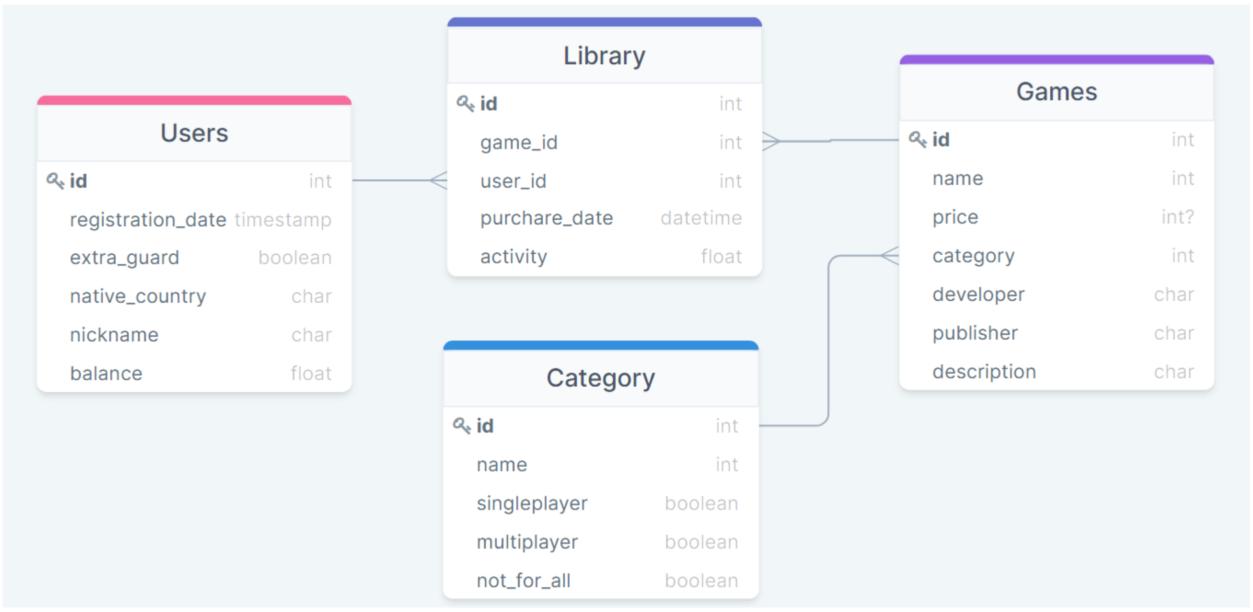
Documentation for Servlet Web Application.

For this project, I developed a web application with CRUD operations on a database using Java Servlet technologies. The architectural design of this application is Model-View-Controller.

First, we need to make a database with a different relationship.



Picture 1. Database schema.

Relationships:

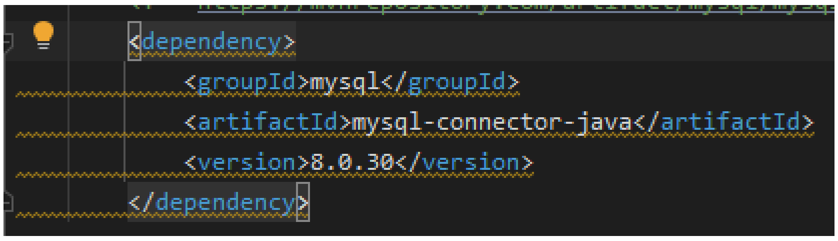
One to many:

- Users – Library.
- Games – Library.
- Category – Games.

Many to many:

- Users – Games.

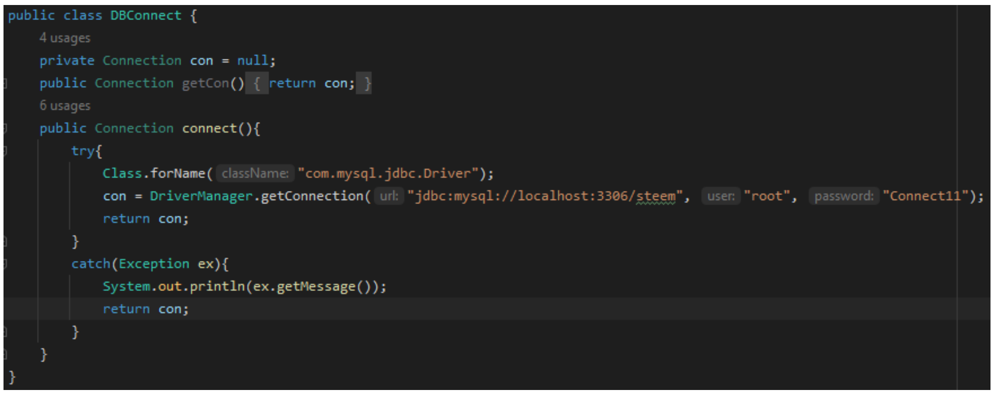
Then, we need to connect to the MySQL Server. In the configuration file, we need to add the MySQL dependency.



```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.30</version>
</dependency>
```

Picture 2. Add dependency.

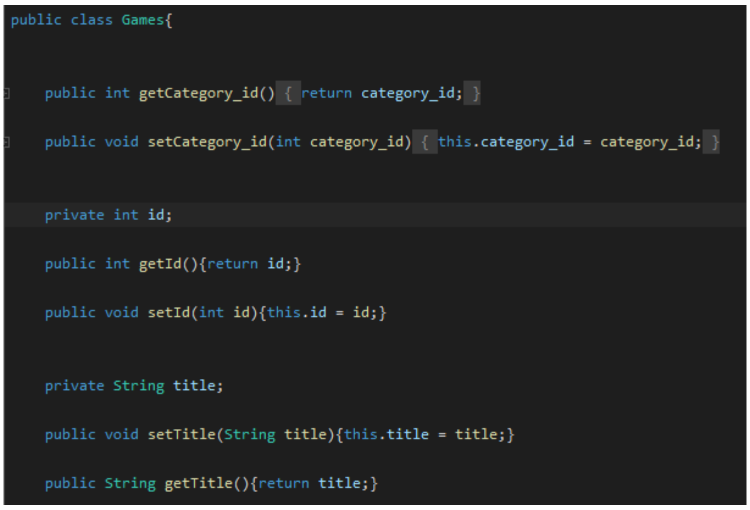
Next, we need to establish a connection between the SQL server and our project using the java.sql class Driver Manager.



```
public class DBConnect {
    4 usages
    private Connection con = null;
    public Connection getCon() { return con; }
    6 usages
    public Connection connect(){
        try{
            Class.forName( className: "com.mysql.jdbc.Driver");
            con = DriverManager.getConnection( url: "jdbc:mysql://localhost:3306/steem",  user: "root",  password: "Connect11");
            return con;
        }
        catch(Exception ex){
            System.out.println(ex.getMessage());
            return con;
        }
    }
}
```

Picture 3. Connection with database.

As a Model part of this project, I use Data Access Object classes and classes with the same SQL fields.



```
public class Games{


    public int getCategory_id() { return category_id; }

    public void setCategory_id(int category_id) { this.category_id = category_id; }


    private int id;

    public int getId(){return id;}

    public void setId(int id){this.id = id;}


    private String title;

    public void setTitle(String title){this.title = title;}

    public String getTitle(){return title;}
```

Picture 4. Class with the same fields.

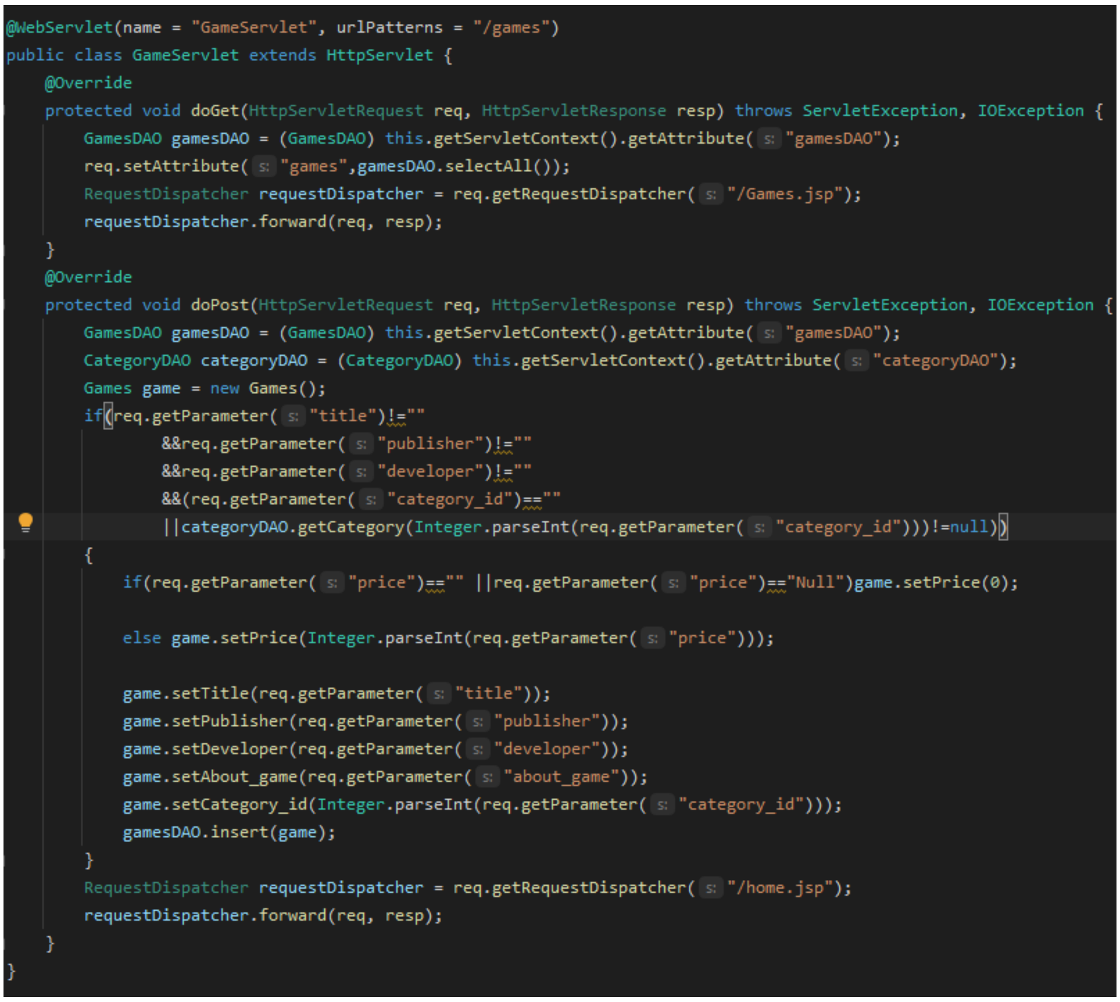
DAO classes consist of creating, updating, deleting, and reading methods.

Example of delete operation:

```
public void delete(int id){
    PreparedStatement state;
    try{
        state = connect.prepareStatement("delete from category where id=?");
        state.setInt(1, id);
        state.executeUpdate();
    }
    catch (SQLException ex){
        Logger.getLogger(CategoryDAO.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

We need to create PreparedStatement for SQL query and paste id number. All methods consist of prepared SQL queries and input parameters. PreparedStatement

The controller part of this project uses Java Servlets technology. They are making Rest API responses.



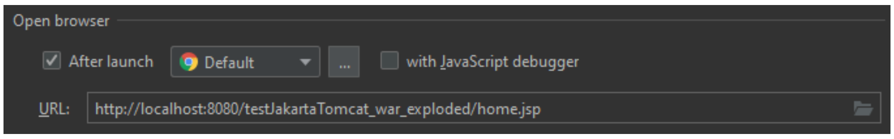
```
@WebServlet(name = "GameServlet", urlPatterns = "/games")
public class GameServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        GamesDAO gamesDAO = (GamesDAO) this.getServletContext().getAttribute( s: "gamesDAO");
        req.setAttribute( s: "games",gamesDAO.selectAll());
        RequestDispatcher requestDispatcher = req.getRequestDispatcher( s: "/Games.jsp");
        requestDispatcher.forward(req, resp);
    }
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        GamesDAO gamesDAO = (GamesDAO) this.getServletContext().getAttribute( s: "gamesDAO");
        CategoryDAO categoryDAO = (CategoryDAO) this.getServletContext().getAttribute( s: "categoryDAO");
        Games game = new Games();
        if(req.getParameter( s: "title")!=""
                &&req.getParameter( s: "publisher")!=""
                &&req.getParameter( s: "developer")!=""
                &&(req.getParameter( s: "category_id")==""
                ||categoryDAO.getCategory(Integer.parseInt(req.getParameter( s: "category_id")))!=null))
        {
            if(req.getParameter( s: "price")=="" ||req.getParameter( s: "price")=="Null")game.setPrice(0);

            else game.setPrice(Integer.parseInt(req.getParameter( s: "price")));

            game.setTitle(req.getParameter( s: "title"));
            game.setPublisher(req.getParameter( s: "publisher"));
            game.setDeveloper(req.getParameter( s: "developer"));
            game.setAbout_game(req.getParameter( s: "about_game"));
            game.setCategory_id(Integer.parseInt(req.getParameter( s: "category_id")));
            gamesDAO.insert(game);
        }
        RequestDispatcher requestDispatcher = req.getRequestDispatcher( s: "/home.jsp");
        requestDispatcher.forward(req, resp);
    }
}
```

Picture 5. GameServlet.

Each table has a 3 servlets:

- (TableName)Servlet has two methods: doGet, response data to the user, and doPost, which collects data from the user (in insert case).
- (TableName)Redact also has two methods: doGet to update a record in this table and doPost to save the updated record.
- (TableName)Delete only has a doPost method for deleting a record from a table.

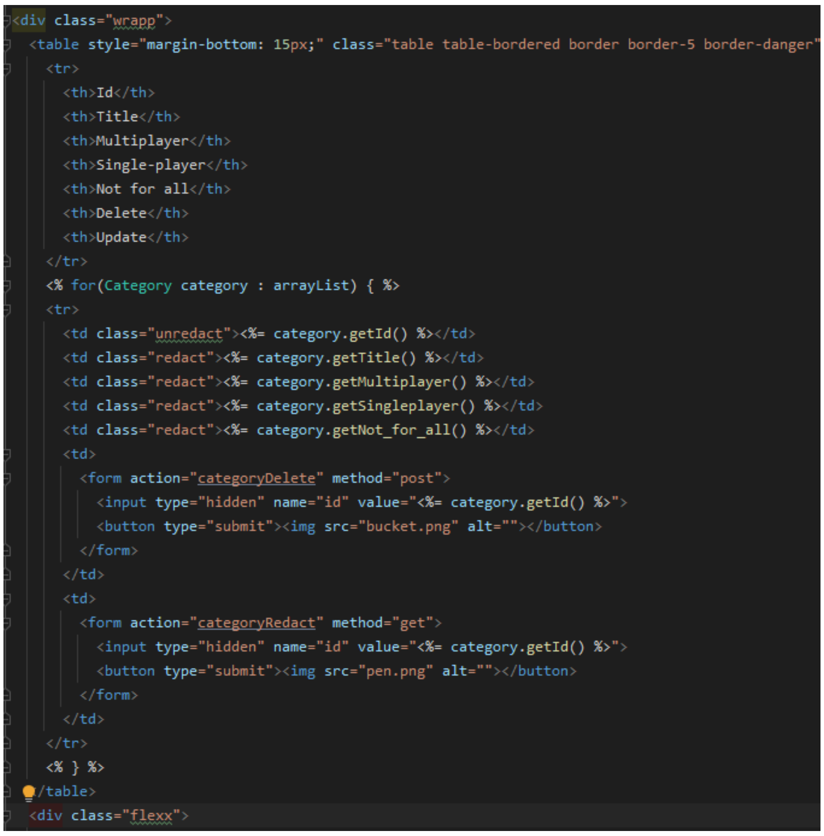
The view part uses Java Servlet Pages technology. To specify the start directory, we need to update the Tomcat Server Configuration.



```
Open browser
  ☑ After launch   Default   ...   ☐ with JavaScript debugger
  URL: http://localhost:8080/testJakartaTomcat_war_exploded/home.jsp
```

Picture 6. Tomcat Server Configuration.

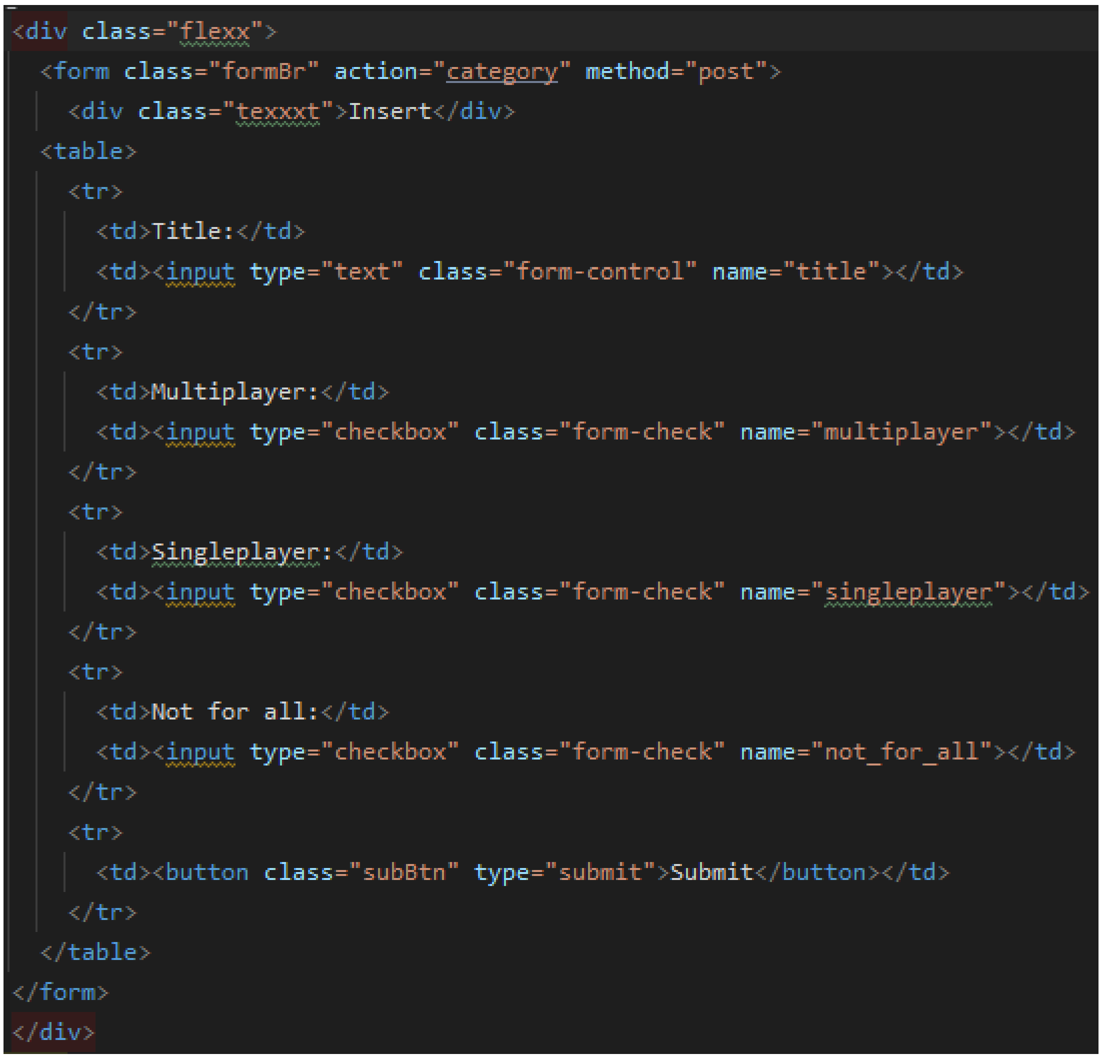
Next we get information from method of DAO classes and using java servlet pages print it with cycle.



```
<div class="wrapp">
  <table style="margin-bottom: 15px;" class="table table-bordered border border-5 border-danger"
    <tr>
      <th>Id</th>
      <th>Title</th>
      <th>Multiplayer</th>
      <th>Single-player</th>
      <th>Not for all</th>
      <th>Delete</th>
      <th>Update</th>
    </tr>
    <% for(Category category : arrayList) { %>
    <tr>
      <td class="unredact"><%= category.getId() %></td>
      <td class="redact"><%= category.getTitle() %></td>
      <td class="redact"><%= category.getMultiplayer() %></td>
      <td class="redact"><%= category.getSingleplayer() %></td>
      <td class="redact"><%= category.getNot_for_all() %></td>
      <td>
        <form action="categoryDelete" method="post">
          <input type="hidden" name="id" value="<%= category.getId() %>">
          <button type="submit"><img src="bucket.png" alt=""></button>
        </form>
      </td>
      <td>
        <form action="categoryRedact" method="get">
          <input type="hidden" name="id" value="<%= category.getId() %>">
          <button type="submit"><img src="pen.png" alt=""></button>
        </form>
      </td>
    </tr>
    <% } %>
  </table>
  <div class="flexx">
```

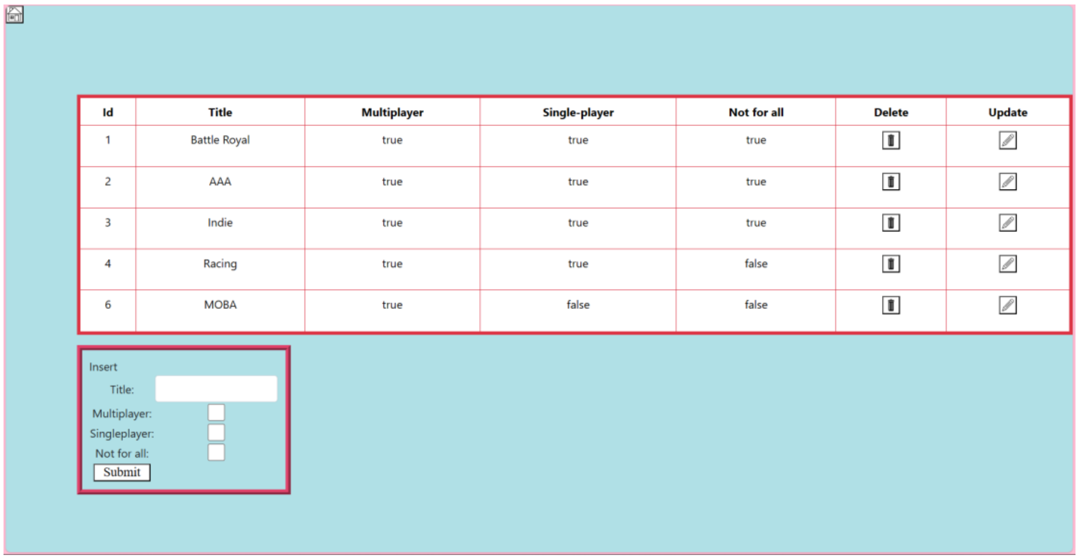
Picture 7. JSP table output part.

Post method of Rest API interact with java servlet sending data to servlets.



```
<div class="flexx">
  <form class="formBr" action="category" method="post">
    <div class="texxxt">Insert</div>
  <table>
    <tr>
      <td>Title:</td>
      <td><input type="text" class="form-control" name="title"></td>
    </tr>
    <tr>
      <td>Multiplayer:</td>
      <td><input type="checkbox" class="form-check" name="multiplayer"></td>
    </tr>
    <tr>
      <td>Singleplayer:</td>
      <td><input type="checkbox" class="form-check" name="singleplayer"></td>
    </tr>
    <tr>
      <td>Not for all:</td>
      <td><input type="checkbox" class="form-check" name="not_for_all"></td>
    </tr>
    <tr>
      <td><button class="subBtn" type="submit">Submit</button></td>
    </tr>
  </table>
</form>
</div>
```

Picture 8. Input form for create new record to the database.



Id	Title	Multiplayer	Single-player	Not for all	Delete	Update
1	Battle Royal	true	true	true	🗑	✎
2	AAA	true	true	true	🗑	✎
3	Indie	true	true	true	🗑	✎
4	Racing	true	true	false	🗑	✎
6	MOBA	true	false	false	🗑	✎

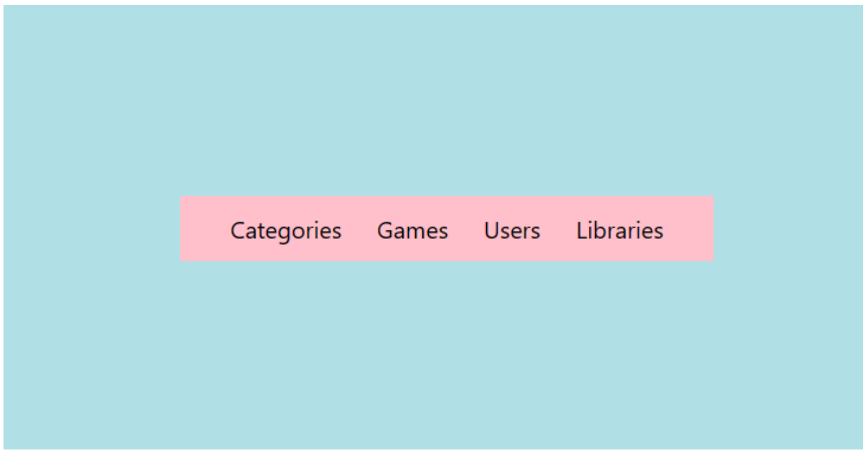
Insert

Title:	
Multiplayer:	☐
Singleplayer:	☐
Not for all:	☐

Picture 9. Games page view.

In home page we have only references to all table view links.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Home</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
<div class="wrap">
    <div class="content">
        <a class="tikal"
href="/testJakartaTomcat_war_exploded/category">Categories</a>
        <a class="tikal" href="/testJakartaTomcat_war_exploded/games">Games</a>
        <a class="tikal" href="/testJakartaTomcat_war_exploded/users">Users</a>
        <a class="tikal" href="/testJakartaTomcat_war_exploded/library">Libraries</a>
    </div>
</div>
</body>
</html>
```



Picture 10. Home page.