

Dokumentace pokladna 3.0 - web

ZÁVĚREČNÝ PROJEKT
DVOŘÁK TOMÁŠ

Obsah

Návrh projektu.....	2
Vývojové prostředí.....	2
Databáze.....	2
Administrační aplikace	2
Administrační aplikace zahrnuje:	2
Objednávací panel.....	2
Panel přípravy	2
Panel u výdeje	2
Popis řešení projektu.....	3
Database connection.....	3
Práce s databází v Model.Polozka, Model.Pridavek, Model.Objednavka	3
Administrační aplikace	6
Vložení přídavku či položky	6
Úprava položky či přídavku.....	8
Prohlížení objednávek.....	8
Objednávací panel.....	10
Panel přípravy a panel výdeje	12
E-R diagram	16
Závěr.....	16

Návrh projektu

Vývojové prostředí

Pro vývoj projektu bude využíváno Visual studio Code od Microsoftu

Databáze

Bude využívána MYSQL databáze, databáze bude využita z předchozího projektu, nevylučuji nutnost jejího rozšíření, či její změny

Administrační aplikace

Administrační aplikace bude samostatně fungující jednotka. Využiji její funkčnost v minulém projektu a pokud bude potřeba její úprava, upravím ji podle potřeby. Plus k ní přidám prohlížení již zpracovaných objednávek

Administrační aplikace zahrnuje:

1. Úprava produktů a jejich kategorií
2. Přidávání produktů, kategorií
3. Přidávání přídavek
4. Úprava přídavek
5. Prohlížení objednávek

Objednávací panel

Objednávací panel bude řešený podobně jako v minulém projektu pokladny. Nicméně zde bude využíváno architektury MVC a bude se jednat o webové rozhraní. Již se zde nebude pracovat s RMI, ale bude se využívat Modelů pro komunikaci s databází např. bude Model Produkt, Položka, Přídavek. Servlety(kontrolery) budou zajišťovat požadavky od webového serveru a zajišťovat operace s daty pro daný Model. Jednotlivé produkty budou pravděpodobně vypsané jakožto odkazy, které budou nějak hezky nastýlované a buď zobrazí modální okno s přídávky nebo odkážou na stránku s přídávky k danému produktu. Při potvrzení se budou přidávat do listu, který dostane data metodou POST nebo GET, zvážím při psaní daného Kódu. Na objednávací stránce bude také nějaký div, do kterého se budou zapisovat jednotlivé položky s cenou a u každé položky bude tlačíko/odkaz, který bude sloužit k odebrání položky z objednávky. Dále zde bude tlačítko pro potvrzení objednávky, které odešle data do nějaké fronty v panelu přípravy, a tlačítko k stornování celé objednávky. Při potvrzení objednávky se vytiskne účtenka s danou objednávkou.

Panel přípravy

Budou se zde zobrazovat pouze objednávky, které ještě nejsou připravené a konkrétně názvy položek a číslo objednávky. Pro zakliknutí dokončení objednávky zde využiji nějaké tlačítko, které odešle požadavek a data panelu u výdeje. Po tomto kroku zmizí daná objednávka z přípravy a bude brána jako objednávka, připravená na výdej.

Panel u výdeje

Tento panel bude zobrazovat čísla jak připravovaných objednávek, tak i čísla objednávek připravovaných k výdeji. Bude zde zase nějaké tlačítko u dané objednávky, které označí objednávku jako vydanou a objednávka poté zmizí z panelu výdeje

Popis řešení projektu

Database connection

Třída Db je ve složce Model. Používá se zde Singleton, aby bylo vytvořena vždy jen jedna instance databáze. V konstruktoru se inicializuje ovladač komunikace s databází mysql.jdbc.Driver. A jsou zde metody pro získání spojení s databází, navazuje se pomocí DriverManager.getConnection(url, user, password).

```
You, 12 minutes ago | 1 author (You)
package Model;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

You, 12 minutes ago | 1 author (You)
public class Db {
    private static Db instance;

    public static Db get() throws ClassNotFoundException, SQLException {
        if (instance == null) {
            instance = new Db();
        }
        return instance;
    }
}

You, a week ago + v0.0
private String cs = "jdbc:mysql://localhost:3306/pokladna", user = "root", pass = ""; The value of the fi

private Db() throws ClassNotFoundException, SQLException {
    Class.forName("com.mysql.jdbc.Driver");
}

private final String typDB = "mysql";
private final String serverName = "localhost";
private final int port = 3306;
private final String defaultDB = "pokladna";

public Connection getConnection() throws SQLException {
    return getConnection(typDB, serverName, port, defaultDB, "root", "");
}

public Connection getConnection(String typDB, String serverName, int port, String defaultDB, String user,
    String pass) throws SQLException {
    return DriverManager.getConnection(cs, user, pass);
}
```

Práce s databází v Model.Polozka, Model.Pridavek, Model.Objednavka

Zde nastala oproti minulému projektu změna. Již nevyužívám ke komunikaci s databází rmi serveru, ale využívám webového serveru. Jednotlivé operace jsou prováděny v metodách přímo jednoho Modelu. V metodách pro získání položek, objednávek nebo Přídavků si vytvořím list s daným objektem, dále pomocí try resources si získám spojení s databází, které uložím do objektu třídy Connection: dále si vytvořím statement, který slouží k provádění sql příkazů. Dále si vytvořím objekt Třídy ResultSet, který slouží k uložení výsledků vytažených z databáze pomocí sql dotazu. A v cyklu While projíždím výsledky a z nich vytvářím dané potřebné objekty, jednotlivé hodnoty z databáze se získají metodou get"datový typ" (název sloupce v sql) a následně tento objekt přidám do listu.

Při vkládání do databáze používám transakce(commit) v metodě save (). Nejdříve se vyhodnocuje podmínka, kde, pokud id objektu je menší než 0 => objekt se bude vkládat, pokud je id větší než 0 => objekt se bude upravovat. Nastavím AutoCommit na false a v jednotlivých try reasourcích si vytvořím preparedStatementy se sql dotazem. Pomocí

metod set“datový typ“ (index, hodnota) nastavím parametry hodnot ze sql dotazu. Následně provedu pomocí funkce executeUpdate dané sql příkazy a funkcí commit () potvrdím transakci, pokud se vyskytně nějaká výjimka, transakce se zamkne a zruší. Metoda pro získání všech položek, objednávek, přírůdků je statická, aby byla přístupná odevšud se stejnými daty bez nutnosti vytvoření objektu. Pro získání konkrétního jednoho objektu používám metodu load ().

Získání všech hodnot z databáze

```
public static List<Polozka> getAll() {  
    List<Polozka> polozky = new ArrayList<>();  
    try (Connection conn = Db.get().getConnection();  
        Statement polozkyStmt = conn.createStatement();  
        ResultSet polozkyRs = polozkyStmt.executeQuery(  
            "SELECT polozky.ID, polozky.nazev, polozky.cena, polozky.druh, polozky.isActive FROM polozky WHERE polozky.isActive = 1")) {  
        while (polozkyRs.next()) {  
            System.out.println(polozkyRs.getString("nazev"));  
            polozky.add(new Polozka().setId(polozkyRs.getInt("ID")).setNazev(polozkyRs.getString("nazev"))  
                .setCena(polozkyRs.getDouble("cena")).setDruh(polozkyRs.getString("druh"))  
                .setActive(polozkyRs.getBoolean("isActive")));  
        }  
    } catch (SQLException | ClassNotFoundException e) {  
        e.printStackTrace();  
        polozky = null;  
    }  
}
```

Úprava objektu

```
else {  
    try (Connection conn = Db.get().getConnection()) {  
        conn.setAutoCommit(false);  
        try (PreparedStatement stmt = conn.prepareStatement(  
            "UPDATE polozky SET polozky.isActive = ?, polozky.nazev = ?, polozky.cena = ?, polozky.druh = ? WHERE polozky.ID = ?")) {  
            int pom;  
            if (this.isActive())  
                pom = 1;  
            else  
                pom = 0;  
            stmt.setInt(1, pom);  
            stmt.setString(2, this.getNazev());  
            stmt.setDouble(3, this.getCena());  
            stmt.setString(4, this.getDruh());  
            stmt.setInt(5, this.getId());  
            if (stmt.executeUpdate() != 1) {  
                throw new Exception("Nepodařilo se upravit polozku");  
            }  
        } catch (SQLException e) {  
            e.printStackTrace();  
            conn.rollback();  
            System.out.println(e.getMessage());  
            throw e;  
        }  
        conn.commit();  
        return true;  
    } catch (Exception e) {  
        e.printStackTrace();  
        System.out.println(e.getMessage());  
        return false;  
    }  
}
```

Vkládání do databáze

```
if (this.id <= 0) {
    try (Connection conn = Db.get().getConnection()) {
        conn.setAutoCommit(false);

        try (java.sql.PreparedStatement stmt = conn.prepareStatement(
            "INSERT INTO polozky (nazev, cena, druh) VALUES (?, ?, ?)",
            PreparedStatement.RETURN_GENERATED_KEYS)) {
            stmt.setString(1, this.getNazev());
            stmt.setDouble(2, this.getCena());
            stmt.setObject(3, this.getDruh());

            if (stmt.executeUpdate() != 1) {
                throw new Exception("Nepodaril se zapis polozky");
            }

            ResultSet rs = stmt.getGeneratedKeys();

            if (rs.next()) {
                this.setId(rs.getInt(1));
            }
            rs.close();
        } catch (Exception e) {
            e.printStackTrace();
            conn.rollback();
            throw e;
        }
        conn.commit();
        return true;
    } catch (Exception e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
        return false;
    }
}
```

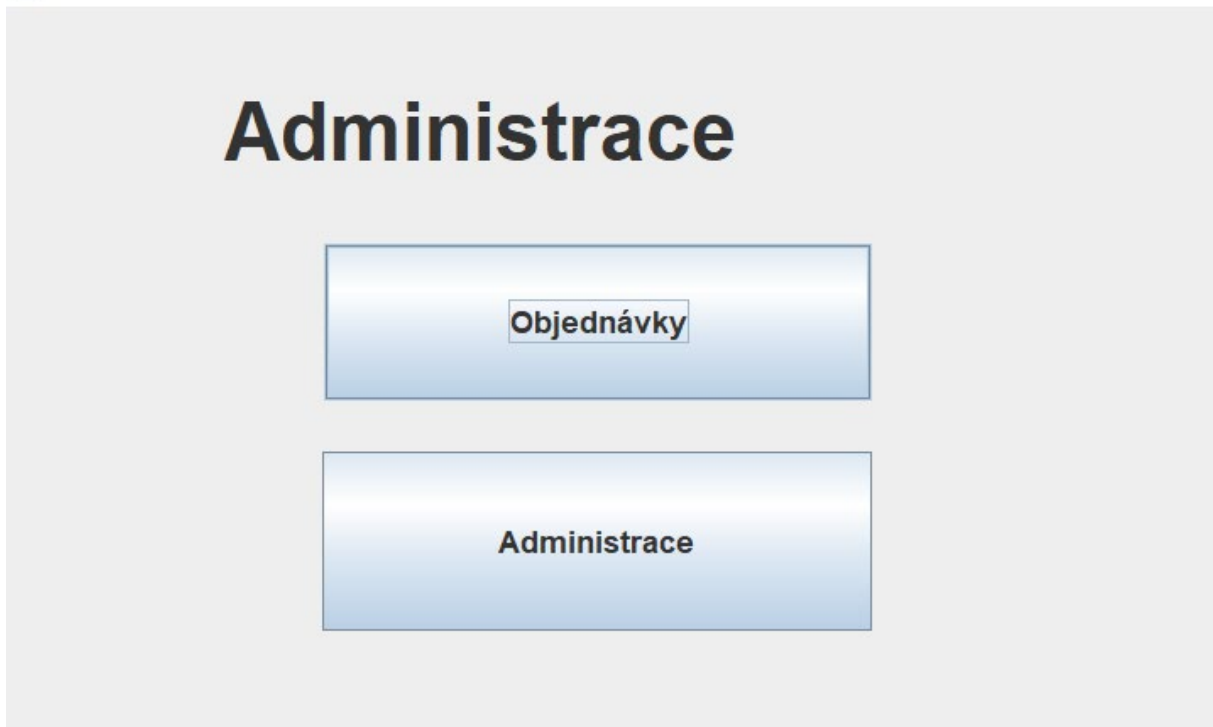
Administrační aplikace

Administrační aplikace se změnila oproti návrhu a minulému projektu jen v tom, že jsem ji adaptoval na MVC architekturu modelů (např. funkce save []). A dále jsem přidal možnost prohlížení objednávek.

Administrace je aplikace s GUI. Při spuštění se otevře okno s rozcestím, kde jsou dva JButtons. Button s nápisem Objednávky zobrazí okno s vypsáním objednávek a z databáze. Druhy s nápisem Administrace otevře okno AdministraceRozcesti, kde jsou Buttons s úpravou položky nebo přídavku a vložení položky nebo přídavku.

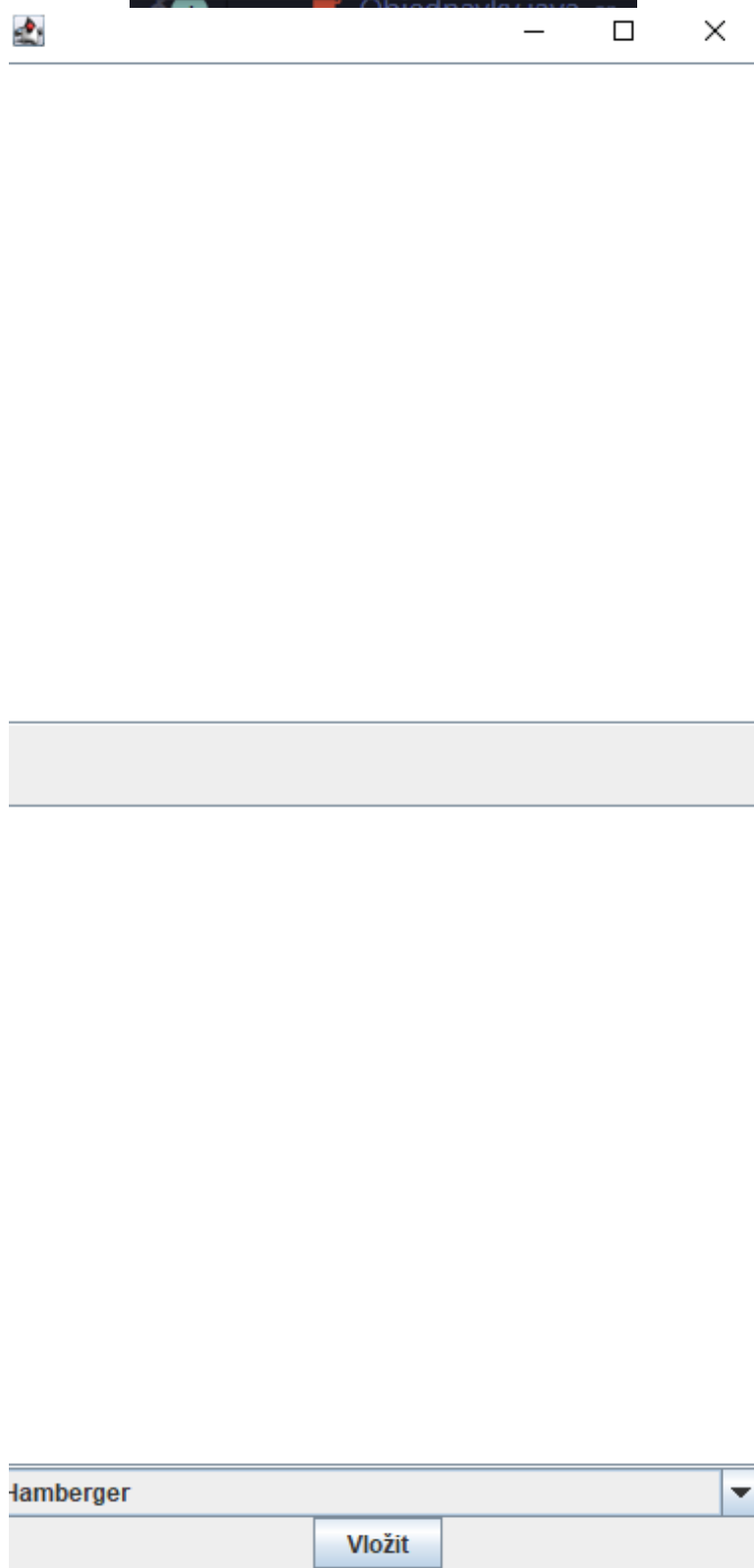
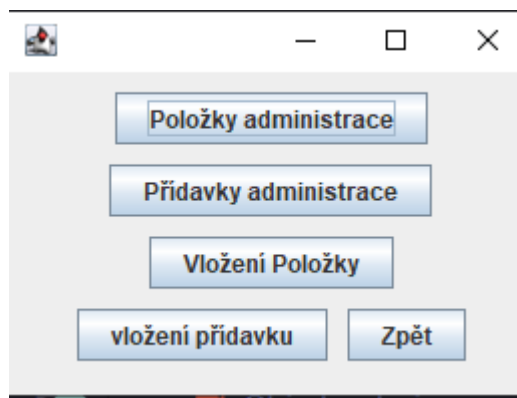


— □ ×



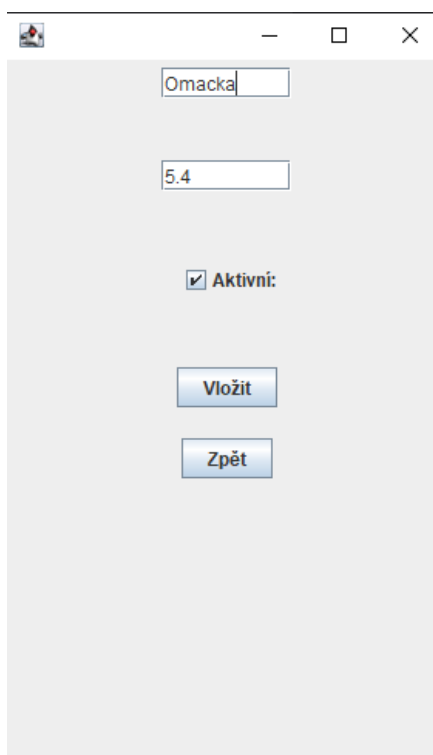
Vložení přídavku či položky

Při kliknutí na button vložení se otevře Frame, ve kterém jsou pole pro zapsání názvu a ceny a u položky ještě je JComboBox s druhem položky. A Button s potvrzením vložení. Při zmáčnutí buttonu potvrzení se pomocí Naming.lookup odkáže na Třidu na serveru vytvoří se objekt a nastaví se mu hodnoty, které se vezmou z daných polí ve Framu. Při úspěšném zapsání se objeví JOptionPane s oznámením že daná položka nebo přídavek se zapsal.



Úprava položky či přídavku

Při kliknutí na úpravu se otevře okno se všemi položkami nebo přídávky. Při kliknutí na položku nebo přídavek se otevře okno, kterému v konstruktoru předávám tu danou položku, na kterou jsem klikl, s formulářem, kde jsou hodnoty té dané položky/ přídavku, který chci upravovat. Při potvrzení úpravy se vytvoří ten daný objekt, nastaví se mu hodnoty a předá se metodě, která má za úkol upravit data v databázi. Pokud vše proběhlo v pořádku, zobrazí se hláška, že položka/ přídavek byla upravena.



Omacka

5.4

☒ Aktivní:

Vložit

Zpět

Prohlížení objednávek

Do historie objednávek se dostává kliknutím na tlačítko „Objednávky“ v oknu AdminUvodniFrame. Po kliknutí na toto tlačítko se zobrazí ObjednavkyFrame, kde je JTabbed pane, kde v každé záložce je objednávka od nejstarší po nejnovější. Zde se pomocí funkce `getAll()`. Dále se pomocí `foreach` cyklu, ve které se prochází načtené objednávky z databáze a dalších vnořených `foreach` cyklů pro výpis položek a přídavků sestavuje objednávka pro prohlížení. Objednávka se sestavuje do `JTextArea`i pomocí `append` a zavolání metod pro získání dat z objednávky (cena, položky, název, čas) a dále se zde upravuje získaný databázový formát data na český. To se dělá pomocí objektů `SimpleDateFormat`, `Date` a `DateFormat`, kde se `SDF` nastaví databázový formát do objektu `Date` se pomocí `SDF.parse(formát)` předá datum z databáze, vytvoří se `DateFormat` s patternem požadovaného formátu data a pomocí funkce `format(Date)` se uloží do řetězce zformátované datum.

Prohlížení objednávek



— □ ×

Č65	Č66	Č67	Č68	Č69	Č70	Č71	Č72	
Č57	Č58	Č59	Č60	Č61	Č62	Č63	Č64	
Č49	Č50	Č51	Č52	Č53	Č54	Č55	Č56	
Č41	Č42	Č43	Č44	Č45	Č46	Č47	Č48	
Č33	Č34	Č35	Č36	Č37	Č38	Č39	Č40	
Č25	Č26	Č27	Č28	Č29	Č30	Č31	Č32	
Č17	Č18	Č19	Č20	Č21	Č22	Č23	Č24	
Č9	Č10	Č11	Č12	Č13	Č14	Č15	Č16	
Č1	Č2	Č3	Č4	Č5	Č6	Č7	Č8	

Objednávka z: 05.04.2021 13:17:11

McDvoři	25.5 Kč	
+Omacka	5.4 Kč	
DvořiBurger se sýrem	55.5 Kč	
DvořiBurger	35.5 Kč	
+Omacka	5.4 Kč	
+Omacka	5.4 Kč	

Celková cena: 163.6 Kč

Zpět

Objednávací panel

Objednávací panel je ve formě webové stránky. Využívá architekturu MVC, kde se pomocí servletu posílají do něho data. Jedná se o .jsp stránku, která pomocí jsp tagů vypisuje data, umožňuje průchod Listem atd. Jsou zde vypsány položky s tlačítkem, které předá do url adresy parametr sId dané položky. Dále je zde aktuální objednávka, která je modifikovatelná. Při kliknutí na tlačítko u položky se zobrazí přídavky. Při kliknutí na tlačítko pod přídavkem, přídavek se přidá k položce, kterou jsme si vybrali. Následně se vše potvrdí tlačítkem ok, a to nás přesměruje zpět na objednávací panel, kde vidíme naši objednávku. U objednávky je tlačítko na dokončení objednávky, která objednávku zapíše do databáze a pošle objednávku na přípravu.

Dvořina				
DvořičBurger	DvořičBurger se sýrem	McDvořič	Pálivý Dvořič	Shompův mls
Cena: 35.5 Kč	Cena: 55.5 Kč	Cena: 25.5 Kč	Cena: 40.5 Kč	Cena: 30.0 Kč
Hamberger	Hamberger	Hamberger	Hamberger	Wrap
<input type="button" value="Přidat"/>	<input type="button" value="Přidat"/>	<input type="button" value="Přidat"/>	<input type="button" value="Přidat"/>	<input type="button" value="Přidat"/>

Objednávka

Celková cena: 0.0 Kč

Dvořina		
Omačka	Tatarka	Hořčice
Cena: 5.4 Kč	Cena: 10.5 Kč	Cena: 5.5 Kč
<input type="button" value="Přidat"/>	<input type="button" value="Přidat"/>	<input type="button" value="Přidat"/>

PolozkaServlet.java

Tento servlet se stará o zobrazování přídavek, položek, zapisování do databáze. Pomocí parametrů z URL adresy jednotlivé podmínky provádí různé věci. Pokud není žádný parametr s ID položky, tak se zobrazí objednávací panel s objednávkou a položkami. Nastaví se atribut s položkami a odešle se pohledu objednavka.jsp. Pokud ID položky není null a zároveň ještě není nastaveno ID přídavek, tak se vytvoří daná položka a přidá se do listu objednávky. Následně se zobrazí pohled objednavka.pridavek.jsp, kde jsou vypsány přídavky. Po kliknutí na tlačítko nějakého přídavek se nastaví parametr Id přídavek a přidá se k poslední položce v objednávce. Dále se zde řeší odstraňování položky z objednávky. Z URL adresy se vezme parametr odstran, ve kterém je index dané položky a pomocí remove se odstraní z listu. Také lze odstraňovat jednotlivé přídavky, zde slouží parametr odPolozky, kde je index té položky a index přídavek v listu přídavek té dané položky. Po kliknutí na odstran přídavek se pomocí remove odebere daný přídavek od položky. Dále lze k jednotlivé položce přidávat přídavek. Zde je parametr kPolozce, který když je v URL adrese, tak se zobrazí přídavky a označuje index položky v objednávce. Pak se pomocí add přidá přídavek k položce. Nakonec je zde podmínka, která obsluhuje dokončení objednávky její vytisknutí a zapsání do databáze.

```

if (req.getParameter("ID_polozka") == null) {
    req.setAttribute("polozkyList", Polozka.getAll());
    if (Objednavka.getObjednavka().isEmpty()) {
        req.setAttribute("prazdna", true);
    } else {
        req.setAttribute("objednavka", Objednavka.getObjednavka());
        req.setAttribute("prazdna", false);
    }
    getServletContext().getRequestDispatcher("/objednavka.jsp").forward(req, resp);
    return;
}

if (req.getParameter("ID_polozka") != null && req.getParameter("ID_pridavek") == null) {

    int id = Integer.parseInt(req.getParameter("ID_polozka"));

    Polozka p = new Polozka(id);

    Objednavka.getObjednavka().add(p);

    req.setAttribute("pridavkyList", Pridavek.getAll());
    req.setAttribute("ID_polozka", req.getParameter("ID_polozka"));

    getServletContext().getRequestDispatcher("/objednavka.pridavek.jsp").forward(req, resp);

    return;
}

if (req.getParameter("ID_polozka") != null && req.getParameter("ID_pridavek") != null) {
    req.setAttribute("pridavkyList", Pridavek.getAll());
    req.setAttribute("ID_polozka", req.getParameter("ID_polozka"));

    getServletContext().getRequestDispatcher("/objednavka.pridavek.jsp").forward(req, resp);
    int id = Integer.parseInt(req.getParameter("ID_pridavek"));
    Pridavek pridavek = new Pridavek(id);

    Objednavka.getObjednavka().getLast().getPridavky().add(pridavek);
    return;
}

```

You, an hour ago | 1 author (You)

```
@WebServlet(name = "index", urlPatterns = { "/objednavka" })
```

```
public class PolozkaServlet extends HttpServlet {
    private Tiskarna t = Tiskarna.getITiskarna();
```

```
@Override
```

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
```

```
    req.setAttribute("cena", Objednavka.sum());
    if (req.getParameter("dokoncit") != null) {
        Objednavka o = new Objednavka();
        o.setPolozky(Objednavka.getObjednavka());
        o.setCena(Objednavka.sum());
```

```
        java.util.Date dt = new java.util.Date();
```

```
        java.text.SimpleDateFormat sdf = new java.text.SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
```

```
        String currentTime = sdf.format(dt);
```

```
        o.setCasObjednavky(currentTime);
        t.Tiskni(o);
        o.save();
        Objednavka.getObjednavka().clear();
        resp.sendRedirect("/dvorina/index.jsp");
        return;
    }
```

```
    if (req.getParameter("odstran") != null) {
        Objednavka.getObjednavka().remove(Integer.parseInt(req.getParameter("odstran")));
```

```
        req.setAttribute("cena", Objednavka.sum());
    }
```

```
    if (req.getParameter("odPolozky") != null && req.getParameter("pridavek") != null) {
        Objednavka.getObjednavka().get(Integer.parseInt(req.getParameter("odPolozky"))).getPridavky()
            .remove(Integer.parseInt(req.getParameter("pridavek")));
        req.setAttribute("cena", Objednavka.sum());
    }
```

```
    if (req.getParameter("kPolozce") != null) {
        req.setAttribute("pridavkyList", Pridavek.getAll());
        req.setAttribute("kPolozce", req.getParameter("kPolozce"));
```

```
        getServletContext().getRequestDispatcher("/objednavka.pridavek.pridat.jsp").forward(req, resp);
    }
```

```
    if (req.getParameter("kPolozce") != null && req.getParameter("ID_pridavek") != null) {
        Pridavek p = new Pridavek(Integer.parseInt(req.getParameter("ID_pridavek")));
        Objednavka.getObjednavka().get(Integer.parseInt(req.getParameter("kPolozce"))).getPridavky().add(p);
        req.setAttribute("cena", Objednavka.sum());
    }
```

VydejServlet

Tento servlet zobrazuje pohled vydej.panel.jsp, kde jsou zobrazené objednávky připravené k výdeji a připravované objednávky.

PripravaServlet

Tento servlet zobrazuje pohled priprava.panel.jsp, kde jsou zobrazené připravované objednávky.

Panel přípravy a panel výdeje

K získávání a předávání dat mezi panely jsem využil websocket server, který společně s Javascriptem zpracovává požadavky a odesílá je všem klientům, které jsou k němu přihlášeny. K definování websocket serveru se používá anotace @ServerEndpoint, do které se dá odkaz, pod kterým se k němu chceme připojit. Websocket má 4 metody, které jsou nadepsané anotacemi @OnConnect = provádí se, když se poprvé připojí klient, @OnMessage = provádí se, když přichází zpráva, @OnClose = provádí se při odpojení klienta a @OnError = provádí se, když nastane nějaká chyba. K zpracování požadavků mám definovanou samostatnou třídu, která rozesílá data klientům a říká co se má stát. v websocket serveru v metodě OnMessage zpracovávám zprávu v podobě JSON souboru, kde se získává akce, která se má provést = add => přidá se objednávka k přípravě, remove => odebere se z přípravy a přidá se k výdeji, vyzvednuta => odebere objednávku z objednávek připravených k výdeji.

Výdej

V přípravě

Příprava

Objednávka číslo: 1
cena:40.5 Kč

K vyzvednutí

Objednávka číslo: 0 Objednávka číslo: 1
cena:25.5 Kč cena:40.5 Kč

Vydej objednávku

Vydej objednávku

Objednávka číslo: 0
cena: 25.5 Kč

Vyzvednuta

ObjednavkaSessionHandler

Tato třída zpracovává a odesílá požadavky. Jsou zde uloženy připojené sessiony, objednávky k výdeji a objednávky v přípravě. Pomocí session.getBasicRemote().sendText() se odesílají data jednotlivým sessionám. Dále zde nastavuji action do Json souboru, který potom zpracovává zase javascript.

Websocket.js

K websocketu se připojuje vytvořením objektu Websocket, který má v konstruktoru adresu serveru websocketu. Nastavuji mu funkci onMessage, která se provádí, pokud přijde nějaká zpráva. Pokud event. Data action je add => zavolá se funkce která do stránky vypíše předávanou objednávku. Remove => při zavolání remove se odebere objednávka s daným ID ze stránky a v ObjednavkaSessionHandle se pošle se na výdej. Vydej => vypíše se objednávka, která je určena k výdeji. Vyzvednuta => odebere se z výdeje a je vydaná. Při kliknutí na tlačítka u jednotlivých objednávek se zavolají funkce, které nastaví action a pomocí ws.send je pošlou na websocket server, který si je zpracuje a pošle zpět odpověď.

```
You, 2 hours ago | 1 author (You)
@ServerEndpoint("/endpoint")
public class WebSocketServer {
    private Session session; The value of the field WebSocketServer.session is not used
    private static Set<WebSocketServer> endpoints = new CopyOnWriteArraySet<>();

    @Inject
    private ObjednavkaSessionHandler sessionHandler = ObjednavkaSessionHandler.getInstance();

    @OnOpen
    public void onOpen(Session session) throws IOException {
        this.session = session;
        endpoints.add(this);
        System.out.println("pripojeno" + session.getId());
        sessionHandler.addSession(session);
    }

    @OnClose
    public void onClose(Session session) {
        sessionHandler.removeSession(session);
    }

    @OnMessage
    public void objednavkaReceiver(String message, Session session) throws IOException, EncodeException {
        try (JsonReader reader = Json.createReader(new StringReader(message))) {
            JsonObject jsonMessage = reader.readObject();

            if ("add".equals(jsonMessage.getString("action"))) {
                Objednavka o = Objednavka.getAll().getLast();
                sessionHandler.addObjednavka(o);
            }

            if ("remove".equals(jsonMessage.getString("action"))) {
                int id = (int) jsonMessage.getInt("id");
                sessionHandler.vydejObjednavka(id);
            }

            if ("vyzvednuta".equals(jsonMessage.getString("action"))) {
                int id = (int) jsonMessage.getInt("id");
                sessionHandler.vyzvednutaObjednavka(id);
            }
        }
        System.out.println("prijato");
    }

    @OnError
    public void onError(Throwable t) {
        System.out.println("onError::" + t.getMessage());
    }
}
```

```

20 public class ObjednavkaSessionHandler {
21     private int objednavkyId = 0;
22     private final Set<Session> sessions = new HashSet<>();
23     private final Set<Objednavka> Objednavka = new HashSet<>();
24     private final Set<Objednavka> ObjednavkaVydej = new HashSet<>();
25     private static ObjednavkaSessionHandler instance;
26
27     public static ObjednavkaSessionHandler getInstance() {
28         if (instance == null) {
29             instance = new ObjednavkaSessionHandler();
30         }
31         return instance;
32     }
33
34     public Set<Objednavka> getObjednavkaVydej() {
35         return ObjednavkaVydej;
36     }
37
38     public void addSession(Session session) {
39         sessions.add(session);
40
41         for (Objednavka objednavka2 : Objednavka) {
42             JsonObject addMessage = createAddMessage(objednavka2);
43             sendToSession(session, addMessage);
44         }
45         for (Objednavka objednavka3 : ObjednavkaVydej) {
46             JsonObject addMessage = createAddMessage(objednavka3);
47             sendToSession(session, addMessage);
48         }
49     }
50
51     public void removeSession(Session session) {
52         sessions.remove(session);
53     }
54
55     public List<Objednavka> getObjednavky() {
56         return new ArrayList<>(Objednavka);
57     }
58
59     public void addObjednavka(Objednavka objednavka) {
60         objednavka.setId(objednavkyId);
61         Objednavka.add(objednavka);
62         objednavkyId++;
63
64         JsonObject addMessage = createAddMessage(objednavka);
65         sendToAllConnectedSessions(addMessage);
66     }
67
68     public void vydejObjednavka(int id) {
69         Objednavka o = getObjednavkaById(id);
70         if (o != null) {
71             Objednavka.remove(o);
72             ObjednavkaVydej.add(o);
73         }
74     }
75
76     JsonProvider provider = JsonProvider.provider();
77     JsonObject removeMessage = provider.createObjectBuilder().add("action", "remove").add("id", id).build();
78     sendToAllConnectedSessions(removeMessage);
79     posliNaVydej(o);
80 }

```

```

var ws;

ws = new WebSocket("ws://localhost:8080/dvorinald/endpoint");

ws.onerror = (event) => {
    console.log(event.data);
}
ws.onmessage = onMessage;

function onMessage(event) {
    let objednavka = JSON.parse(event.data);

    if (objednavka.action == "add") {
        printObjednavkaPriprava(objednavka);
    }
    if (objednavka.action == "remove") {
        document.getElementById(objednavka.id).remove();
    }
    if (objednavka.action == "vydej") {
        printObjednavkaVydej(objednavka);
    }
    if (objednavka.action == "vyzvednuta") {
        document.getElementById(objednavka.id).remove();
    }
}

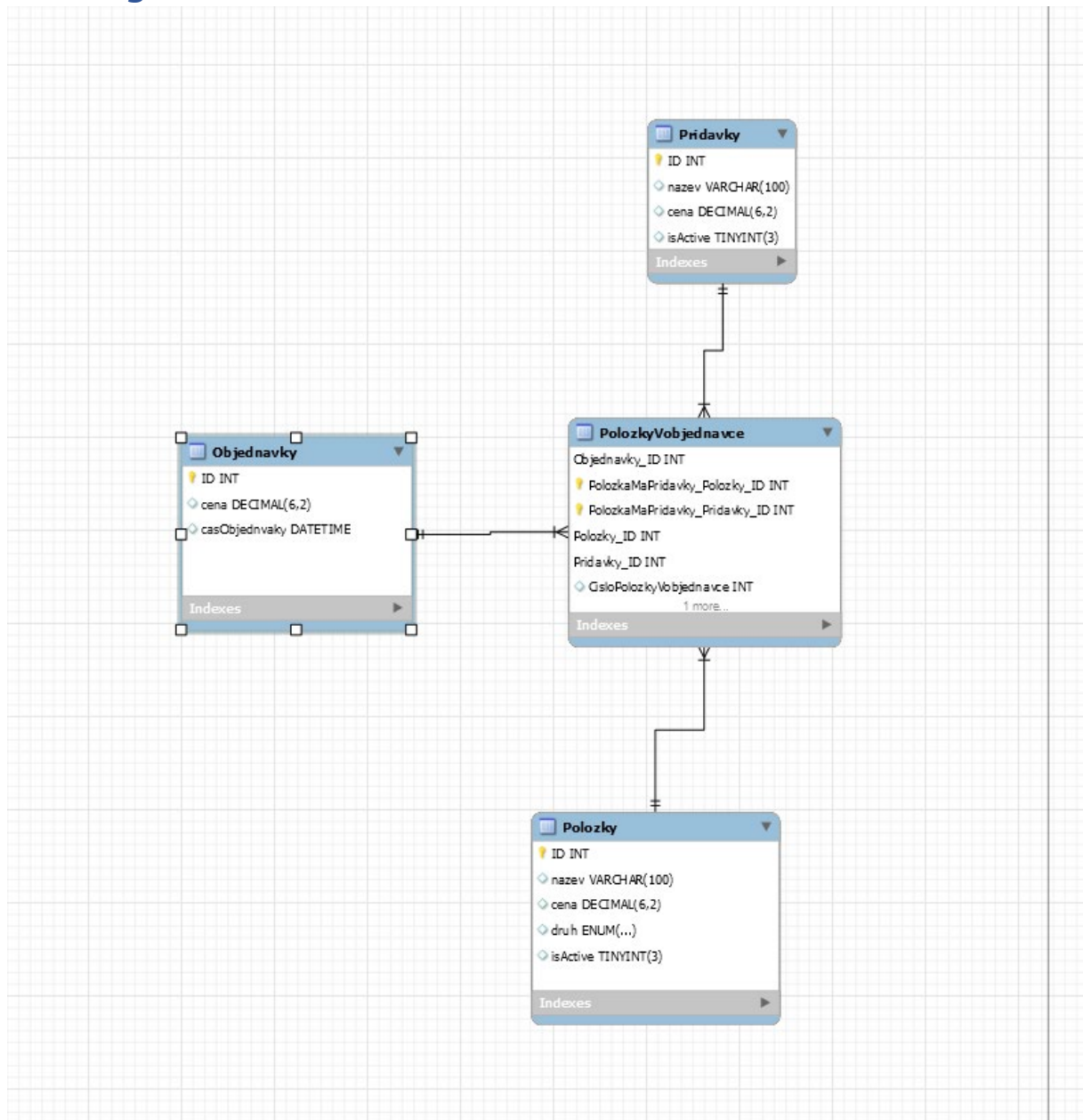
function addObjednavka() {
    let objednavkaAction = {
        action: "add"
    };
    ws.send(JSON.stringify(objednavkaAction));
}

function removeObjednavka(element) {
    let id = element;
    let objednavkaAction = {
        action: "remove",
        id: id
    };
    ws.send(JSON.stringify(objednavkaAction));
}

function VydejObjednavka(element) {
    let id = element;
    let objednavkaAction = {
        action: "vyzvednuta",
        id: id
    };
    ws.send(JSON.stringify(objednavkaAction));
}

```


E-R diagram



Závěr

Projekt mi dal zase nové věci do mého programátorského repertoáru. S pár věcmi jsem měl problém, než jsem je začal chápat, ale vyřešily se. Projekt mě bavil a jsem spokojený s výsledkem.