikacja ma umożliwiać przeglądanie oraz kupno dzieł. Możliwa ponadto będzie rezerwacja oraz sprzedaż biletów na wydarzenia organizowane przez instytucję. Pracownicy galerii za pośrednictwem aplikacji będą mogli zarządzać danymi dzieł, artystów i

wydarzeń. System musi być stabilny, prosty w obsłudze zarówno dla pracowników galerii jak i klientów. Interfejs powinien być intuicyjny i funkcjonalny – powinien implementować wszystkie przewidywane funkcje niezbędne podczas zarządzania instytucją kultury. Co więcej sama aplikacja nie może zajmować dużo pamięci. Do utworzenia projektu niezbędny jest zespół projektowy oraz przestrzeń biurowa wraz z serwerami co umożliwi pracę nad tworzonym systemem. Aplikacja ma być globalna.

Głównym zadaniem systemu jest zautomatyzowanie czynności związanych z zarządzaniem galerią sztuki oraz ułatwienie klientom dostępu do oferty galerii poprzez digitalizację zasobów. System będzie także ułatwieniem w komunikacji między klientem instytucji a jej pracownikami. Ma on za zadanie przetwarzać, analizować i aktualizować dane dotyczące zasobów galerii oraz wydarzeń przez nią organizowanych. Wymagania dotyczące systemu zostały opracowane wspólnie z działem IT galerii, którego zadaniem będzie kontrolowanie działania systemu. System będzie się składał z aplikacji klienta, serwera oraz bazy danych. System będzie używany przez pracowników instytucji oraz potencjalnych klientów galerii. Pracownicy będą mieli możliwość dodawania lub modyfikowania szczegółów dotyczących zasobów galerii i wydarzeń. Klienci natomiast za pośrednictwem systemu będą mogli zakupić bilety na wydarzenia organizowane przez galerię. Będą również mieli możliwość przeglądania zasobów galerii oraz zakupu dzieła. Administrator systemu ma możliwość tworzenia i modyfikacji kont użytkowników. Ma dostęp do danych użytkowników. Jest on odpowiedzialny za zarządzanie kontami.

Za pośrednictwem strony można także zgłosić się jako artysta. Powoduje to nadesłanie zgłoszenia do pracownika, który po zweryfikowaniu może je zaakceptować. W zakres projektu oprócz wyżej opisanego systemu należy wliczyć także pełną dokumentację oraz odpowiednie przygotowanie przyszłych użytkowników do sprawnej obsługi systemu. Projekt wymaga również odpowiedniego zaplecza technicznego oraz wyszkolonego personelu. W ramach projektu wykonawca gwarantuje 3 miesięczny okres próbny. Jest to czas na zgłaszanie uwag oraz pomoc techniczną od strony wykonawcy. W zakres projektu wchodzi: komunikacja z serwerem, komunikacja między użytkownikami, komunikacja z serwerem baz danych, zapewnienie odpowiedniego zaplecza technicznego, szkolenia pracowników, okres próbny, serwisowanie przez określony czas, bezpieczne wdrożenie systemu. Proces wdrażania systemu informatycznego jest jedną z najbardziej wrażliwych na

błędy faz. Podczas wdrożenia system jest przekazywany klientowi który go zamówił, dlatego konieczne jest opracowanie odpowiedniej strategii wdrożeniowej. Jeszcze przed wdrożeniem systemu warto wykonać testy produktu. Mają one na celu wykazać wszystkie nieścisłości systemu, błędy, usterki itp. Do procesu wdrażania najlepiej poprosić o pomoc wykwalifikowane do tego osoby np. wykształconych informatyków, którzy mogą przetestować system od każdej strony. W tym przypadku również została powołana specjalna grupa osób zajmująca się tego rodzaju testami.

W procesie wdrażania należy pamiętać jak ważne jest dobre przygotowanie użytkowników do korzystania z systemu. Z tego powodu warto pomyśleć o szkoleniach lub spotkaniach, na których wyjaśnione zostaną wszystkie funkcjonalności systemu oraz zaprezentowane zostanie jego użytkowanie. Warto także dać możliwość przetestowania systemu przez każdego użytkownika, a w razie problemów lub pytań być gotowym, by niezwłocznie pomóc. Przed wdrożeniem oprócz testów w zamkniętym środowisku warto również sprawdzić zaplecze sprzętowe klienta oraz w razie potrzeby zakupić dodatkowo elementy, które są niezbędne do prawidłowego funkcjonowania systemu informatycznego. Czas wykonania projektu szacuje się na 2 miesiące od rozpoczęcia pracy. Czas wdrożenia to tydzień. Okres próbny natomiast wynosi 3 miesiące. Po 6 latach od wdrożenia wygasa wsparcie techniczne wykonawcy. Po dokonaniu analizy przedwdrożeniowej i spotkaniu z osobami odpowiedzialnymi za wdrożenie systemu poznaliśmy oczekiwania klienta. Poprzez dialog ustaliliśmy ogólny zakres systemu. Poznawszy oczekiwania mogliśmy zacząć pracę nad wyborem technologii, architekturą systemu oraz specyfikacją wymagań. Opracowany system powinien usprawnić działanie galerii *GalART* oraz rozszerzyć działalność instytucji poza fizyczne ramy. Pozwoli to na zyskanie większej ilości potencjalnych klientów, co wpłynie na zyski i funkcjonowanie galerii. Oczekiwane jest co najmniej dwukrotny wzrost zainteresowania ofertą galerii. Wykonawca zobowiązuje się do utworzenia odpowiedniej, zoptymalizowanej bazy danych, stworzenia aplikacji pozwalającej na użytkowanie systemu, zapewnienie, serwisowanie i konserwację zaplecza technicznego, wdrożenie systemu łącznie ze szkoleniami dla użytkowników, pomoc w razie problemów, awarii.

Opis problemu

System będzie używany przez pracowników instytucji oraz potencjalnych klientów galerii. Pracownicy będą mieli możliwość dodawania lub modyfikowania szczegółów dotyczących zasobów galerii i wydarzeń. Będą mieli także dostęp do biletów użytkowników. Klienci natomiast za pośrednictwem systemu będą mogli zakupić bilety na wydarzenia organizowane przez galerię. Będą również mieli możliwość przeglądania zasobów galerii oraz zakupu dzieła, wyszukania dzieła lub wydarzenia i podglądu swoich biletów lub zakupionych eksponatów. Administrator systemu ma możliwość tworzenia i modyfikacji kont użytkowników. Ma dostęp do danych użytkowników. System składał się będzie z części klienckiej, serwera oraz bazy danych. System powinien rozwiązać problem małej rozpoznawalności galerii na rynku ogólnopolskim oraz, w przyszłości, światowym. Jest to jeden z najważniejszych problemów, z którym boryka się aktualnie instytucja. System pozwoli przekroczyć fizyczne granice dostępności zasobów galerii zarówno w kontekście przestrzennym ale również w kontekście ułatwienia dostępu do eksponatów osobom nie będącym w stanie fizycznie pojawić się w sali wystawowej. Osiągnięcie produktu gotowego w stu procentach wymaga poprawnego działania wszystkich używanych zasobów udostępnionych przez inne firmy lub osoby.

Jako założenia projektowe przyjmujemy:

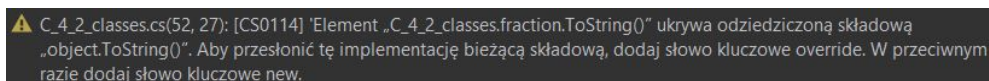
- Określenie celów projektu - Koszty, czas, pracochłonność i wszelkie ograniczenia odnośnie zasobów powinny być określone i wzięte pod uwagę.
- Minimalizacja ryzyka - korzyści, które przyniesie wynik projektu wraz z prawdopodobnymi ryzykami powinny być jak najwcześniej rozpatrzone. Na przykład należy rozpatrzyć wszelkie względy bezpieczeństwa.
- Planowanie prac potrzebnych do przygotowania planu projektu i elementów sterowania

Metoda rozwiązania

Metoda ostrzegania

Głównym miejscem gdzie taka metoda może zostać wykorzystana jest obszar, w którym nie ma możliwości stwierdzenia pewności wystąpienia błędu i przez to usunięcia tworzonego elementu. Założeniem metody jest, że osoba otrzymująca błąd poświęci swój czas, aby się zapoznać z otrzymanymi informacjami na temat błędu oraz w razie potrzeby wykona potrzebne kroki w celu jego neutralizacji.

Najczęściej błędy są sygnalizowane informacją tekstową, która znacznie wyróżnia się na tle otoczenia. Nie rzadko stosuje się także sygnały dźwiękowe z wysokim lub stale modulowanym dźwiękiem. Niekiedy także dodawane są piktogramy, które powszechnie są stosowane do ostrzegania. W świecie programowania systemem ostrzegania jest np. *console log* z informacjami o postępie kompilacji, w tym przypadku konkretniej z informacjami o możliwych błędach.

A screenshot of a Java compiler warning message. It starts with a yellow triangle icon. The text reads: "C_4_2_classes.cs(52, 27): [CS0114] 'Element „C_4_2_classes.fraction.ToString()” ukrywa odziedziczoną składową „object.ToString()”. Aby przesłonić tę implementację bieżącą składową, dodaj słowo kluczowe override. W przeciwnym razie dodaj słowo kluczowe new."/>

⚠ C_4_2_classes.cs(52, 27): [CS0114] 'Element „C_4_2_classes.fraction.ToString()” ukrywa odziedziczoną składową „object.ToString()”. Aby przesłonić tę implementację bieżącą składową, dodaj słowo kluczowe `override`. W przeciwnym razie dodaj słowo kluczowe `new`.

Do takich systemów można także zaliczyć miejskie systemy ostrzegania o zagrożeniach lub samochodowe systemy ostrzegania podczas parkowania. Ostatecznie jednak wszystkie dalsze kroki są zależne od człowieka obsługującego dany element i to on może zdecydować o podejmowanych czynnościach w stosunku do otrzymanych informacji.

Metoda koniecznego kroku

Podstawowym założeniem metody jest, aby elementy wymagające wcześniejszego kroku lub dotarcia do nich o odpowiednim czasie udostępniały taką możliwość. Na przykład jeżeli potrzeba wykonać daną czynność w określonym czasie, to czas taki powinien być przyszły w stosunku do momentu jego ustalenia, gdyż wykonanie czegoś w przeszłości byłoby niemożliwe. Jeśli natomiast jakaś czynność wymaga wcześniejszego kroku należy dopilnować aby wcześniejszy krok był dostępny w zakresie globalnym bądź po przejściu innego kroku.

Najpopularniejszym rozwiązaniem problemu w którym należy zastosować konieczny krok jest maszyna stanów, która umożliwia przechodzenie pomiędzy poszczególnymi stanami w określonej kolejności. Taką maszynę często wykorzystuje się podczas opisu rozkazów procesora w języku VHDL. Nieświadomie często takie rozwiązanie jest wykorzystywane we wszelakich przedsiębiorstwach.

Opis działania programu

Metoda ostrzegania

Metoda została zaimplementowana w module dodawania wydarzenia. W momencie dodania wydarzenia system sprawdza czy występują inne wydarzenia tego samego dnia. Jeżeli takie zostały odnalezione to są one wyświetlane wraz z ostrzeżeniem. W tym momencie użytkownik sam może zdecydować czy wydarzenia będą ze sobą kolidować i w razie konieczności usunąć wybrane wydarzenie.

W przypadku jeśli wydarzenia nie zostaną odnalezione w tym samym dniu pojawia się jedynie informacja o pomyślnym dodaniu wydarzenia.

Metoda koniecznego kroku

Metoda koniecznego korku została zaimplementowana w miejscach zakupu biletu, rezerwacji biletu oraz dodania biletu do koszyka. Całość opiera się o ideę maszyny stanów co pozwala na łatwe i szybkie sprawdzanie czy poprzedni wymagany krok został wykonany. W przypadku kiedy zostanie znaleziony odpowiedni stan sygnalizujący wykonanie poprzedniego zadania system nie zareaguje i pozwoli na dalsze przetwarzanie danych.

Jeśli jednak stan nie zostanie odnaleziony to aplikacja wykrywając błąd przekierowuje użytkownika na stronę główną.

Implementacja

Metoda ostrzegania

Dodanie wydarzenia

Pierwszym krokiem jest wypełnienie przez użytkownika formularza.

```
<form method="post" action="/addEvent" enctype="multipart/form-data">
  <input type="date" name="date"><br>
  <input type="time" name="time"><br>
  <input type="text" name="place" placeholder="miejsce"><br>
  <input type="text" name="title" placeholder="tytuł"><br>
  <input type="text" name="type" placeholder="typ"><br>
  <input type="text" name="desc" placeholder="opis"><br>
  <input type="number" name="priceu" placeholder="cena ulgowa"><br>
  <input type="number" name="pricen" placeholder="cena normalna"><br>
  <input type="number" name="il" placeholder="ilość biletów"><br>
  <input type="file" name="fileName"><br><br><br>
  <input type="submit" value="dodaj">
</form>
```

plik addEvent.jsp

Następnie dane z formularza są kierowane pod *servlet controller* o ścieżce */addEvent* gdzie system kolejno:

- Pobiera dane z formularza
- Zapisuje dane do bazy jeśli wydarzenie jest z przyszłą datą (dodanie wydarzenia z datą przeszłą jest niemożliwe)
- Zapisuje plik zdjęciowy do wydarzenia
- Ustawia zmienną zapytania na datę dodanego wydarzenia
- Przekierowuje działanie systemu do strony z informacją o dodaniu wydarzenia

```
package Controller;

import ...

@WebServlet(name = "dodaj event", urlPatterns = {"/addEvent"})
public class AddEventController extends HttpServlet {

    private static final long serialVersionUID = 1L;
    private ServletFileUpload uploader = null;
```

```

@Override
public void init() throws ServletException{
    DiskFileItemFactory fileFactory = new DiskFileItemFactory();
    File filesDir = (File) getServletContext().getAttribute("FILES_DIR_FILE");
    fileFactory.setRepository(filesDir);
    this.uploader = new ServletFileUpload(fileFactory);
}

@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {

    request.setCharacterEncoding("UTF-8");
    response.setCharacterEncoding("UTF-8");

    String Udate="";
    String Utime="";
    String Uplace="";
    String Utitle="";
    String Utype="";
    String Udesc="";
    String Upriceu="";
    String Upricen="";
    String Uil="";
    String is = "nie dodane";

    if(!ServletFileUpload.isMultipartContent(request)){
        throw new ServletException("Content type is not multipart/form-data");
    }

    response.setContentType("text/html");
    try {
        List<FileItem> fileItemsList = uploader.parseRequest(request);
        Iterator<FileItem> fileItemsIterator = fileItemsList.iterator();
        while(fileItemsIterator.hasNext()){

            FileItem fileItem = fileItemsIterator.next();

            Udate = fileItem.getString("UTF-8").trim();fileItem = fileItemsIterator.next();
            Utime = fileItem.getString("UTF-8").trim();fileItem = fileItemsIterator.next();
            Uplace = fileItem.getString("UTF-8").trim();fileItem = fileItemsIterator.next();
            Utitle = fileItem.getString("UTF-8").trim();fileItem = fileItemsIterator.next();
            Utype = fileItem.getString("UTF-8").trim();fileItem = fileItemsIterator.next();
            Udesc = fileItem.getString("UTF-8").trim();fileItem = fileItemsIterator.next();
            Upriceu = fileItem.getString("UTF-8").trim();fileItem = fileItemsIterator.next();
            Upricen = fileItem.getString("UTF-8").trim();fileItem = fileItemsIterator.next();
            Uil = fileItem.getString("UTF-8").trim();fileItem = fileItemsIterator.next();

            System.out.println(Udate);

            if(checkPattern.isDateHigher(Udate)) {
                System.out.println("in");
            }
        }
    }
}

```

```

        String mide = EventPhotoDAO.addEvent(request, Udate, Utime, Uplace, Utitle, Utype, Udesc,
        Upriceu, Upricen, Uil);

        File file = new
File("M:\\DOKUMENTY\\STUDIA\\PK\\III\\JAVA\\WEBPROJEKTGALERIA\\maven\\mavenstart\\src\\main\\webapp\\files\\
e_" + mide + ".jpg");
        fileItem.write(file);
        is = "dodane";
    }
}
} catch (FileUploadException e) {
    System.out.println((e));
} catch (Exception e) {
    System.out.println((e));
}

request.setAttribute("Udate", Udate);
RequestDispatcher rd = request.getRequestDispatcher("addedEvent.jsp");
rd.forward(request, response);

}
}

```

plik AddEventController.java

W ostatnim kroku strona z informacją o dodaniu wyświetla tekst o powodzeniu dodania do bazy oraz opcjonalnie w razie potrzeby wydarzenia z tego samego dnia wraz z ostrzeżeniem

- Pobiera atrybut zawierający datę z requesta zapytania
- sprawdza czy data rzeczywiście została przekazana
- Tworzy połączenie do bazy i wyszukuje pasujące wydarzenia
- W przypadku znalezienia wydarzeń z tego samego dnia są one pobierane z bazy danych oraz wyświetlane
- W przypadku nie odnalezienia wydarzeń z tego samego dnia jest wyświetlana tylko informacja o dodaniu wydarzenia

```

<%
String Udate = (String) request.getAttribute("Udate");

if(Udate != null){
    Connection conn = null;
    try {
        //łącze się do bazy danych
        conn = DriverManager.getConnection(
            "jdbc:mysql://localhost/galeria?useUnicode=true&characterEncoding=UTF-8",
            "root", ""
        );
        String JDBS_DRIVER = "com.mysql.jdbc.Driver";

```


Metoda koniecznego kroku

Zakup biletów

Pierwszym krokiem jaki musi wykonać użytkownik jest wypełnienie formularza. Wraz z wysłaniem formularza do następnej części systemu jest ustawiana zmienna *state*, która informuje w jakim stanie znajduje się system.

Część odpowiedzialna za ustawienie stanu

```
<input type="hidden" name="state" value="state1" ><br> <!--ustawienie stanu-->
```

plik buy.jsp

Cały kod formularza

```
<form method="get" action="/bought">

    <%
        if(request.getParameter("typ") != null && request.getParameter("typ").equals("bilet_u")){
            out.println("\n" +
                "                                <input type=\"number\" name=\"ilosc\" placeholder=\"ilosc
ulgowych\" value=\"\"+request.getParameter("ilosc")+"\"><br>");

        }
        else{
            out.println("\n" +
                "                                <input type=\"number\" name=\"ilosc\" placeholder=\"ilosc
ulgowych\"><br>");
        }

        if(request.getParameter("typ") != null && request.getParameter("typ").equals("bilet_n")){
            out.println("\n" +
                "                                <input type=\"number\" name=\"ilosc\" placeholder=\"ilosc
normalnych\" value=\"\"+request.getParameter("ilosc")+"\"><br>");
        }
        else{
            out.println("\n" +
                "                                <input type=\"number\" name=\"ilosc\" placeholder=\"ilosc
normalnych\"><br>");
        }
    %>

    <input type="hidden" name="state" value="state1" ><br> <!--ustawienie stanu-->
    <input type="hidden" name="id" value="<%out.print(id);%>" ><br>
    <input type="hidden" name="k" value="<%out.print(request.getParameter("k"));%>" ><br>
    <input type="submit" value="kup"><br>
</form>
```

plik buy.jsp

Kolejny krok obsługuje servlet controller o ścieżce */bought*, który sprawdza czy poprzedni stan został ustalony oraz na jego podstawie decyduje o przetwarzaniu otrzymanych danych oraz ustawieniu kolejnego stanu bądź następuje przekierowanie do strony głównej.

Część odpowiedzialna za sprawdzenie stanu

```
String state = (String)request.getParameter("state");
if(state == null || !state.equals("state1")){
    System.out.println("a"+state);
    RequestDispatcher rd = request.getRequestDispatcher("index.jsp");
    rd.forward(request, response);
}
```

plik BoughtController.java

Część odpowiedzialna za ustawienie nowego stanu

```
request.setAttribute("state", new String("state2"));
```

plik BoughtController.java

Cały kod kontrolera

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

    //sprawdzanie stanu
    String state = (String)request.getParameter("state");
    if(state == null || !state.equals("state1")){
        System.out.println("a"+state);
        RequestDispatcher rd = request.getRequestDispatcher("index.jsp");
        rd.forward(request, response);
    }
    //koniec sprawdzania stanu

    //ustawienie nowego stanu
    request.setAttribute("state", new String("state2"));

    HttpSession session = request.getSession();

    String idevent = request.getParameter("id");
    String idclient = (String)session.getAttribute("id");
    String ilulgowie = request.getParameter("iloscu");
    String ilnormalne = request.getParameter("iloscn");
```

```

if(ilulgowce.equals("")) ilulgowce="0";
if(ilnormalne.equals("")) ilnormalne="0";

String il = Integer.toString(Integer.valueOf(ilulgowce)+Integer.valueOf(ilnormalne));

EventPhotoDTO event = EventPhotoDAO.getEventPhotoByID(Integer.parseInt(idevent));

int ilbilet = Integer.valueOf(event.getIlosc_biletow()) - Integer.valueOf(il);

String qr;
if(ilbilet >= 0 && (!ilulgowce.equals("0") || !ilnormalne.equals("0"))){
    qr = TicketDAO.buyTicket(idclient, idevent, ilulgowce, ilnormalne, ilbilet);
    request.setAttribute("qr", qr);
    RequestDispatcher rd = request.getRequestDispatcher("bought.jsp");
    rd.forward(request, response);
}
else{
    request.setAttribute("brak", "brak wystarczającej ilości biletów");
    RequestDispatcher rd = request.getRequestDispatcher("bought.jsp");
    rd.forward(request, response);
}
}

```

plik BoughtController.java

Ostatni krok to wyświetlenie zakupionych biletów lub przekierowanie do strony głównej w przypadku braku ustalenia odpowiedniego stanu.

Cały kod sprawdzający stan

```

<%
    String state = (String)request.getAttribute("state");
    if(state == null || !state.equals("state2")){
        System.out.println("b"+state);
        state = (String)request.getAttribute("state");
        RequestDispatcher rd = request.getRequestDispatcher("index.jsp");
        rd.forward(request, response);
    }
%>

```

plik bought.jsp

Rezerwacja biletów

Przed wszystkim użytkownik musi wypełnić formularz. Wraz z danymi formularza jest wysyłana informacja o stanie, który został ustawiony przez formularz.

Część odpowiedzialna za ustawienie stanu

```
<input type="hidden" name="state" value="state1" ><br><!--ustawienie stanu-->
```

plik reservation.jsp

Kolejnym krokiem jest obsługa danych z formularza przez odpowiedni controller. W pierwszej kolejności sprawdza on ustawienie stanu i jeśli jest on poprawny pozwala na dalsze przetwarzanie danych. W przypadku kiedy stan nie został ustalony następuje przekierowanie na stronę główną

Część odpowiedzialna za sprawdzenie stanu

```
String state = (String)request.getParameter("state");
if(state == null || !state.equals("state1")){
    System.out.println("a"+state);
    RequestDispatcher rd = request.getRequestDispatcher("index.jsp");
    rd.forward(request, response);
}
```

plik ReservedController.java

Część odpowiedzialna za ustawienie nowego stanu

```
request.setAttribute("statee", new String("state2"));
```

plik ReservedController.java

Cały kod kontrolera

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

    String state = (String)request.getParameter("state");
    if(state == null || !state.equals("state1")){
        System.out.println("a"+state);
        RequestDispatcher rd = request.getRequestDispatcher("index.jsp");
```

```

        rd.forward(request, response);
    }
    request.setAttribute("statee", new String("state2"));

    HttpSession session = request.getSession();

    String idevent = request.getParameter("id");
    String idclient = (String)session.getAttribute("id");
    String ilulgowe = request.getParameter("ilosc_u");
    String ilnormalne = request.getParameter("ilosc_n");

    if(ilulgowe.equals("")) ilulgowe="0";
    if(ilnormalne.equals("")) ilnormalne="0";

    String il = Integer.toString(Integer.valueOf(ilulgowe)+Integer.valueOf(ilnormalne));

    EventPhotoDTO event = EventPhotoDAO.getEventPhotoByID(Integer.parseInt(idevent));

    int ilbilet = Integer.valueOf(event.getIlosc_biletow()) - Integer.valueOf(il);

    String qr;
    if(ilbilet >= 0){
        TicketDAO.reserveTicket(idclient, idevent, Integer.valueOf(ilulgowe),
Integer.valueOf(ilnormalne), ilbilet);
        request.setAttribute("state", "zarezerwowałeś bilety");
    }
    else{
        request.setAttribute("state", "Brak biletów. Zostało "+event.getIlosc_biletow());
    }
    RequestDispatcher rd = request.getRequestDispatcher("cart.jsp");
    rd.forward(request, response);
}

```

plik ReservedController.java

Ostatnim krokiem jest wyświetlenie informacji o rezerwacji biletów. W przypadku braku odpowiedniego stanu strona przekierowywana jest na stronę główną.

Kod odpowiedzialny za sprawdzenie stanu

```

String state = (String)request.getAttribute("statee");
if(state == null || !state.equals("state2")){
    System.out.println("b"+state);
    RequestDispatcher rd = request.getRequestDispatcher("index.jsp");
    rd.forward(request, response);
}

```

plik reserved.jsp

Dodawanie biletów do koszyka

Dodawanie biletu do koszyka także wymaga implementacji systemu odpowiedzialnego za sprawdzenie czy poprzedni krok został wykonany. W tym przypadku jednak wymagane są jedynie dwa kroki. Najpierw użytkownik wypełnia formularz, a następnie system zostaje przekierowany do kontrolera, który odpowiada za przetwarzanie danych.

Część odpowiedzialna za ustawienie pola ze stanu

```
<input type="hidden" name="state" value="state1" ><br><!--ustawia stan-->
```

plik addCart.jsp

Całość formularza

```
<form method="get" action="/confirmCard">
    <input type="hidden" name="state" value="state1" ><br><!--ustawia stan-->
    <input type="number" name="iloscu" placeholder="ilość ulgowych"><br>
    <input type="number" name="iloscn" placeholder="ilość normalnych"><br>
    <input type="hidden" name="id" value="%out.print(event.getID_Eventt());%" ><br>
    <input type="submit" value="dodaj"><br>
</form>
```

plik addCart.jsp

Kontroler przyjmuje Dane z formularza wraz ze stanem po czym sprawdza otrzymany stan. W przypadku pomyślnego sprawdzenia stanu system działa dalej bez przeszkód. Jeśli stan nie zostanie ustalony to następuje przekierowanie do strony głównej

Część odpowiedzialna za sprawdzenie stanu

```
String state = (String)request.getParameter("state");
if(state == null || !state.equals("state1")){
    RequestDispatcher rd = request.getRequestDispatcher("index.jsp");
    rd.forward(request, response);
}
```

plik ConfirmCardController.java

Przykładowe działanie programu

Dodawanie wydarzenia

Po dodaniu wydarzenia zostajemy przekierowani na stronę z informacją o dodaniu wydarzenia. Jeśli dodamy wydarzenie z inną datą ale taką żeby nie było kolidujących wydarzeń to zostanie jedynie wyświetlona informacja o dodaniu wydarzenia.

DODAJ WYDARZENIE

book-4835623_19201.jpg

DODANO WYDARZENIE

Event został dodany

Galeria GalART © 2019



SEN

Spotkanie

Sala wystawowa ul. Sobieskiego

2020-06-13 01:00

cena normalna: 30.0

cena ulgowa: 20.0

(..)Granica między światłem i cieniem jest jednym z najważniejszych wyróżników w malarstwie Dariusza Milczarka (...) Swoje kompozycje kreuje bazując na wykonanych przez siebie zdjęciach. Zaskakująca jest deklaracja autora gdy w fotografii widzi „możliwość zwolnienia tempa, zatrzymania zmienności”, co poza oczywistymi funkcjami tego medium i szaleńczego dynamizmu obrazowania zapowiada refleksyjność, zastanowienie, nawet kontemplację. W tej deklaracji posuwa się jeszcze dalej, bo przy dużej wiedzy, świadomości czym jest dla niego fotografia, jaki jest jej potencjał i możliwości totalnego działania, opowiada się za malarstwem jako medium dla siebie najważniejszym. Malarstwo według niego daje nieograniczoną swobodę wypowiedzi, wolność. Taka postawa jest wbrew powszechnym tendencjom, modom i godna szacunku, tym bardziej, że to świadoma decyzja, bezkompromisowa na rzecz jakże ważnej, kto wie czy nie najważniejszej wartości jaką jest bezinteresowne wypowiedzenie własnej prawdy o sobie i świecie, tu za pomocą malarstwa.

Galeria GalART © 2019

Jeśli dodajemy wydarzenie w terminie, w którym inne zostało już dodane. Zostaje wyświetlona informacja o współistniejących wydarzeniach oraz wydarzenia z tego samego dnia.

DODAJ WYDARZENIE

Choose File

books-1245690_19201.jpg

dodaj

Galeria GalART © 2019

DODANO WYDARZENIE

Event został dodany ale występują eventy które mogą kolidować



SEN

Spotkanie

cena od: 30.0

(..).Granica między światłem i cieniem jest jednym z najważniejszych wyróżników w malarstwie Dariusz...

Kupno biletu

Formularz kupna biletu



SEN

Spotkanie

Sala wystawowa ul. Sobieskiego

cena regularna: 30.0

cena ulgowa: 20.0

Bilet poprawnie zakupiony



Rezerwacja biletu

Formularz rezerwacji

SEN

Spotkanie

Sala wystawowa ul. Sobieskiego

cena normalna: 30.0

cena ulgowa: 20.0

rezerwuj

Potwierdzenie rezerwacji

zarezerwowałeś bilety


Dodawanie biletów do koszyka

Stan początkowy koszyka

KOSZYK

Brak rzeczy w koszyku

Dodanie biletów do koszyka



KRAKÓW

cena regularna: 20.0

cena ulgowa: 15.0

Uwiedziony spojrzeniem łani zabrnięte
jamy, do których trzeba zejść i zajrzeć
której można okrzepnąć w promieniach
rozdrapanym gałęziami skrawku nieba

10

2

dodaj

Stan koszyka

KOSZYK



KRAKÓW

Ulgowe: 10

Normalne: 2

wykup zrezygnuj

Testy

Dodawanie wydarzenia

Najpierw dodajemy wydarzenie

DODAJ WYDARZENIE

13/06/2020

01:00

Sala czerwona

Senny

Spotkanie

..

40

45

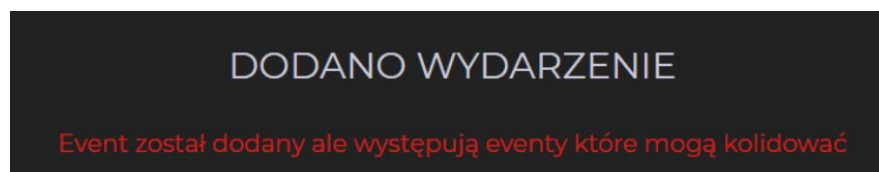
500

Choose File books-1245690_19201.jpg

dodaj

Galeria GalART © 2019

Po dodaniu wydarzenia zostajemy przekierowani na stronę z informacją o dodaniu wydarzenia. Zostaje wyświetlona informacja o współistniejących wydarzeniach oraz wydarzenia z tego samego dnia.

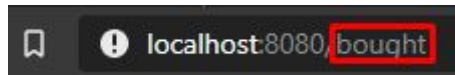




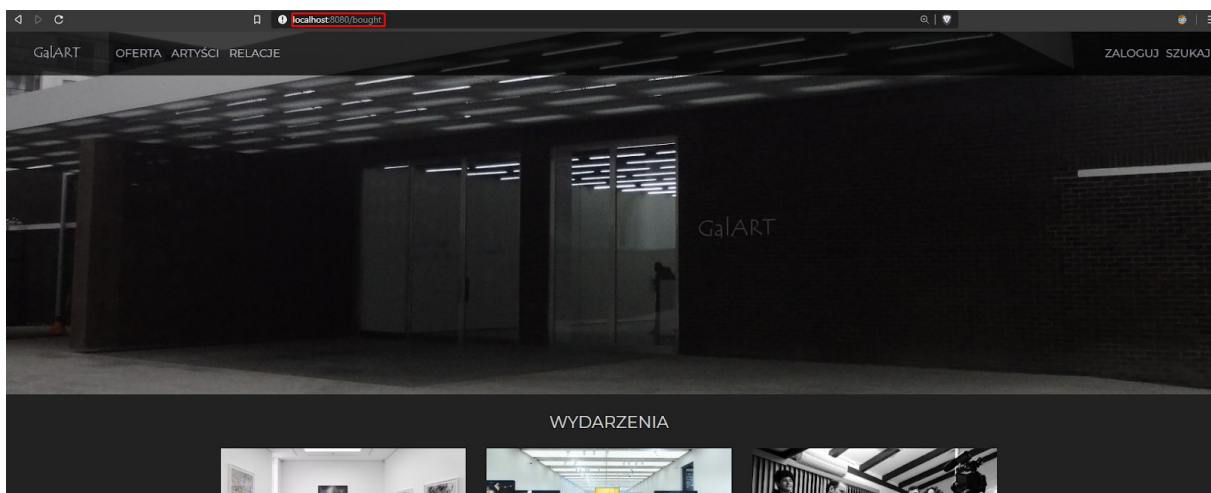
Kupno biletu

Jak zostało pokazane wyżej kupno biletu działa poprawnie jeśli jest przeprowadzanie w prawidłowy sposób. W przypadku próby przejścia pod adres wymagający poprzedniego stanu system powinien przekierować użytkownika do strony głównej.

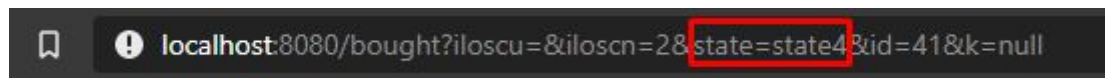
Próba przejścia po adres kontrolera odpowiedzialnego za zakup biletów bez żadnego stanu



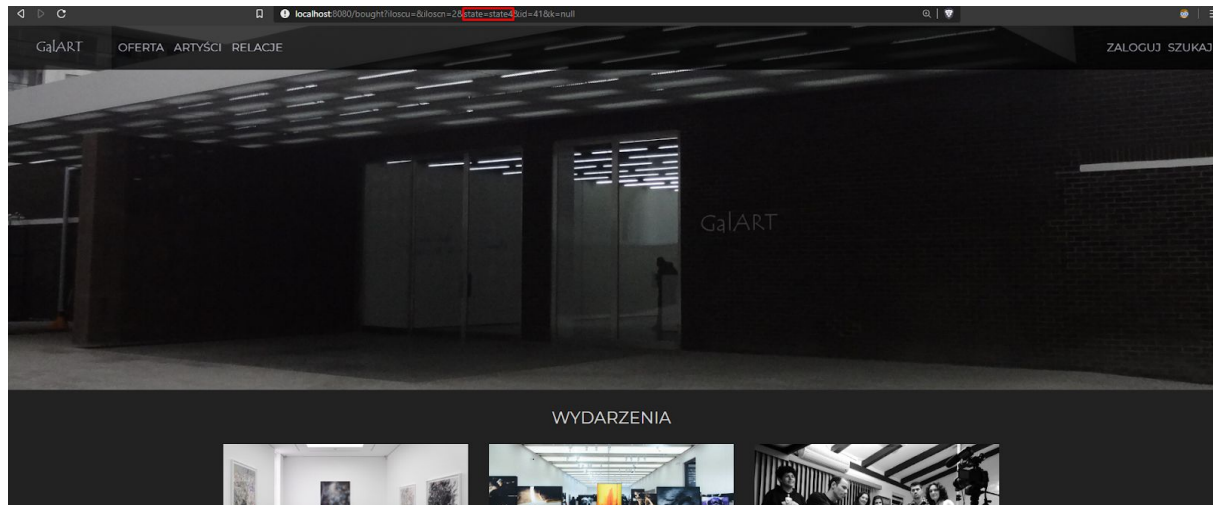
Nastąpiło przekierowanie do strony głównej



Próba przejścia z ustawionym błędnym stanem

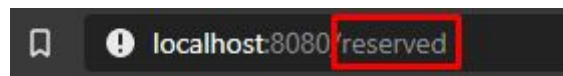


Spowodowało to ten sam błąd i w rezultacie przejście do strony głównej

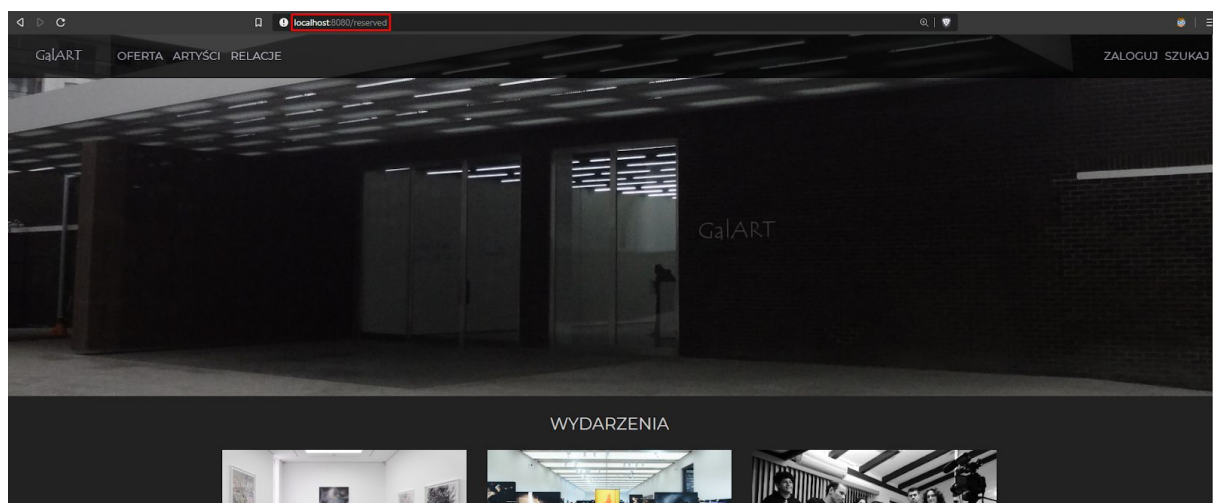


Rezerwacja biletu

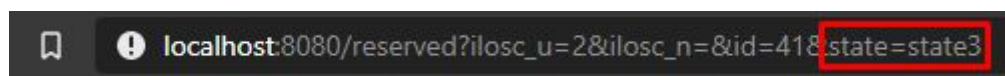
Próba przejścia pod adres kontrolera



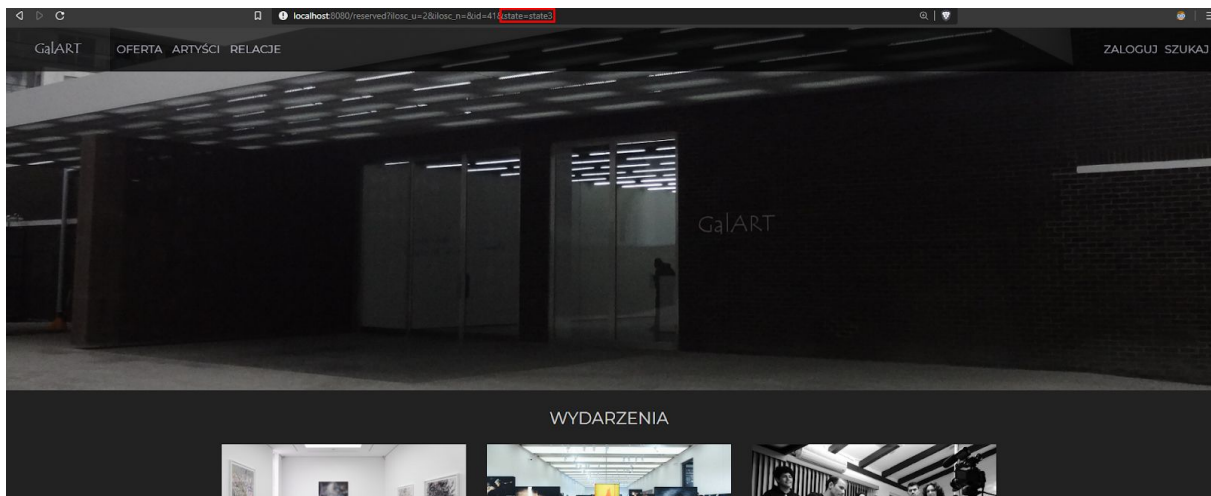
Po wykonaniu takiego zapytania użytkownik przekierowany jest na stronę główną.



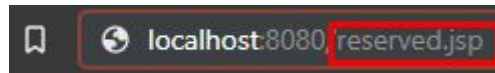
Próba przejścia z błędnym stanem



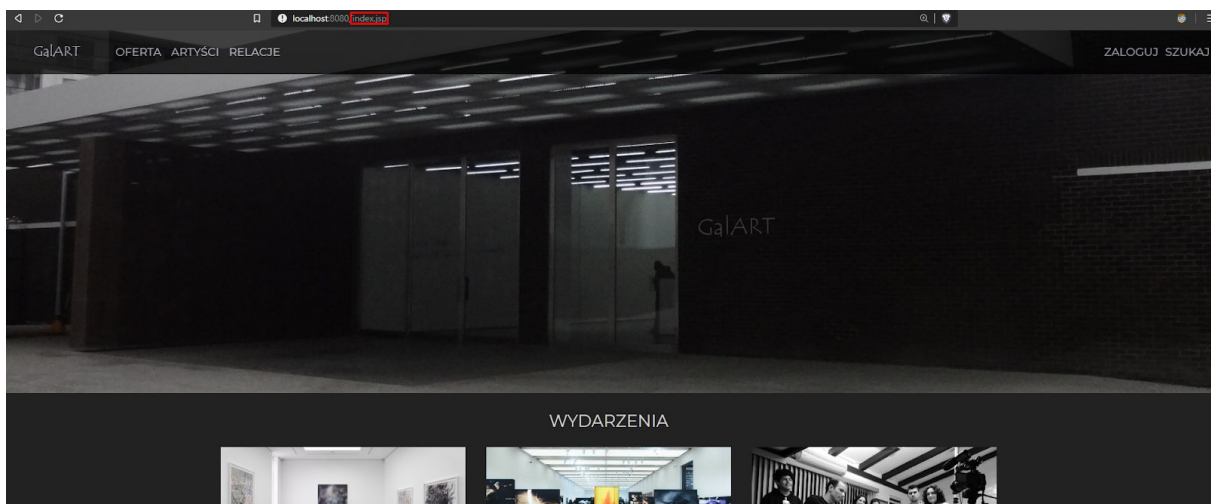
Po wykonaniu takiego zapytania użytkownik przekierowany jest na stronę główną.



Próba przejścia pod plik informujący o powodzeniu rezerwacji

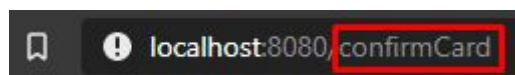


Po wykonaniu takiego zapytania użytkownik przekierowany jest na stronę główną.

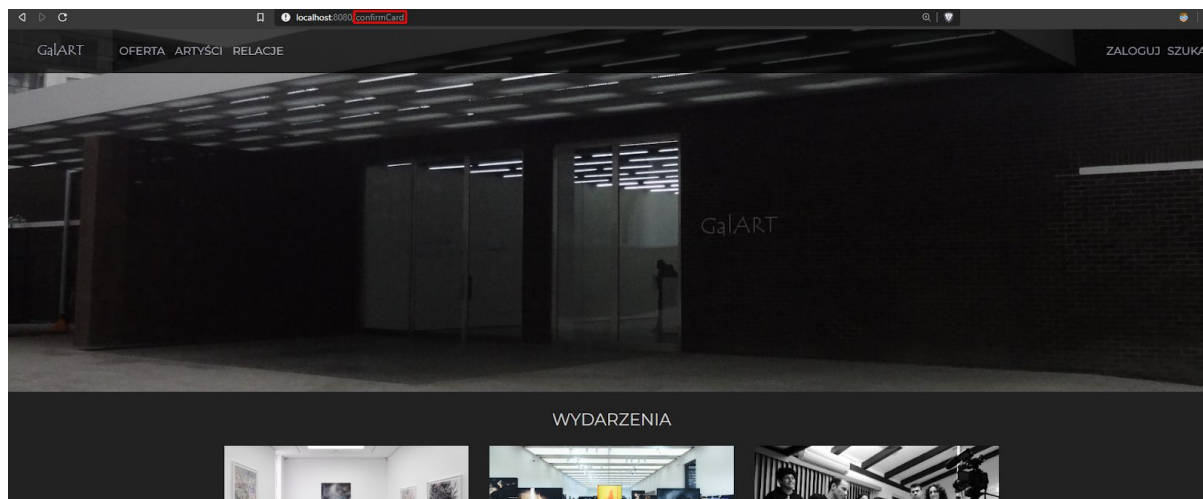


Dodanie biletów do koszyka

Próba przejścia pod adres kontrolera



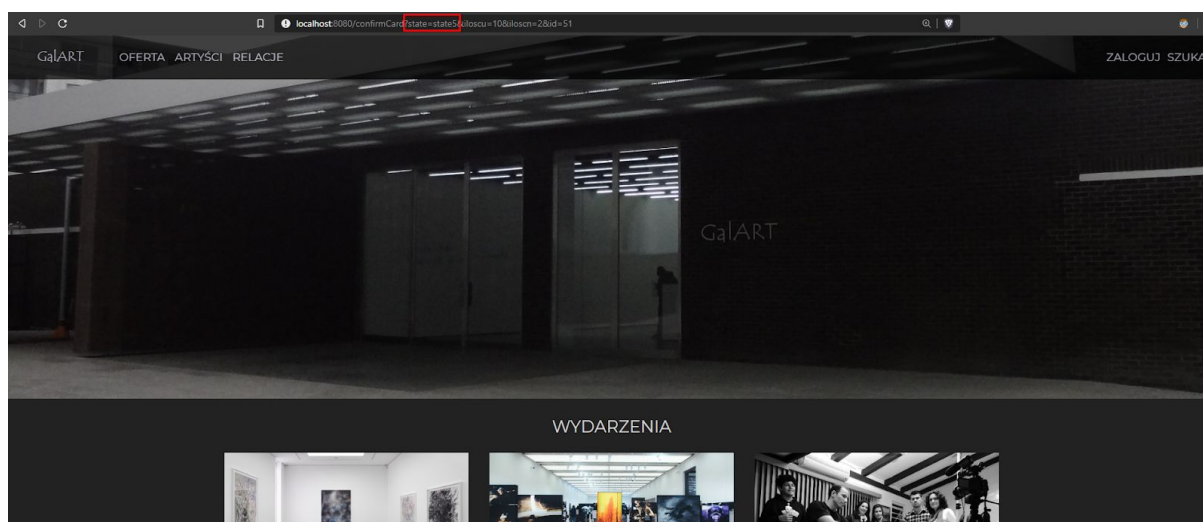
Po wykonaniu takiego zapytania użytkownik przekierowany jest na stronę główną.



Próba przejścia z błędnym stanem



Po wykonaniu takiego zapytania użytkownik przekierowany jest na stronę główną.



Otrzymane wyniki i wnioski

Ze względu na nieskomplikowane metody wszystkie zaimplementowane elementy działały bez zarzutu w momencie przeprowadzenia testów. Staraliśmy się przeprowadzić wszystkie możliwe testy w trakcie sprawdzania działania obu metod tak by spełniały wymagania zaimplementowanych metod.

Po przeprowadzeniu testów doszliśmy do wniosków, że użyte funkcje po prostu działają. Dodatkowym atutem jest łatwa rozszerzalność metody koniecznego kroku o dodatkowe czynności, które powinny znaleźć się na początku, końcu bądź wewnątrz już istniejącej implementacji.

Wykorzystane materiały, narzędzia, literatura

Bibliografia

- Gandalf, “Maszyny stanów na tablicach” [online]
<https://ucgosu.pl/2019/08/implementacja-maszyn-stanu-na-tablicach> [dostęp: 20.05.2020].
- Stęposz Ewa, Płodzień Jakub, “Projektowanie Systemów Informacyjnych : Wykłady” [online] <https://edu.pjwstk.edu.pl/wyklady/pri/scb/index99.html> [dostęp: 20.05.2020].
- Tochman Rafał, “POKA-YOKE” [online] <https://www.jakosc.biz/poka-yoke> [dostęp: 20.05.2020].

Narzędzia

- zintegrowane środowisko programistyczne **JetBrains: IntelliJ IDEA**
- narzędzie automatyzujące budowę oprogramowania na platformę Java **Maven**
- zintegrowany pakiet **Xampp**

Załącznik: Uruchomienie projektu

Aby uruchomić projekt należy posiadać wyżej wymienione narzędzia. Pierwszym krokiem jest importowanie bazy danych, której eksport znajduje się w pliku z rozszerzeniem *.sql*. W celu uruchomieniu projektu należy otworzyć go za pomocą narzędzia **IntelliJ IDEA**, a następnie wybrać zakładkę *maven*. Kolejnym krokiem jest wybranie listy *pluggins* → *jetty* → *jetty:run*.