# Java Web Development Basics 12:47

# Exam Preparation

Exam Preparation problems for the [“Java Web Development Basics” course @ SoftUni](https://softuni.bg/courses/java-web-development-basics). Submit your solutions on the course page, so that you can be evaluated by your fellow colleagues.

# Java EE Block – 30 pts

The next several tasks, will test your knowledge on the Java EE components you’ve seen throughout the course. For your solutions, just submit your source code. It will be configured and deployed on Tomcat, while being checked.

## Simple JSPs – 10 pts

Using your knowledge on Java EE, implement 2 simple JSPs which will be used to render data dynamically. But to render data dynamically, we will need some sort of data object. So let’s create our Product. The Product is a data object which stores data about an abstract **product**. You will see later what it will be used for.

First implement a class Product – which has these properties:

* Name – a **String**.
* Description – a **String**.
* Type – can be one of the following values – (“Food”, “Domestic”, “Health”, “Cosmetic”, “Other”)

**Initialize** a collection which holds **3 Products** with the **following data**:

|  |  |  |  |
| --- | --- | --- | --- |
| Products | | | |
| Field | 1 | 2 | 3 |
| Name | Chushkopek | Injektoplqktor | Plumbus |
| Description | A universal tool for … | Dunno what this is… | A domestic tool for everything |
| Type | Domestic | Cosmetic | Food |

Now that you’ve got a data collection, we can start implementing our JSPs.

### All Products

Implement a JSP – products/all.jsp, which renders only the names of the Products. Upon clicking a name of a Product, you should be **redirected** to a **details page**, with **query parameter** – the name of the Product.

### Product Details

Implement a JSP – products/details.jsp, which **renders full data** about the **selected Product**. The selected Product should be extracted from the collection, using the name from the **query parameters**.

**Note**: The design of the tasks stated above is by your choice, if it fulfills the requirements.

## Simple Servlets – 20 pts

Using your knowledge on Java EE, implement 3 simple Servlets, which work with a shared data.

### Product Create

Implement a **Servlet** – ProductCreateServlet, which listens on route “/products/create”.

Upon a **GET** request, it should return a form which accepts **3 inputs** – a name, a description and a type.  
**NOTE**: You should only be able to submit “Food”, “Domestic”, “Health”, “Cosmetic”, “Other” as values for the type.

The **form** should send a POST request to the same route.

Upon a **POST** request, you should **redirect** to “/products/all”.

### Product All

Implement a **Servlet** – ProductAllServlet, which listens on route “/products/all”.

Upon a **GET** request, the **Servlet** should render a page with the names of all created Products. Upon clicking a name of a Product, you should be **redirected** to a “/products/details”, with **query parameter** – the name of the Product.

### Product Details

Implement a **Servlet** – ProductDetailsServlet, which listens on route “/products/details”.

Upon a **GET** request, the **Servlet** should extract the Product with the given name in the **query parameters**, and render a page with **full information** about it.

**Note**: The design of the tasks stated above is by your choice, if it fulfills the requirements.

**Note**: The algorithm for transfering data, is by your choice, if it fulfills the requirements.

# Application Block – 70 pts (12:47-13:50) (14:05-14:49 P) (15:00-) NEXT DAY: (15:17)

# CHUSHKA

**CHUSHKA** (**C**entral **H**ierarchically-**U**niversal **S**ales **H**ost **K**ickstarter **A**pplication) is a universal web application for selling products, like a web shop. You have been tasked to implement this application by the **Codex Input / Output** organization. There are several requirements you must follow in the implementation.

The application has been developed to somewhere. The previous developer was not able to make the Admin functionality so that falls to you. He also broke some of the Framework’s main modules: Javache, Broccolina, Summer.

Below, you will see, how the application should behave, if implemented correctly. You should also fix It, so that it is implemented correctly.

## Technological Requirements

* Use the Javache Web Server
* Use the Broccolina and Toyote request handlers
* Use the Summer Framework
* Use Hibernate native (no Spring Data)

The Technological Requirements are **ABSOLUTE**. If you **do not follow** them, you will **NOT** be scored.

Now that you know the Technological Requirements, let us see what are the Functional Requirements.

## Database Requirements

The **Database** of the **CHUSHKA** application needs to support **3 entities**:

### User

* Has an Id – a UUID String.
* Has an Username
* Has a Password
* Has a Full Name
* Has an Email
* Has an Role – can be one of the following values (“User”, “Admin”)

### Product

* Has an Id – a UUID String.
* Has a Name
* Has a Price
* Has a Description
* Has a Type – can be one of the following values (“Food”, “Domestic”, “Health”, “Cosmetic”, “Other”)

### Order

* Has an Id – a UUID String
* Has a Product – a Product
* Has a Client – an User
* Has an Ordered On – a LocalDateTime

Implement the entities with the **correct datatypes**.

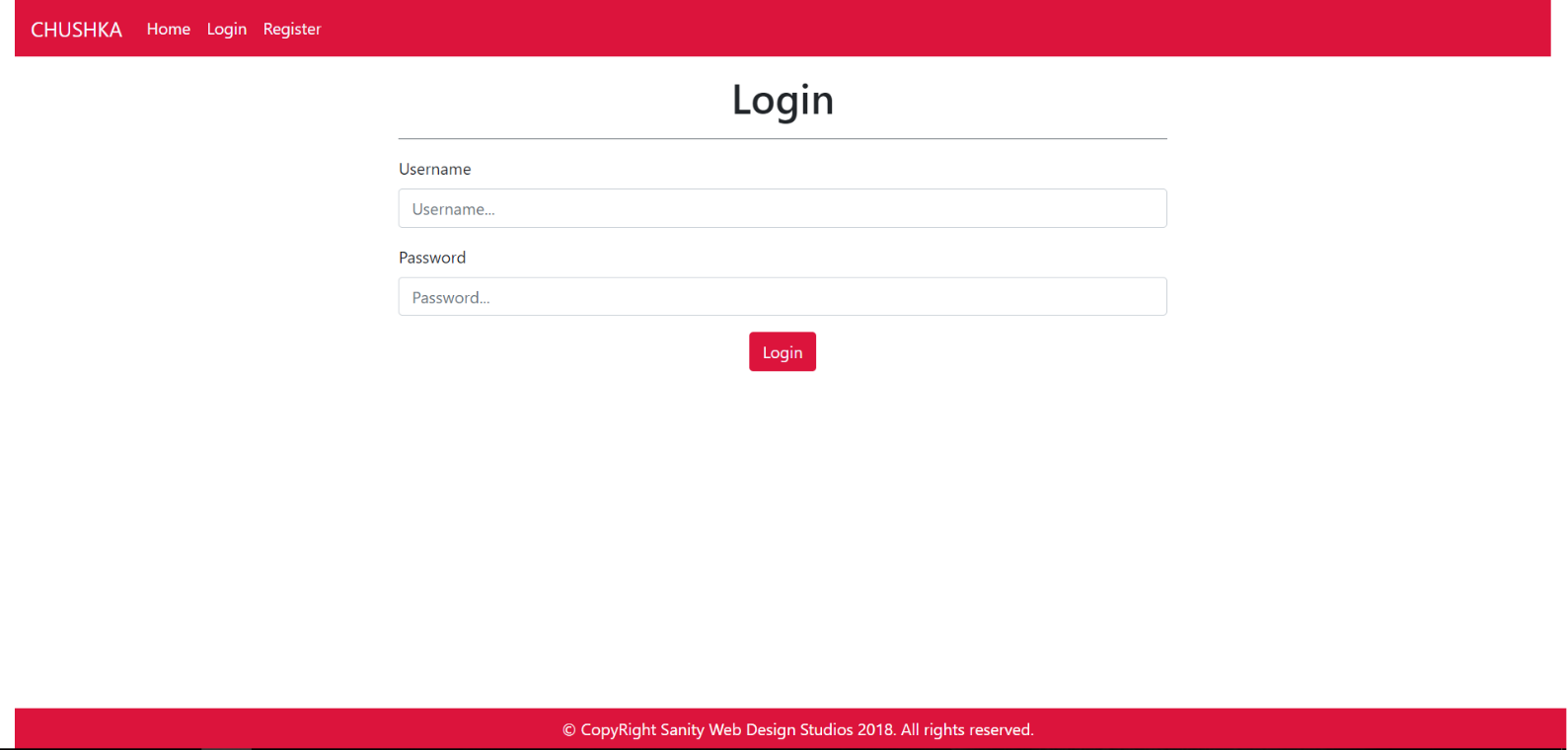
## Template Requirements

### Guest Templates

These are the **templates** and **functionalities**, accessible by Guests (**logged out** users).

#### Index Template (route = “/”) (logged out user)

#### Login Template (route = “/login”) (logged out user)



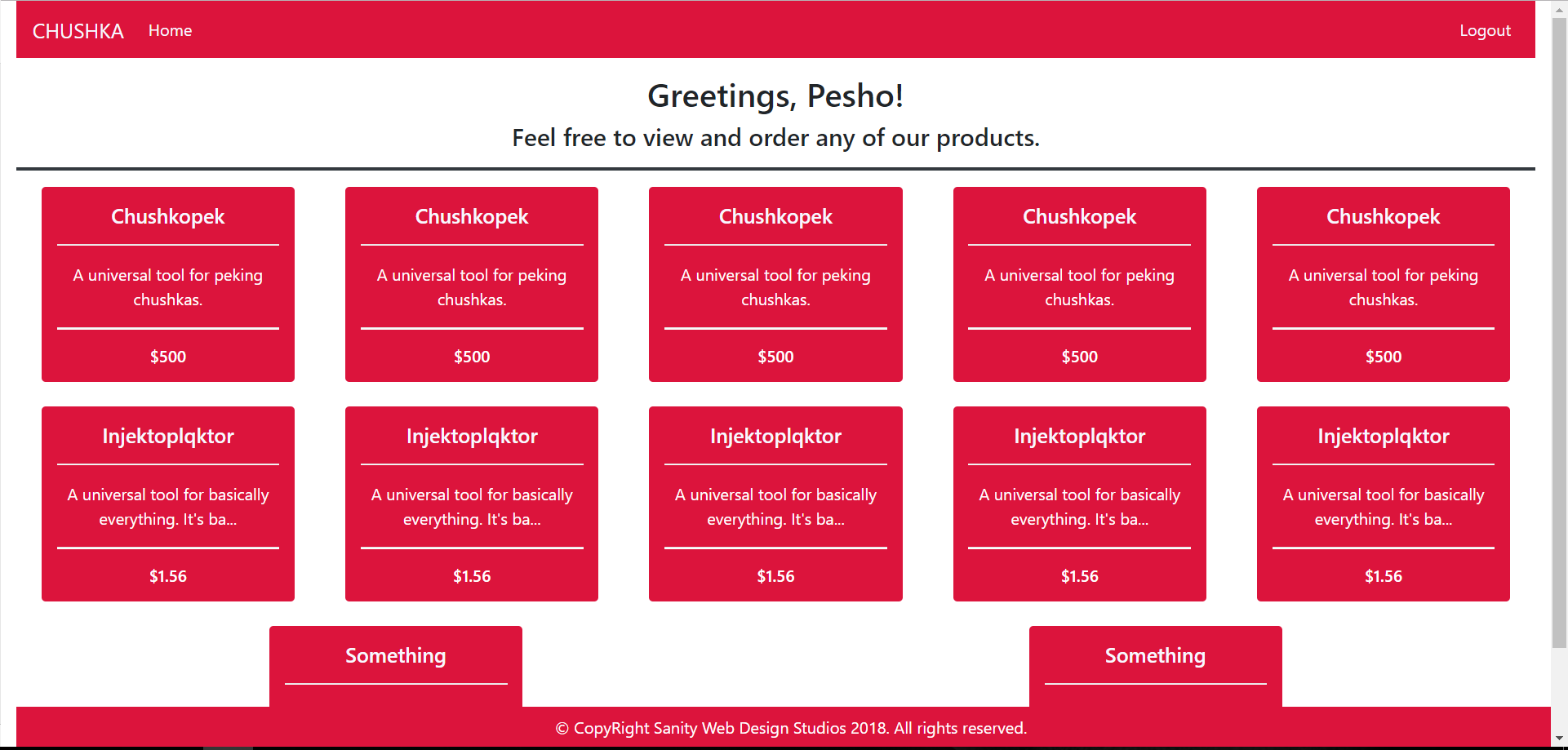
#### Register Template (route = “/register”) (logged out user)

### User Templates

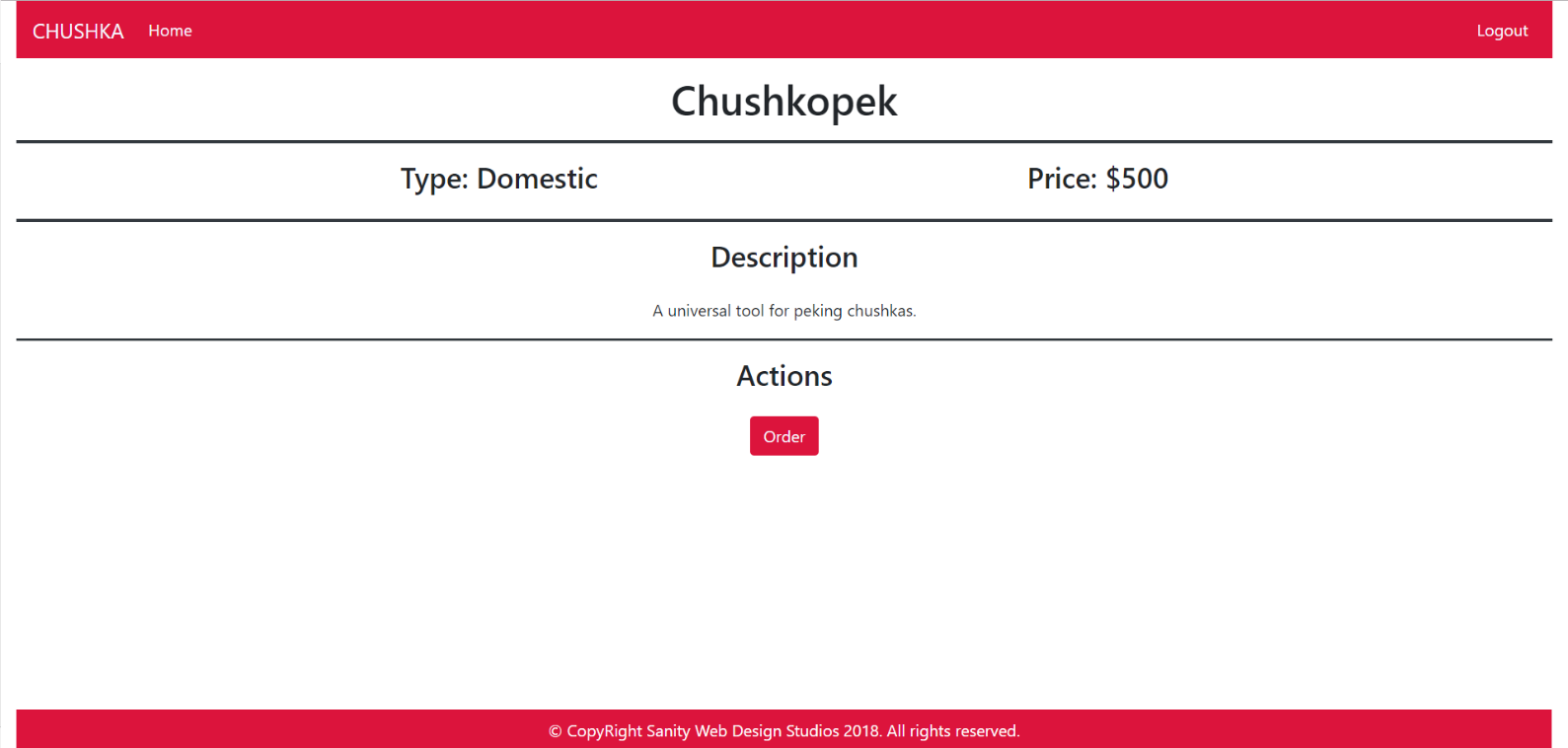
These are the **templates** and **functionalities**, accessible by Users (**logged in** users with Role - User).

#### LoggedIn Index Template (route=”/home”) (logged in user)

**NOTE**: Products on this page have their description substringed to the 50th symbol, and then accompanied by 3 dots “…”



#### Product Details Template (route=”/products/details/{id}) (logged in user)

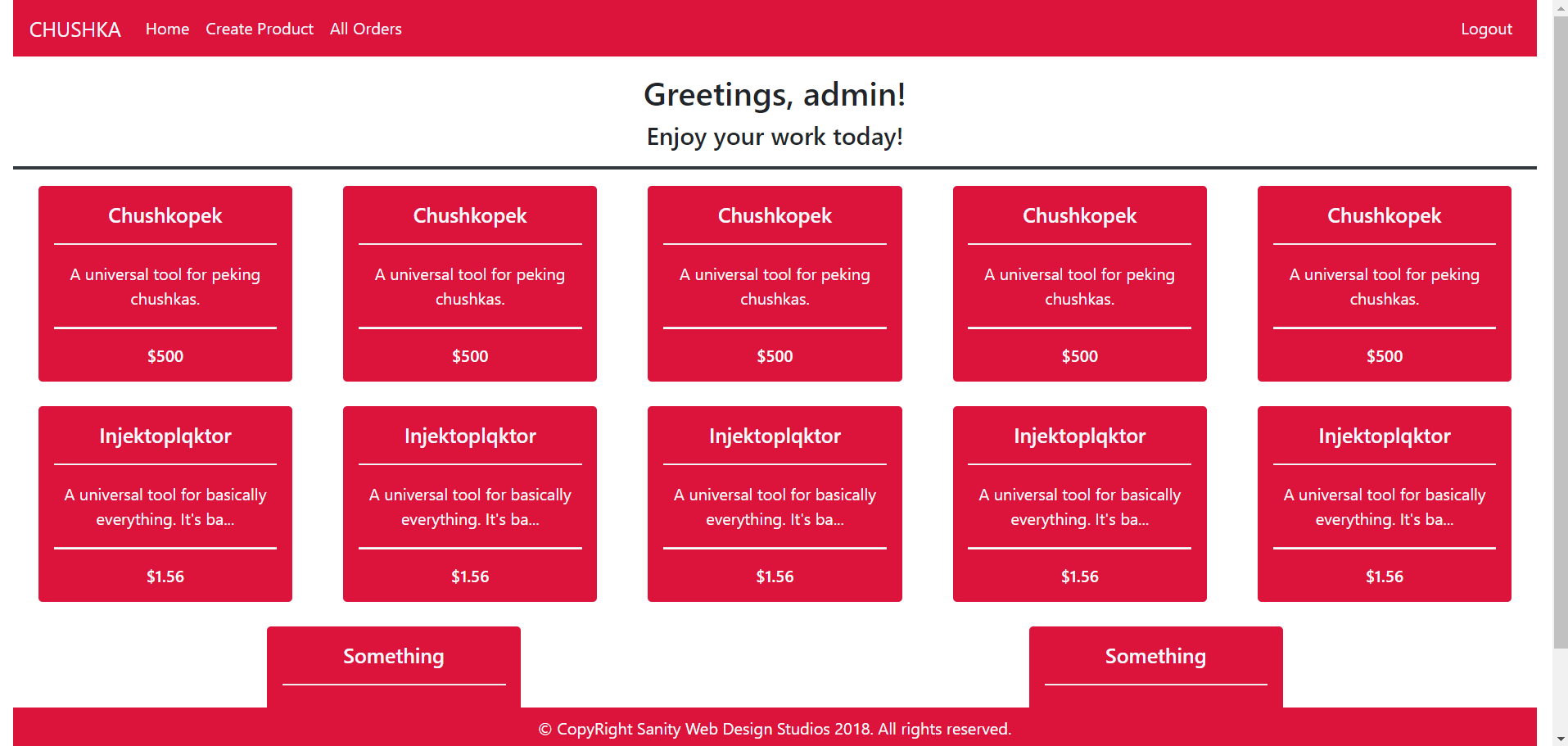


### Admin Templates

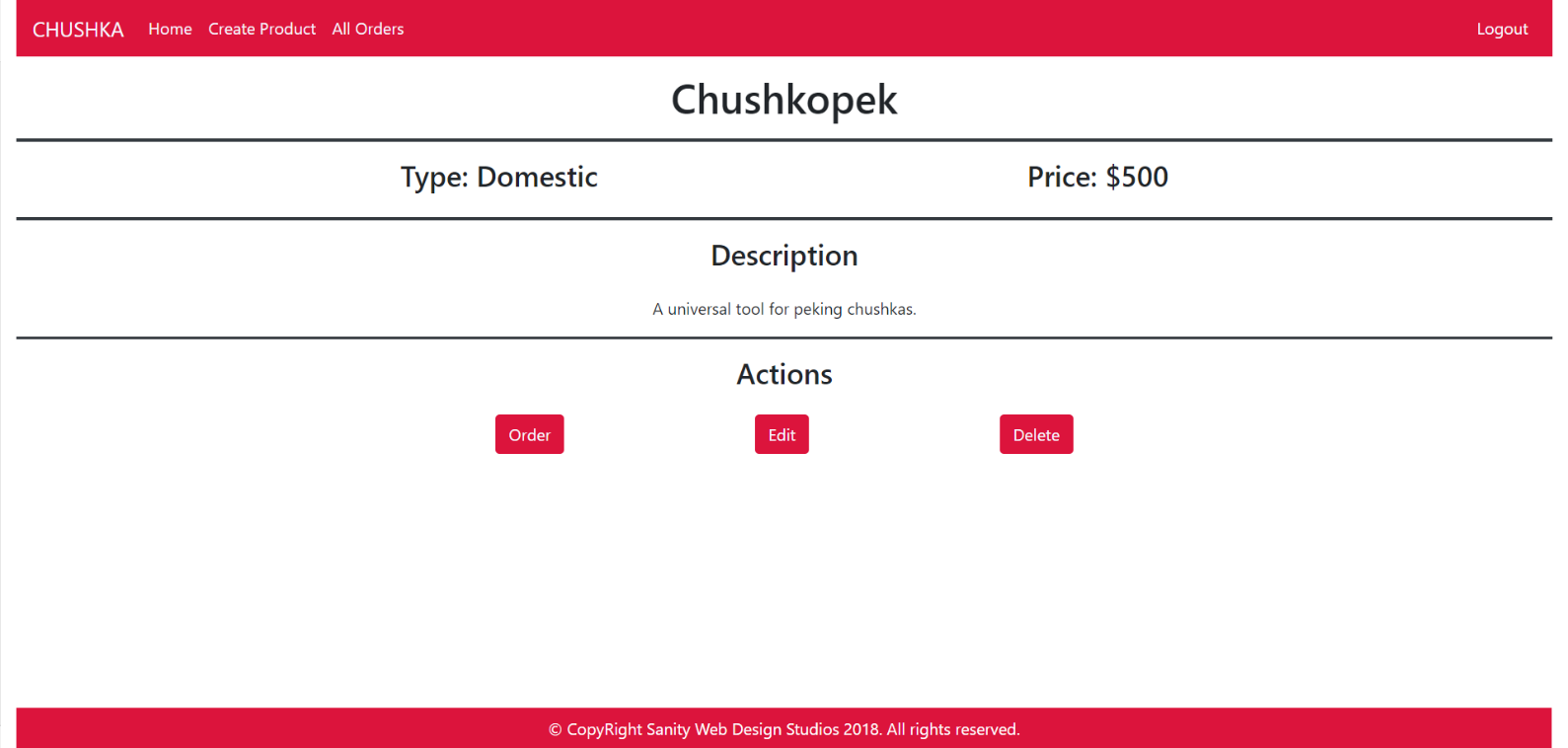
These are the **templates** and **functionalities**, accessible by Admins (**logged in** users with Role - Admin).

#### Admin Index Template (route=”/home”) (logged in admin)

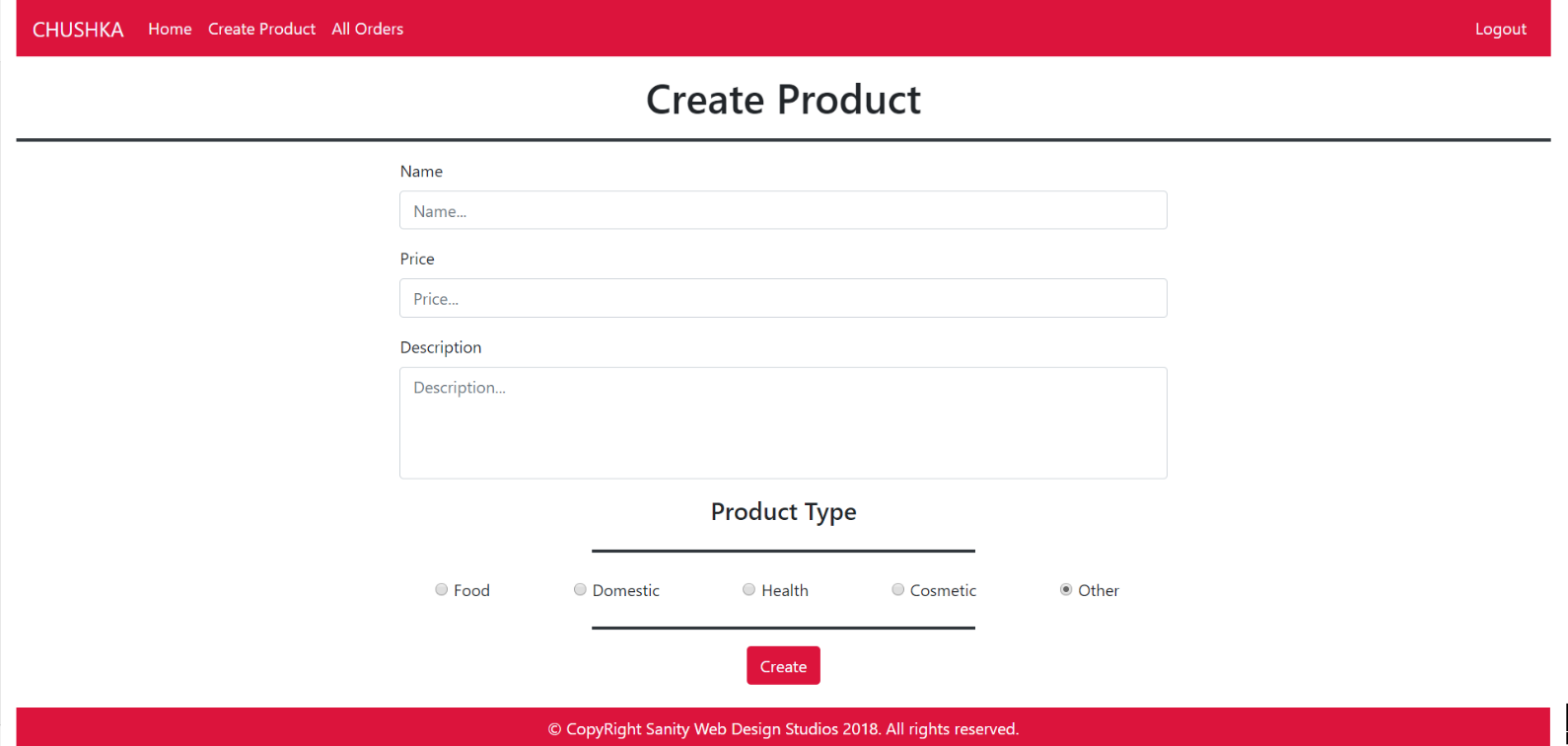
**NOTE**: Products on this page have their description substringed to the 50th symbol, and then accompanied by 3 dots “…”



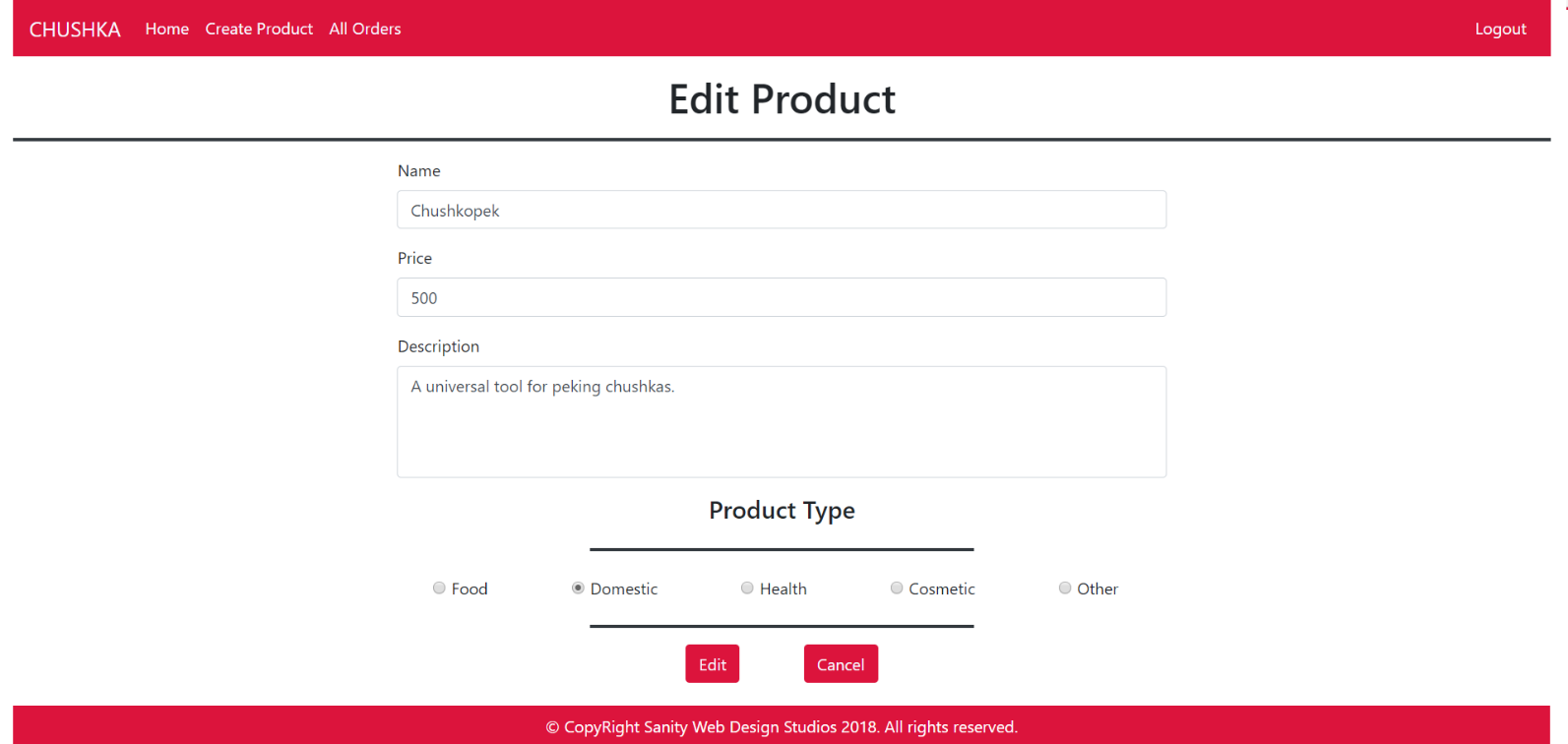
#### Prod. Details Admin Temp. (route=”/products/details/{id}”) (logged in admin)



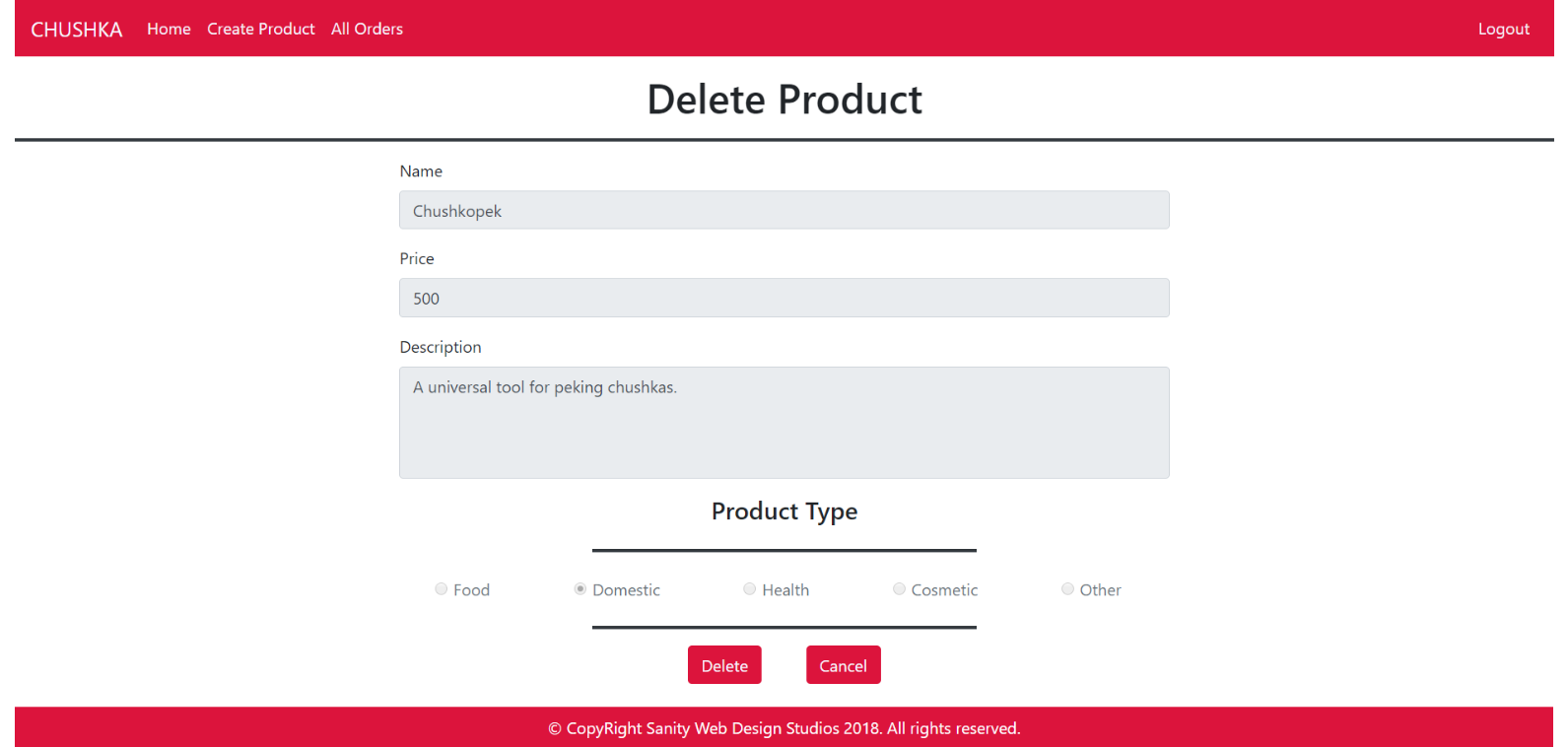
#### Product Create Template (route=”/products/create”) (logged in admin)



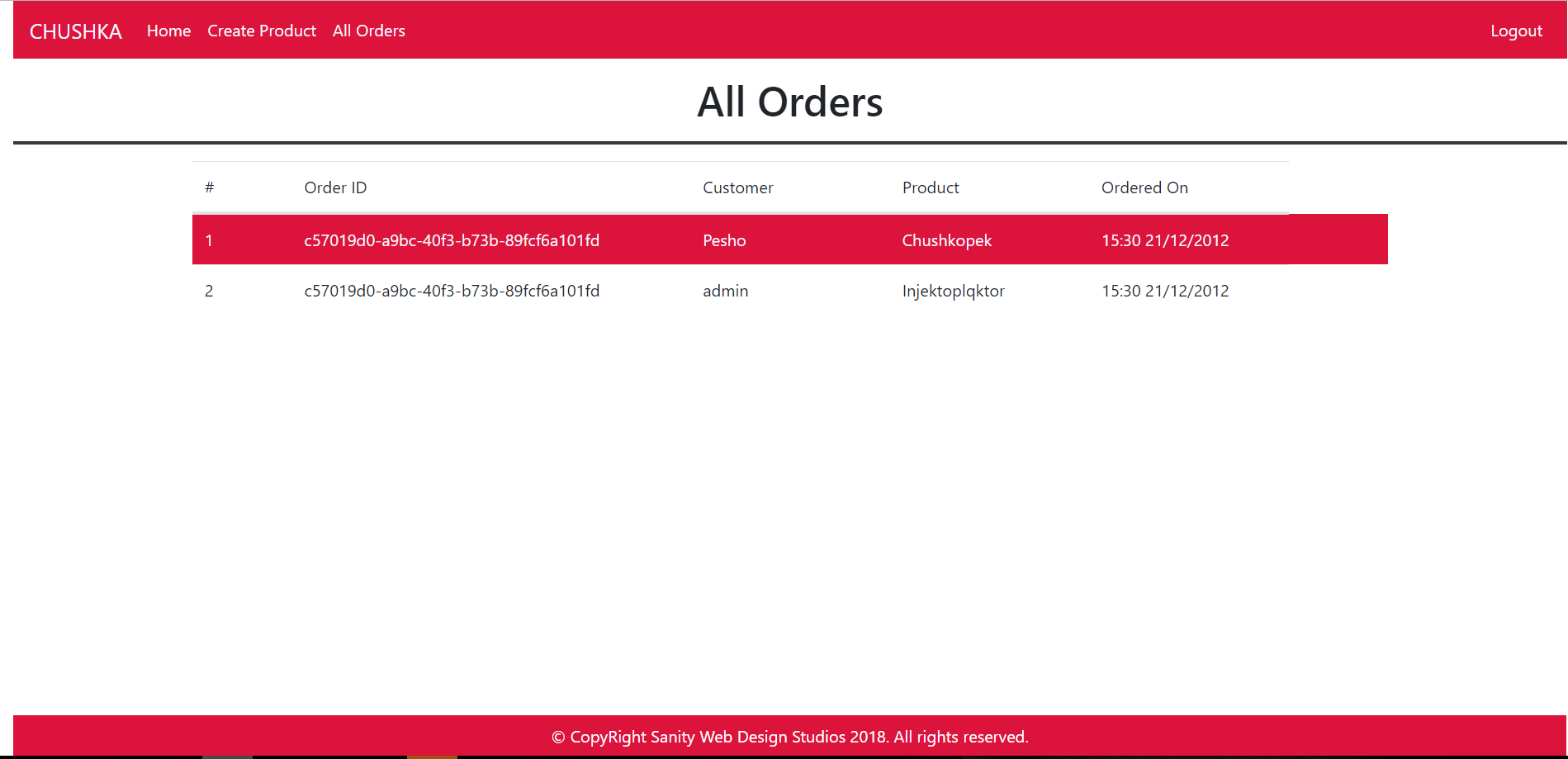
#### Product Edit Template (route=”/products/edit/{id}”) (logged in admin)



#### Product Delete Template (route=”/products/delete/{id}”) (logged in admin)



#### All Orders Template (route=”/orders/all”) (logged in admin)



Some of the templates have been given to you in the application skeleton, but the others will be for you to implement, so make sure you implement them correctly. You can use the given ones as helpers.

**NOTE**: The templates should look **EXACTLY** as shown above.

**NOTE**: The templates do **NOT** **require** **additional** **CSS**. Only **bootstrap** is enough.

## Functional Requirements

The Functionality Requirements describe the functionality that the **Application** must support.

The **application** should provide Guest (not logged in) users with the functionality to login, register and **view** the Index page.

The **application** should provide Users (logged in) with the functionality to logout, and view all products, and **order** a Product.

The **application** should provide Admins (logged in) with the functionality to logout, and view all products, **order** a Product, **create** a Product, **edit** a Product, **delete** a Product, and **view** all Orders.

When you register a new User, it should be assigned with a **role** – User.

Upon clicking on the **rectangular object** holding the data about a certain product, you should be redirected to its Details page.

Upon **ordering** a Product, an Order should be **created** and **persisted**.

**NOTE**: To test your **Admin** **functionality**, make one user an **Admin** through the database, or make the **first registered** user – an **Admin**.

The **application** should **store** its **data** into a MySQL database, using Hibernate native.

## Security Requirements

The Security Requirements are mainly access requirements. Configurations about which users can access specific functionalities and pages.

* Guest (not logged in) users can access Index page and functionality.
* Guest (not logged in) users can access Login page and functionality.
* Guest (not logged in) users can access Register page and functionality.
* Users (logged in) can access User LoggedIn Index page and functionality.
* Users (logged in) can access User Product Details page and functionality.
* Users (logged in) can access the Order Product functionality.
* Users (logged in) can access Logout functionality.
* Admins (logged in) can access **every functionality** a **normal** logged in User can.
* Admins (logged in) can access Admin LoggedIn Index page and functionality.
* Admins (logged in) can access the Admin Product Details page and functionality.
* Admins (logged in) can access the Product Create page and functionality.
* Admins (logged in) can access the Product Edit page and functionality.
* Admins (logged in) can access the Product Delete page and functionality.
* Admins (logged in) can access the All Orders page and functionality.