The Template Method pattern can be effectively used in controllers to manage exceptions, user logout, and session management, ensuring consistent behavior across different endpoints. By defining a template method in a base controller class, you can encapsulate the common logic for these functionalities, while allowing subclasses (endpoint-specific controllers) to customize specific steps. This approach promotes code reusability, maintainability, and a clear separation of concerns.

Here's how it can be applied:

1. **Abstract Base Controller:** Create an abstract base controller class. This class will define the template method (e.g., handleRequest) and abstract methods for the specific steps to be customized.
2. **Template Method:** The template method outlines the overall algorithm. For example:

public abstract class BaseController {  
 public final void handleRequest(Request request, Response response) {  
 try {  
 setupSession(request, response);  
 Object data = processRequest(request, response);  
 handleResponse(data, response);  
 } catch (Exception e) {  
 handleException(e, response);  
 } finally {  
 cleanupSession(request, response);  
 }  
 }  
  
 protected abstract void setupSession(Request request, Response response);  
 protected abstract Object processRequest(Request request, Response response);  
 protected abstract void handleResponse(Object data, Response response);  
 protected abstract void handleException(Exception e, Response response);  
 protected abstract void cleanupSession(Request request, Response response);  
 }

**1. Concrete Subclasses (Endpoint Controllers):**

Each endpoint (e.g., /users/login, /products/search) will have its own controller class inheriting from the BaseController.

**2. Customizing Steps:**

Subclasses will override the abstract methods to provide specific implementations for their endpoint.

* + setupSession: May involve checking user authentication, setting up session attributes, or redirecting unauthorized users.
  + processRequest: Handles the core logic of the endpoint (e.g., retrieving data from a database, processing user input).
  + handleResponse: Formats the response data (e.g., converting to JSON, XML).
  + handleException: Handles specific exceptions for the endpoint (e.g., validation errors, data not found). It might log the error, return a specific HTTP status code, or display a user-friendly message.
  + cleanupSession: Performs actions like logging out a user or invalidating a session after processing the request.

**3. Benefits:**

* + **Consistency:** Ensures a consistent approach to exception handling, logout, and session management across all controllers.
  + **Reusability:** Avoids code duplication by encapsulating common logic in the base class.
  + **Extensibility:** Easily add new endpoints or customize existing ones by extending the base controller.
  + **Maintainability:** Changes to the overall algorithm (template method) only need to be made in one place.

**4. Example (Login Endpoint):**

public class LoginController extends BaseController {  
 @Override  
 protected void setupSession(Request request, Response response) {  
 *// Check if the user is already logged in. If so, redirect.*  
 }  
  
 @Override  
 protected Object processRequest(Request request, Response response) {  
 *// Get username and password from the request*  
 *// Authenticate the user against the database*  
 *// If successful, create a session*  
 }  
  
 @Override  
 protected void handleResponse(Object data, Response response) {  
 *// Send a success response (e.g., JSON with user details)*  
 }  
  
 @Override  
 protected void handleException(Exception e, Response response) {  
 *// Handle authentication errors, validation errors, etc.*  
 }  
  
 @Override  
 protected void cleanupSession(Request request, Response response) {  
 *// If login was successful, set up session cookie or token*  
 }  
 }

By using the Template Method pattern, you can create a robust and maintainable system for managing exceptions, logout, and session management in your controllers, while ensuring consistent behavior across all your endpoints.