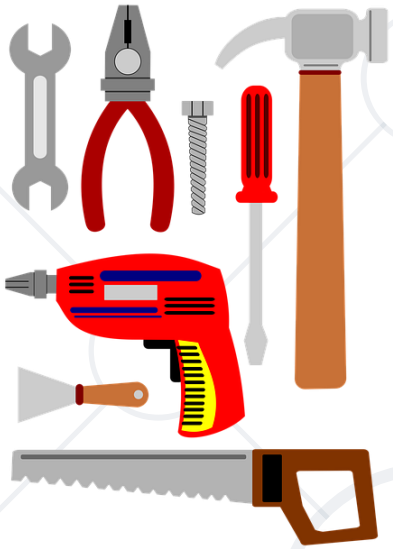
# Modular Applications

## Common Scenarios and Best Practices




**SoftUni Team**

**Technical Trainers**

Software University


SoftUni


**Software University**

https://softuni.bg

# Have a Question?

Software University

sli.do

#js-advanced

# Table of Contents



# Best Practices

## Common Scenarios and Techniques

# Component Approach

- **Components** are a common theme among contemporary **frameworks** and **libraires**
- Focused on **separation of concerns** and **composability**:
  - Combine **presentation**, **style** and **business logic** in a single unit
  - Encapsulate **state** and **control**
  - Expose only necessary **interfaces**
  - **Decoupled** from the environment (via **dependency injection**)
  - Highly **composable** with other components

# Application State

- Avoid **storing state** in the DOM
- Avoid attempting to **infer state** from the DOM
  - E.g., using the text content of an HTML element to reconstruct what a database record looked like
- Try to write **declarative** DOM logic:
  - Describe what the DOM **should** look like for a given **state**
  - When the state **changes**, the DOM **follows**
  - Rendering libraries allow for **efficient DOM redraws**

# Routing

- Attempt to couple application content with the **URL route**
  - This allows more efficient use of **browser history** and **sharing** links to specific parts of the application
  - Can be done with **paths**, **query parameters** or **fragments**
- Examples:
  - **Search terms** should be included as query parameters
  - If a catalog is paginated, include the **current page** in the URL
  - Toggleable content or **sub-navigation** can also be included

# Action Feedback

- Provide **instant acknowledgement** for user actions:
  - Change appearance when links and buttons are **clicked**
  - Clear the view on **navigation**
  - Show **loading indicators** during network requests
  - **Disable input** during requests, to prevent double submission
- Don't overdo feedback:
  - **Don't** attempt to validate input before the user has finished
  - There's **no need** to show notifications for everything

# User Input

- Always **sanitize** user input:
  - Remove leading and trailing **whitespace**
  - Do not automatically include all form data in the request – only pick the **properties** that are **part of the collection**
  - **Prevent** insertion of **HTML** anywhere in your code
  - **Never** use `eval` where user input is involved
- Remember that the front-end application **does not** provide security – the **server** must **double check** all user actions

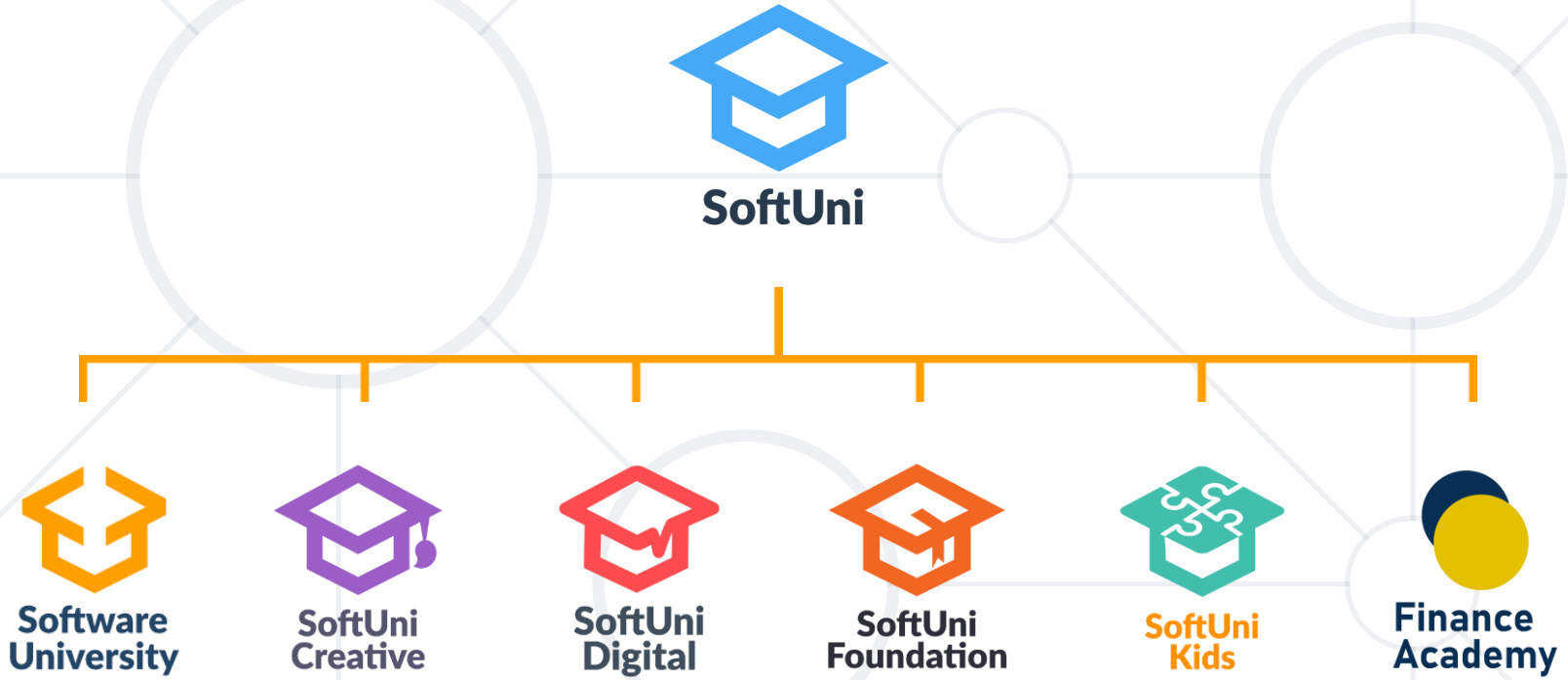
# Error Handling

- Always **anticipate errors** from **network requests** and **user input**
- Errors that can be **resolved automatically** can be handled **behind the scenes**
  - You can **catch** them where they occur
  - E.g., data parsing errors, empty server responses, etc.
- Errors that **concern user action** must be **propagated** to the **presentation layer** of the app (**rethrow**, or don't catch)
  - E.g., validation errors

# Summary

- **Components** are a common theme among contemporary frameworks and libraires
- **Routing** allows more efficient use of browser history and sharing links to specific parts of t he application
- **Error handling** involves using try-catch blocks to gracefully manage and respond to potenti al runtime errors in code execution.

# Questions?



# SoftUni Diamond Partners

Software University












# Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

# License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://softuni.org
- © Software University – https://softuni.bg