

Этап 2

Реализовать обновление списка сущностей (задач, пьес и т.п.).

Переход на страницу со списком должен осуществляться сразу после логина (опционально и после регистрации).

На первом этапе сервлеты логина и регистрации завершались передачей управления на jsp-страницу. На данном этапе нужно отредактировать сервлеты, передавая управление на сервлет обновления списка сущностей. А этот сервлет уже будет форвардить запрос на jsp-страницу.

На данной странице в проекте 1:

- хедер;
- область элементов управления (ссылки или кнопки) для перехода между разделами Today, Tomorrow, Someday, Fixed, Recycle Bin;
- активный раздел списка, который содержит имя раздела и таблицу со столбцами:
 - чекбокс;
 - задачу (краткое содержимое);
 - для разделов Someday, Fixed, Recycle Bin - дату выполнения задачи;
 - элементы управления файлом;
- контекстно-зависимая область элементов управления (ссылки или кнопки) для операций над задачами активного раздела;
- футер.

Пример 1 внутренней области (без хедера и футера).

[Tomorrow 02.02](#) [Someday](#) [Fixed](#) [Recycle Bin](#)

Today 01.02




<input type="checkbox"/>	Task	File
<input type="checkbox"/>	test stage 1	<input checked="" type="checkbox"/> webProject1.war
<input checked="" type="checkbox"/>	sport 15:00	<div>Upload file...</div>
...		

[Add task](#) [Fix](#) [Remove](#)



Пример 2 внутренней области.

[Fixed](#) [Recycle Bin](#)



Today 01.02

	Task	File
	test stage 1 of project	<input checked="" type="checkbox"/> <u>webProject1.war</u>
	sport 15:00	<div>Upload file...</div>
...		

Tomorrow 02.02

	Task	File
	...	<div>Upload file...</div>
...		

Someday

	Task	Date	File
	...		<div>Upload file...</div>
...			

[Add task](#) [Fix](#) [Remove](#)

Возникает вопрос о добавлении методов в интерфейс этапа 1 или создать новый интерфейс.

```
//проект 1
//один интерфейс
public interface ISourceDAO {
    User getUser(String login, String password) throws DAOException;
    User addAndGetUser(String login, String password) throws DAOException;
    List<Task> getTasks(...) throws DAOException;
}

//раздельные интерфейсы
public interface IUserDAO {
    User getUser(String login, String password) throws DAOException;
    User addAndGetUser(String login, String password) throws DAOException;
}
public interface ITaskDAO {
    List<Task> getTasks(...) throws DAOException;
}
```

Теоретически аккаунты и задачи могут храниться в разных источниках данных. Следовательно, нужно использовать два интерфейса.

Есть еще один тонкий момент. Сервлет обновления разделов списка задач должен начинаться проверкой условия отправки запроса залогиненным пользователем.

Антипаттерны

```
public class MainController extends AbstractGetHttpServlet {
    protected void performTask(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        ...
        try {
            TaskImplDB taskDAO = new TaskImplDB();
            //контроллер знает об источнике :(
            ...
        } catch (DAOException e) {
            ...
        }
    }
}
```

```
public class TaskImplDB implements ITaskDAO {
    public List<Task> getTasks(...) throws DAOException {
        ...
        List<Task> tasks = new ArrayList<Task>();
        try {
            cn = DBConnection.getConnection(); //возможен DAOException
            ...
            rs = psSel.executeQuery();          //возможен SQLException
            while (rs.next()) {
                int idTask = rs.getInt(iId);    //возможен SQLException на
любой итерации

                ...
                tasks.add(new Task(idTask,description,date));
            }
            return tasks;
        } catch (SQLException e) {
            LOGGER.log(Level.SEVERE, e.toString(), e);
            return tasks;
            //часть коллекции, а внешне кажется, что полная
            //или return new ArrayList<Task>();
            //пустая коллекция - возможный результат как основного,
            //так и альтернативного сценария
            //или return null;
            //проблема возникновения NullPointerException
        } finally {
            DBConnection.closeResources(cn,psSel,rs);
        }
    }
}
```

На данном этапе главное – наличие РАБОТАЮЩЕГО функционала.

Usability – дело вторичное, так как это учебный проект. Останется время, можно заниматься всякими дизайнами и дружественным интерфейсом.

Этап 3

Реализовать операции для активных элементов списка сущностей.

Каждая операция связана с обновлением источника данных. Следовательно, данные

форм для этих операций должны отправляться на сервер методом POST. Контроллер операции должен завершаться редиректом, если повторная отправка этого же запроса со стороны интерфейса браузера (например, обновить текущую страницу или вернуться на предыдущую) приведет к появлению новых данных или изменению состояния данных. Иначе контроллер можно завершать форвардом запроса на jsp-страницу этапа 2.

Пример 1. Контроллер для добавления задачи в основном сценарии должен завершаться редиректом на jsp-страницу этапа 2 (иначе можно добавить в источник данных одну и ту же задачу два и более раза).

Пример 2. Контроллер для удаления задач можно завершить форвардом (дважды удалить одну и ту же задачу из источника данных невозможно).

Если сущности в БД логически разделяются на непересекающиеся группы (например, задачи привязаны к индивидуальному аккаунту), то в имплементации не нужно синхронизировать обновление источника данных. В противном случае каждое обновление источника данных должно быть реализовано в минимально необходимом синхронизированном блоке (например, если допустить коллективные аккаунты в проекте 1 или заказ билетов в проекте 2).

Каждый сервлет этапа 3, как и сервлет этапа 2, должен начинаться проверкой условия отправки запроса залогиненным пользователем.

Чтобы не дублировать код проверки условия, лучше всего воспользоваться механизмом фильтрации запросов (см. И. Блинов, издание 2007 г., стр. 516-519).

```
public class LoginFilter implements Filter {
    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {
        HttpServletRequest httpRequest = (HttpServletRequest) request;
        HttpSession session = httpRequest.getSession();
        User user = (User) session.getAttribute(ConstantsJSP.USER_KEY);
        if (user == null) {
            session.invalidate();
            HttpServletResponse httpResponse =
                (HttpServletResponse) response;
            httpResponse.sendRedirect(ConstantsJSP.PAGE_LOGIN);
            return;
        }
        chain.doFilter(request, response);
    }
}
```

Чтобы данный фильтр действовал на все URL вида

`http://localhost:8080/project/operation/operation_name`,

его надо зарегистрировать в web.xml следующим образом.

```
<filter>
    <filter-name>LoginFilter</filter-name>
    <filter-class>by.gsu.epamlab.filters.LoginFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>LoginFilter</filter-name>
    <url-pattern>/operation/*</url-pattern>
```

Этап 4

Реализовать дополнительный функционал.

В проекте 1 - это операции с файлом.

Первоначально реализовать операции, не задумываясь о конфликте имен файлов. Если останется время, то тогда заняться данной проблемой.

На все запросы с файлом распространяются соображения этапа 3 (метод отправки, возврат управления, аутентификация, синхронизация).

В главе 12 пособия “Java for the Web with Servlets, JSP, and EJB: A Developer's Guide to J2EE Solutions” by Budi Kurniawan (в папке [gsu-epam-lab/support/texts](https://gsu-epam-lab.github.io/support/texts/java-for-the-web-with-servlets_jsp_and-ejb.pdf) см. файл `java-for-the-web-with-servlets_jsp_and-ejb.pdf`) приведен алгоритм даунлода файла и пример реализации на jsp-странице. В учебном проекте необходимо выполнить обработку запроса согласно шаблона MVC.

В главе 13 этого же пособия рассмотрен процесс аплода файла. Также требуется следование шаблону MVC.