Warsztaty samochodowy

Warsztat na wolny tydzień

v3.1

Cel warsztatów

- Celem warsztatów jest wykonanie projektu systemu **CRM** (wikipedia) dla Warsztatu samochodowego.
- Projekt ma wykorzystywać (servlety oraz JSP).
- Projekt ma wykorzystywać warstwę dostępu do danych w postaci klas **DAO**.
- Projekt powinien oprogramowywać opisane funkcjonalności.

Projekt może zostać rozwinięty o dowolny zestaw własnych funkcjonalności, przykład:

- powiadomienia email (np. zbliżający się koniec przeglądu),
- rezerwacja klienta na wizyty serwisowe,
- dodatkowe raporty.

Funkcjonalności aplikacji

Zarządzanie klientami

Aplikacja ma mieć następujące możliwości:

- przeglądanie klientów,
- dodawanie klienta,
- usuwanie klienta,
- edycję klienta.

Zarządzanie pojazdami klientów

Aplikacja ma mieć następujące możliwości:

- przeglądanie pojazdów,
- dodawanie pojazdu,
- usuwanie pojazdu,
- edycję pojazdu,
- przypisanie pojazdu do klienta.

Funkcjonalności aplikacji

Zarządzanie pracownikami

Aplikacja ma mieć następujące możliwości:

- dodawanie pracownika,
- przeglądanie pracowników,
- usuwanie pracownika,
- edycję pracownika,
- przeglądanie zleceń danego pracownika.

Zarządzanie zleceniami

Aplikacja ma umożliwiać zarządzanie zleceniami:

- dodawanie zlecenia,
- usuwanie zlecenia,
- edycję zlecenia.
- zmianę statusu,
- szczegóły zlecenia.

Funkcje aplikacji (User Stories)

Jako użytkownik aplikacji mam możliwość dodania nowego pracownika.

Pracownik powinien posiadać przynajmniej zestaw danych:

- Imię
- Nazwisko
- Dane adresowe
- Nr telefonu
- Notatka
- Koszt roboczogodziny

Funkcje aplikacji (User Stories)

Jako użytkownik aplikacji mam możliwość dodania nowego klienta.

Powinien posiadać przynajmniej zestaw danych:

- Imię
- Nazwisko
- Data urodzenia (opcjonalnie - będziemy mogli przesłać mu życzenia urodzinowe)

Funkcje aplikacji (User Stories)

Jako użytkownik aplikacji mam możliwość przypisania do klienta posiadanego przez niego pojazdu.

Pojazd powinien posiadać przynajmniej zestaw danych:

- Model
- Marka
- Rok produkcji
- Numer rejestracyjny
- Data kolejnego przeglądu technicznego

Funkcje aplikacji (User Stories)

Jako użytkownik aplikacji mam możliwość dodania nowego zlecenia.

Zlecenie powinno posiadać przynajmniej zestaw danych:

- Data przyjęcia do naprawy
- Planowana data rozpoczęcia naprawy
- Data rozpoczęcia do naprawy
- Przypisany do naprawy pracownik
- Opis problemu
- Opis naprawy
- Status
- Pojazd którego dotyczy naprawa
- Koszt naprawy dla klienta
- Koszt wykorzystanych części
- Koszt roboczogodziny (informacja przepisywana z kosztu roboczogodziny pracownika wykonującego naprawę)
- Ilość roboczogodzin

Funkcje aplikacji (User Stories)

Jako użytkownik aplikacji mam możliwość zmiany statusu danej naprawy.

Proponowany zestaw statusów:

- Przyjęty
- Zatwierdzone koszty naprawy
- W naprawie
- Gotowy do odbioru
- Rezygnacja

Funkcje aplikacji (User Stories)

Jako użytkownik aplikacji mam możliwość wyszukiwania klienta po nazwisku.

Następnie mogę przeglądać pojazdy danego klienta i przejść do historii napraw konkretnego pojazdu.

Historia napraw powinna zawierać pola: data (**Data rozpoczęcia naprawy)** i opis (**Opis naprawy**) (z tabeli reprezentującej zlecenie).

Mam również możliwość przejścia do szczegółów zlecenia - wyświetlają się wtedy wszystkie dane.

Funkcje aplikacji (User Stories)

Jako użytkownik aplikacji mam możliwość przeglądania pracowników.

Mogę przeglądać zlecenia przypisane aktualnie dla danego pracownika (wykorzystaj status **W naprawie**) .

Funkcje aplikacji (User Stories)

Raport 1

Aplikacja ma umożliwiać wyświetlenie raportu przepracowanych roboczogodzin przez wszystkich pracowników, z możliwością określenia **daty od** oraz **daty do**.

Dla uproszczenia przyjmujemy że wykorzystujemy w tym celu **Datę rozpoczęcia naprawy.**

Przykład:

Imię i nazwisko pracownika	**Ilość roboczogodzin**
Arek Arecki	210
Darek Darecki	180
Marek Marek	173

Funkcje aplikacji (User Stories)

Raport 2

Aplikacja ma umożliwiać wyświetlenie uproszczonego raportu zysków z działalności z możliwością określenia **daty od** oraz **daty do**.

Dla uproszczenia przyjmujemy że wykorzystujemy w tym celu **Datę zakończenia naprawy.**

Raport jest uproszczony, ponieważ nie uwzględnia kosztów stałych prowadzenia działalności, najmów, kosztów inwestycji.

Strony, które musi mieć aplikacja

Strona główna aplikacji

Ma mieć możliwość przejścia do poszczególnych elementów aplikacji (nawigacja w postaci linków).

Wyświetlać aktualnie prowadzone naprawy przez każdego z pracowników z możliwością przejścia do konkretnego zlecenia.

Strona klienci

Strona ma wyświetlać listę klientów grup z możliwością przejścia do szczegółów z listą wszystkich pojazdów.

Strony, które musi mieć aplikacja

Strona zlecenia

Ta strona ma wyświetlać listę wszystkich zleceń sortowanych od najnowszych do najstarszych grupy z możliwością przejścia do szczegółów danego zlecenia.

Strona pracownicy

Strona ma wyświetlać wszystkich pracowników z możliwością wyświetlenia danych szczegółowych.

Ekran ten ma również udostępniać możliwość przejścia do ekranu ze zleceniami danego pracownika.

Strona raporty

Strona ma wyświetlać możliwość wybrania typu raportu.

Na kolejnym ekranie wybieramy interesujący nas przedział czasowy dla danego raportu.

Przygotowanie

Przygotowanie

Zadanie: przygotowanie

- Załóż repozytorium na GitHubie.
- Podepnij swoje nowe projekty do repozytorium i zobacz, czy działa (np. przez dodanie pliku **`readme`** na GitHubie i ściągnięciu go na oba komputery).

Zadanie: połączenie do bazy danych

Do stworzenia połączenia wykorzystamy kontener servletów Tomcat oraz odpowiednio skonfigurowany zasób DataSource.

W tym celu należy stworzyć plik **`context.xml`** w lokalizacji META-INF projektu.

Szczegółowe informacje o możliwościach konfiguracyjnych możemy znaleźć tutaj: http://tomcat.apache.org/tomcat-8.0-doc/jdbc-pool.html

Przygotowanie

Zadanie: połączenie bo bazy danych

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
  <Resource name="jdbc/crm"
            auth="Container"
            type="javax.sql.DataSource"
            username="root"
            password="root"
            driverClassName="com.mysql.jdbc.Driver"
            url="jdbc:mysql://localhost:3306/crm"
            connectionProperties="useUnicode=yes;characterEncoding=utf8;"
            maxTotal="100"
            maxIdle="30"
            maxWaitMillis="10000" />
</Context>
```

Przygotowanie

Zadanie: połączenie bo bazy danych

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
  <Resource name="jdbc/crm"
            auth="Container"
            type="javax.sql.DataSource"
            username="root"
            password="root"
            driverClassName="com.mysql.jdbc.Driver"
            url="jdbc:mysql://localhost:3306/crm"
            connectionProperties="useUnicode=yes;characterEncoding=utf8;"
            maxTotal="100"
            maxIdle="30"
            maxWaitMillis="10000" />
</Context>
```

jdbc/crm – nazwa zasobu, z którego będziemy z niej korzystali.

Przygotowanie

Zadanie: połączenie bo bazy danych

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
  <Resource name="jdbc/crm"
            auth="Container"
            type="javax.sql.DataSource"
            username="root"
            password="root"
            driverClassName="com.mysql.jdbc.Driver"
            url="jdbc:mysql://localhost:3306/crm"
            connectionProperties="useUnicode=yes;characterEncoding=utf8;"
            maxTotal="100"
            maxIdle="30"
            maxWaitMillis="10000" />
</Context>
```

Dane autoryzacyjne do bazy danych.

Przygotowanie

Zadanie: połączenie bo bazy danych

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
  <Resource name="jdbc/crm"
            auth="Container"
            type="javax.sql.DataSource"
            username="root"
            password="root"
            driverClassName="com.mysql.jdbc.Driver"
            url="jdbc:mysql://localhost:3306/crm"
            connectionProperties="useUnicode=yes;characterEncoding=utf8;"
            maxTotal="100"
            maxIdle="30"
            maxWaitMillis="10000" />
</Context>
```

Adres **`url`** bazy danych.

Przygotowanie

Zadanie: połączenie bo bazy danych

Dobrą praktyką jest wydzielenia kodu odpowiedzialnego za tworzenie połączenia.

Przykład takiej klasy korzystającej z zasobu, który wcześniej zdefiniowaliśmy, mamy poniżej.

Pozwoli nam to skupić się bezpośrednio na funkcjonalności.

Przygotowanie

Zadanie: połączenie bo bazy danych

```
public class DbUtil {
    private static DataSource ds;
    public static Connection getConn() throws SQLException {
        return getInstance().getConnection();
    }
    private static DataSource getInstance() {
        if(ds == null) {
            try {
                Context ctx = new InitialContext();
                ds = (DataSource)ctx.lookup("java:comp/env/jdbc/crm");
            } catch (NamingException e) {
                e.printStackTrace();}}
        return ds;}
}
```

Przygotowanie

Zadanie: połączenie bo bazy danych

```
public class DbUtil {
    private static DataSource ds;
    public static Connection getConn() throws SQLException {
        return getInstance().getConnection();
    }
    private static DataSource getInstance() {
        if(ds == null) {
            try {
                Context ctx = new InitialContext();
                ds = (DataSource)ctx.lookup("java:comp/env/jdbc/crm");
            } catch (NamingException e) {
                e.printStackTrace();}}
        return ds;}
}
```

java:comp/env/jdbc/crm – zasób określony wcześniej.

Przygotowanie

Zadanie: połączenie bo bazy danych

Aby utworzyć połączenie skorzystamy ze statycznej metody **`getConn()`** w następujący sposób:

```
Connection c = DbUtil.getConn();
```

Co zastępuje znane nam z zajęć o MySQL:

```
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/"
                                              + "crm?useSSL=false",
                                              "root",
                                              "coderslab");
```

Pozostały kod związany z połączeniem pozostanie bez zmian.

Zadania

Modele w aplikacji

Vehicle

Jest to klasa reprezentująca pojazd.

Employee

Pracownik naszego warsztatu.

Customer

Klient warsztatu.

Order

Zlecenie naprawy pojazdu.

Status

Status zlecenia.

Zadanie 1

Stwórz pakiet **`pl.coderslab.dao`**, umieścimy w nim klasy dostępu do danych .

Jest to wzorzec podobny do znanego już nam **`ActiveRecords`** z tą różnicą, że wszystkie metody związane z operacjami na obiekcie są wydzielone do oddzielnej klasy np. **`VehicleDao`**.

Wszystkie nasze metody dostępu do danych powinny się znajdować w nowo utworzonych klasach.

Są to np.:

- **`loadAllByUserId,`**
- **`loadById.`**

Zadanie 1

Klasy **`DAO`** (**`Data Access Object`**) powinny implementować przynajmniej zestaw operacji **`CRUD`**.

Skrót ten oznacza:

Create

Read

Update

Delete

https://en.wikipedia.org/wiki/Data_access_object

Zastanów się jakich metod będzie potrzebowała Twoja aplikacja i umieść je w odpowiednich klasach.

Zadanie 2

Pliki szablonu

- Utwórz pliki **`header.jsp`**, **`footer.jsp`** oraz plik **`index.jsp`**, który będzie je załączał – w ten sposób stworzymy szablon naszej aplikacji.
- W pliku **`header.jsp`** umieścimy linki nawigacyjne naszej aplikacji.
- W pliku **`footer.jsp`** umieścimy stopkę informacyjną.
- W metodzie **`doGet`** pierwszego servletu pobierz przy pomocy odpowiedniej metody klasy **DAO** listę aktualnie prowadzonych napraw.
- Przekaż pobraną listę do widoku **`index.jsp`**, a następnie wyświetl szczegóły zleceń w wierszach tabeli html.

Zadanie 2

Widoki możesz stworzyć z wykorzystaniem biblioteki **Bootstrap**.

Bootstrap znacznie poprawia wygląd strony a jego użycie jest bardzo proste.

Jeżeli również chcecie skorzystać z bootstrapa odsyłam do następujących stron:

- **http://getbootstrap.com/**
- **https://www.w3schools.com/bootstrap/**
- **https://kursbootstrap.pl/**

Zadanie 3

Lista pracowników

Utwórz **servlet** oraz widok w postaci pliku **jsp**.

Wykorzystując klasę **DAO** pobierz a następnie przekaż do widoku listę wszystkich pracowników .

Utwórz pozostałe akcje i widoki:

- dodawanie pracownika
- edycja pracownika
- usuwanie pracownika
- zlecenia pracownika

Zadanie 3

Lista klientów

Utwórz **servlet** oraz widok w postaci pliku **jsp**.

Wykorzystując klasę **DAO** pobierz a następnie przekaż do widoku listę wszystkich klientów .

Utwórz pozostałe akcje i widoki:

- dodawanie klienta
- edycja klienta
- usuwanie klienta
- pojazdy klienta
- zlecenia klienta

Zadanie 3

Pojazdy klientów

Utwórz **servlet** oraz widok w postaci pliku **jsp**.

Wykorzystując klasę **DAO** pobierz a następnie przekaż do widoku listę wszystkich pojazdów klienta.

Aby uzyskać pojazdy danego klienta wykorzystaj pobrany z żądania parametr **id** .

Utwórz pozostałe akcje i widoki:

- dodawanie pojazdu
- edycja pojazdu
- usuwanie pojazdu
- szczegóły pojazdu - wraz z listą wszystkich napraw

Zadanie 3

Lista zleceń

Utwórz **servlet** oraz widok w postaci pliku **jsp**.

Wykorzystując klasę **DAO** pobierz a następnie przekaż do widoku listę wszystkich zleceń.

Utwórz pozostałe akcje i widoki:

- dodawanie zlecenie
- edycja zlecenia
- usuwanie zlecenia
- szczegóły zlecenia

Zadanie 4

Raporty

Utwórz **servlet** oraz widok w postaci pliku **jsp**.

Wykorzystując klasę **DAO** pobierz a następnie przekaż do widoku wyniki specyficzne dla danego raportu.