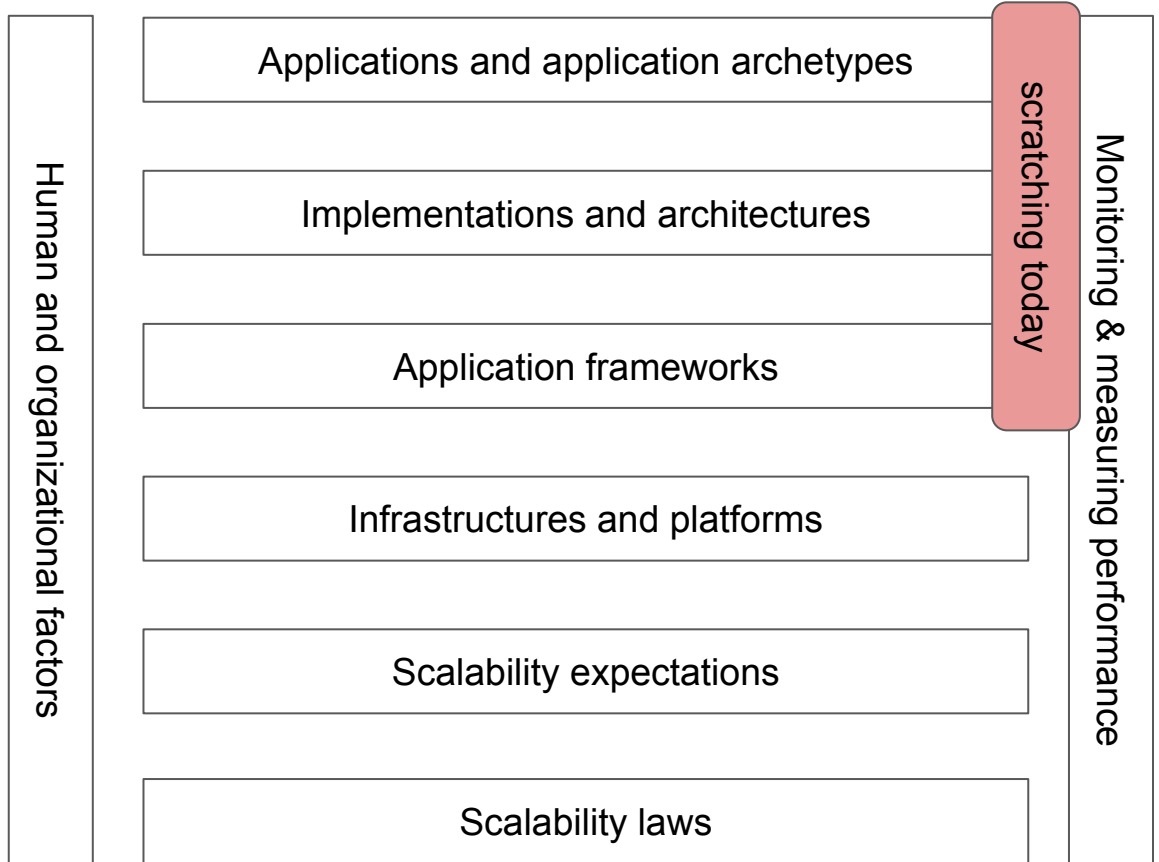


Designing and Building Scalable Web Applications

Lecture 3 / 7.11.2022

The Big Picture



Agenda

- Rendering Approaches
- Web Application Archetypes
- Client-Side Web Development (continued)
- Measuring Performance (continued)
- Second Course Project

Rendering Approaches

Rendering Approaches

Rendering Approaches

- Server-side rendering
 - Data injected to a template on the server producing HTML. Produced HTML is sent to client.

Rendering Approaches

The classic approach from e.g.
Web Software Development
(CS-C3170)



- Server-side rendering
 - Data injected to a template on the server producing HTML. Produced HTML is sent to client.

Rendering Approaches

- Server-side rendering
 - Data injected to a template on the server producing HTML. Produced HTML is sent to client.

The classic approach from e.g.
Web Software Development
(CS-C3170)



Loads full HTML (and linked
resources) on each request.

Rendering Approaches

The classic approach from e.g.
Web Software Development
(CS-C3170)



- Server-side rendering
 - Data injected to a template on the server producing HTML. Produced HTML is sent to client.
- Client-side rendering
 - Server sends HTML with JavaScript to the client. Client retrieves data based on JavaScript. Data rendered using JavaScript on the client.

Loads full HTML (and linked
resources) on each request.

Rendering Approaches

- Server-side rendering
 - Data injected to a template on the server producing HTML. Produced HTML is sent to client.
- Client-side rendering
 - Server sends HTML with JavaScript to the client. Client retrieves data based on JavaScript. Data rendered using JavaScript on the client.

The classic approach from e.g.
Web Software Development
(CS-C3170)

Loads full HTML (and linked
resources) on each request.

Allows storing HTML and
JavaScript on a CDN. Role of
"main server" is to serve data.

Rendering Approaches

- Server-side rendering
 - Data injected to a template on the server producing HTML. Produced HTML is sent to client.
- Client-side rendering
 - Server sends HTML with JavaScript to the client. Client retrieves data based on JavaScript. Data rendered using JavaScript on the client.

The classic approach from e.g.
Web Software Development
(CS-C3170)

Loads full HTML (and linked
resources) on each request.

Allows storing HTML and
JavaScript on a CDN. Role of
"main server" is to serve data.

No need to reload full HTML on
each request → typically
improved user experience.

Rendering Approaches

- Server-side rendering
 - Data injected to a template on the server producing HTML. Produced HTML is sent to client.
- Client-side rendering
 - Server sends HTML with JavaScript to the client. Client retrieves data based on JavaScript. Data rendered using JavaScript on the client.

The classic approach from e.g. Web Software Development (CS-C3170)

Loads full HTML (and linked resources) on each request.

Allows storing HTML and JavaScript on a CDN. Role of "main server" is to serve data.

No need to reload full HTML on each request → typically improved user experience.

Needs client-side logic.

Rendering Approaches

- Server-side rendering
 - Data injected to a template on the server producing HTML. Produced HTML is sent to client.
- Client-side rendering
 - Server sends HTML with JavaScript to the client. Client retrieves data based on JavaScript. Data rendered using JavaScript on the client.

The classic approach from e.g. Web Software Development (CS-C3170)

Loads full HTML (and linked resources) on each request.

Allows storing HTML and JavaScript on a CDN. Role of "main server" is to serve data.

No need to reload full HTML on each request → typically improved user experience.

Needs client-side logic.

Multiple requests made before page is shown.

Rendering Approaches

- **Server-side rendering**
 - Data injected to a template on the server producing HTML. Produced HTML is sent to client.
- **Client-side rendering**
 - Server sends HTML with JavaScript to the client. Client retrieves data based on JavaScript. Data rendered using JavaScript on the client.
- **Hybrid approaches**
 - E.g. some HTML content is produced on the server. Server sends HTML and JavaScript. Additional content retrieved based on JavaScript.

The classic approach from e.g. Web Software Development (CS-C3170)

Loads full HTML (and linked resources) on each request.

Allows storing HTML and JavaScript on a CDN. Role of "main server" is to serve data.

No need to reload full HTML on each request → typically improved user experience.

Needs client-side logic.

Multiple requests made before page is shown.

Rendering Approaches

- Server-side rendering
 - Data injected to a template on the server producing HTML. Produced HTML is sent to client.
- Client-side rendering
 - Server sends HTML with JavaScript to the client. Client retrieves data based on JavaScript. Data rendered using JavaScript on the client.
- Hybrid approaches
 - E.g. some HTML content is produced on the server. Server sends HTML and JavaScript. Additional content retrieved based on JavaScript.

The classic approach from e.g. Web Software Development (CS-C3170)

Loads full HTML (and linked resources) on each request.

Allows storing HTML and JavaScript on a CDN. Role of "main server" is to serve data.

No need to reload full HTML on each request → typically improved user experience.

Needs client-side logic.

Multiple requests made before page is shown.

Best of both worlds?

Rendering Approaches

Rendering Approaches

- Server-side rendering:
 - Data injected to a template on server to produce HTML.
 - Effectively uses some resources for rendering.

Rendering Approaches

- Server-side rendering:
 - Data injected to a template on server to produce HTML.
 - Effectively uses some resources for rendering.
- Client-side rendering and hybrid approaches:
 - Shown content created using JavaScript (on client).
 - Server-side rendering (hybrid approaches) typically done when application is built.
 - Fewer resources required for rendering (from the application point of view).

Hybrid Approach Example

Hybrid Approach Example

Our course site at <https://fitech101.aalto.fi/designing-and-building-scalable-web-applications/> -
content written using MDX, components written with React, site built using Gatsby

<https://mdxjs.com/>

<https://www.gatsbyjs.com/>

<https://reactjs.org/>

Application Archetypes

Application Archetypes

- *Archetype (dictionary.com)*
 - the original pattern or model from which all things of the same kind are copied or on which they are based; a model or first form; prototype

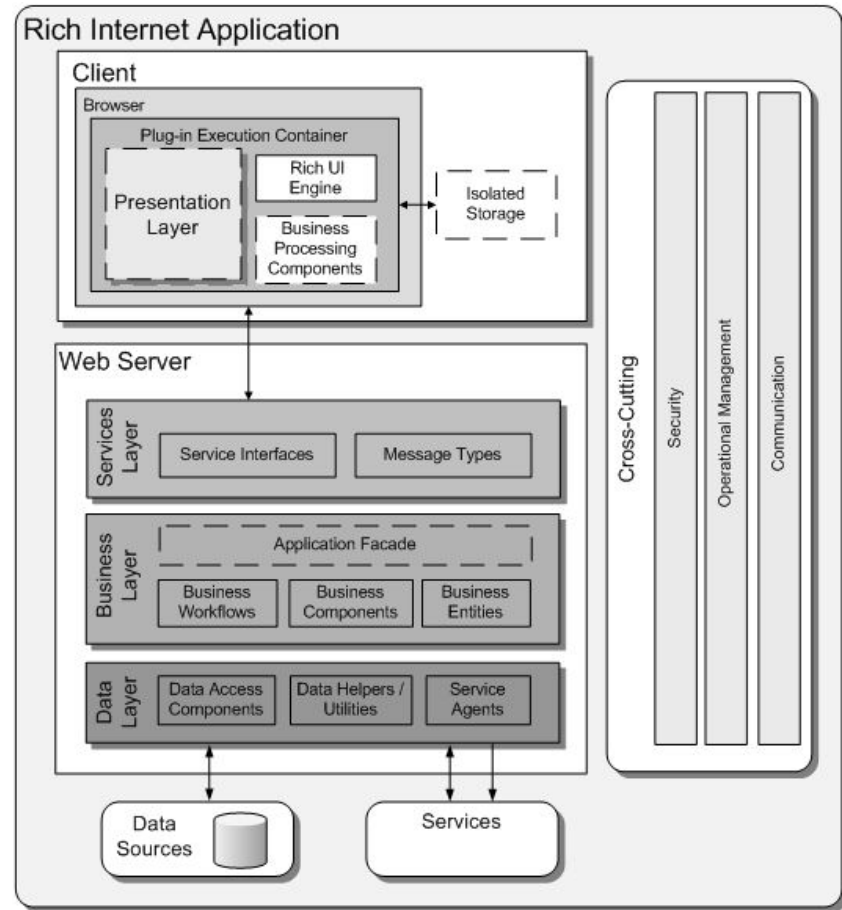
Application Archetypes

- *Archetype (dictionary.com)*
 - the original pattern or model from which all things of the same kind are copied or on which they are based; a model or first form; prototype
- Microsoft Application Architecture Guide (2009):
 - Common application archetypes include mobile applications, rich client applications, rich internet applications, service applications, web applications

[https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ee658104\(v=pandp.10\)](https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ee658104(v=pandp.10))

Application Archetypes

- *Archetype* (dictionary.com)
 - the original pattern or model from which all things of the same kind are copied or on which they are based; a model or first form; prototype
- Microsoft Application Architecture Guide (2009):
 - Common application archetypes include mobile applications, rich client applications, rich internet applications, service applications, web applications

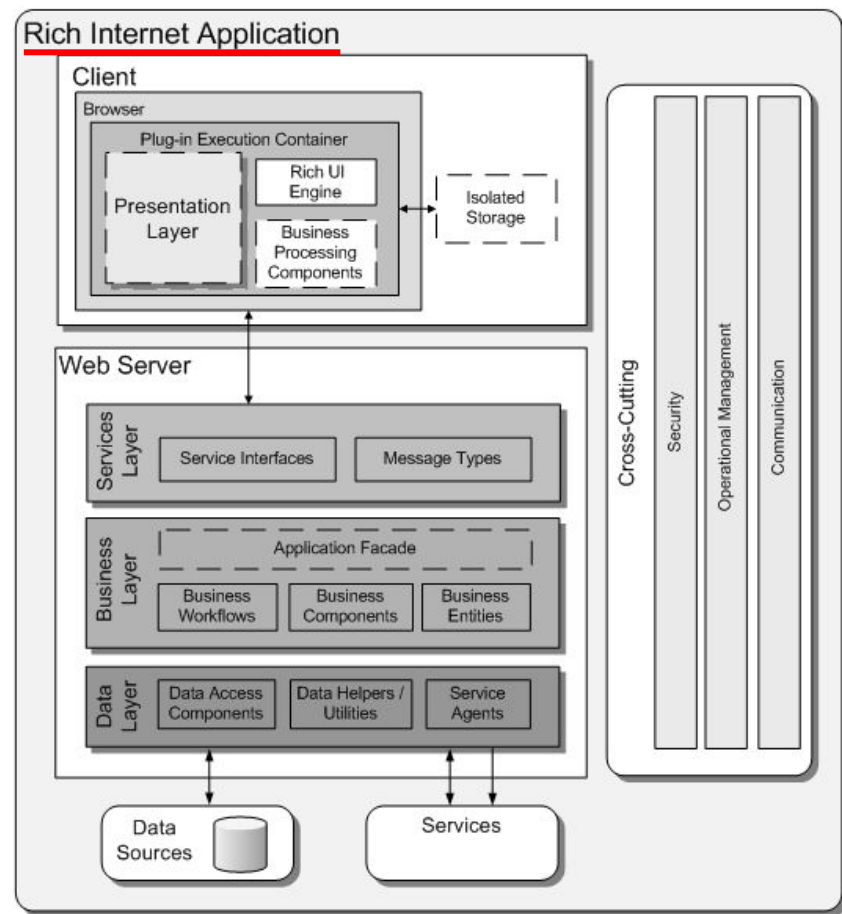


[https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ee658104\(v=pandp.10\)](https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ee658104(v=pandp.10))

Macromedia 2002 ~ Web applications with features and functionality normally associated with desktop applications

Application Archetypes

- *Archetype* (*dictionary.com*)
 - the original pattern or model from which all things of the same kind are copied or on which they are based; a model or first form; prototype
- Microsoft Application Architecture Guide (2009):
 - Common application archetypes include mobile applications, rich client applications, rich internet applications, service applications, web applications



[https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ee658104\(v=pandp.10\)](https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ee658104(v=pandp.10))

Application Archetypes

Application Archetypes

- Mequoda Research Team (2013) – 16 primary website archetypes inferred from > 2000 websites.
 - Membership sites, newsletter websites, reference websites, periodical websites, magazine websites, application websites, community websites, portal websites, blog websites, cart-based store sites, event-based store sites, solo-based store sites, lead generation sites, search engines, classified sites, directory sites.

Application Archetypes

Application Archetypes

- Community websites (e.g. Facebook, LinkedIn)
- Streaming websites (e.g. Netflix, HBO Max)
- Media websites (e.g. Instagram, Pinterest)
- Online shops (e.g. Amazon, Ebay)
- Instant messaging (e.g. WhatsApp, Telegram)
- Discussion boards (e.g. Reddit, Stackoverflow)
- Blog websites (e.g. Blogger, Medium)

Application Archetypes

Starting to identify differences in applications relevant to understand the underlying technologies. Needs differ between apps.

- Community websites (e.g. Facebook, LinkedIn)
- Streaming websites (e.g. Netflix, HBO Max)
- Media websites (e.g. Instagram, Pinterest)
- Online shops (e.g. Amazon, Ebay)
- Instant messaging (e.g. WhatsApp, Telegram)
- Discussion boards (e.g. Reddit, Stackoverflow)
- Blog websites (e.g. Blogger, Medium)

Application Archetypes

Starting to identify differences in applications relevant to understand the underlying technologies. Needs differ between apps.

- Community websites (e.g. Facebook, LinkedIn)
- Streaming websites (e.g. Netflix, HBO Max)
- Media websites (e.g. Instagram, Pinterest)
- Online shops (e.g. Amazon, Ebay)
- Instant messaging (e.g. WhatsApp, Telegram)
- Discussion boards (e.g. Reddit, Stackoverflow)
- Blog websites (e.g. Blogger, Medium)

Big applications may feature multi-purpose content (e.g. blogging functionality on LinkedIn)

Sample Archetype: Blog

Sample Archetype: Blog

Sample Archetype: Blog

- Static content written by an author and published to the web
 - Content can be stored on a CDN

Sample Archetype: Blog

- Static content written by an author and published to the web
 - Content can be stored on a CDN
- Possibility for commenting / reacting to blog posts / subscribing
 - Some dynamic functionality needed

Client-Side Development (Continued)

Client-Side Development

- Key points from Lecture 2:
 - Most client-side frameworks are component-based
 - Client-side applications compile into a bundle that can be deployed online (also into a CDN)
 - A plethora of frameworks (and new ones popping up every now and then)

Previously very briefly looked
into Svelte, let's peek at Astro

<https://astro.build/>

Demo: a Blog with Astro

Demo: a Blog with Astro

Create project template:
`npm create astro@latest`
→ choose "a personal website starter kit"

Demo: a Blog with Astro

Create project template:

```
npm create astro@latest
```

→ choose "a personal website starter kit"

Add Svelte support:

```
npx astro add svelte
```

Demo: a Blog with Astro

Create project template:

`npm create astro@latest`

→ choose "a personal website starter kit"

Add Svelte support:

`npx astro add svelte`

Some magic when using a
component - client:only

Demo: a Blog with Astro

Create project template:

```
npm create astro@latest
```

→ choose "a personal website starter kit"

Add Svelte support:

```
npx astro add svelte
```

Some magic when using a
component - client:only

Build static site:

```
npm run build
```

→ now site in folder "dist"

Measuring Performance

Measuring Performance

Measuring Performance

- In the first course project, we learned to measure time it takes to retrieve a resource, contrasting frameworks and programming languages.

Measuring Performance

- In the first course project, we learned to measure time it takes to retrieve a resource, contrasting frameworks and programming languages.
 - Considerable differences between some frameworks within the same programming language, smaller differences between some other.
 - Considerable differences between some programming languages, smaller differences between some other.
 - Endpoints that use a database typically perform less well when compared to endpoints that simply serve static content.

Measuring Performance

- In the first course project, we learned to measure time it takes to retrieve a resource, contrasting frameworks and programming languages.
 - Considerable differences between some frameworks within the same programming language, smaller differences between some other.
 - Considerable differences between some programming languages, smaller differences between some other.
 - Endpoints that use a database typically perform less well when compared to endpoints that simply serve static content.
- The project measured time it takes to retrieve a resource. This may not, however, reflect the time that showing a resource to the user would take.

Measuring Performance

- In the first course project, we learned to measure time it takes to retrieve a resource, contrasting frameworks and programming languages.
 - Considerable differences between some frameworks within the same programming language, smaller differences between some other.
 - Considerable differences between some programming languages, smaller differences between some other.
 - Endpoints that use a database typically perform less well when compared to endpoints that simply serve static content.
- The project measured time it takes to retrieve a resource. This may not, however, reflect the time that showing a resource to the user would take.
 - E.g. Resource with links to other resources that are retrieved before content is shown.

Measuring Performance

Measuring Performance

- Core Web Vitals

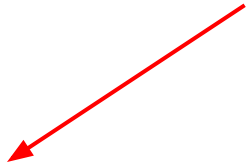
Measuring Performance

- Core Web Vitals
 - Largest Contentful Paint
 - First Input Delay
 - Cumulative Layout Shift

Measuring Performance

- Core Web Vitals
 - Largest Contentful Paint
 - First Input Delay
 - Cumulative Layout Shift

The time it takes for the **largest** element of a page (e.g. an image, paragraph) to be rendered (shown) to the user.

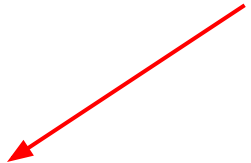


Measuring Performance


- Core Web Vitals

- Largest Contentful Paint
- First Input Delay
- Cumulative Layout Shift

The time it takes for the **largest** element of a page (e.g. an image, paragraph) to be rendered (shown) to the user.



The time it takes for the application to react to an input event from the user (e.g. clicking on a link or a button, ...).



Measuring Performance

- Core Web Vitals

- Largest Contentful Paint
- First Input Delay
- Cumulative Layout Shift

The time it takes for the **largest** element of a page (e.g. an image, paragraph) to be rendered (shown) to the user.

The time it takes for the application to react to an input event from the user (e.g. clicking on a link or a button, ...).

A value describing stability of the user interface - used in efforts to minimize unnecessary moving of layout components.

Measuring Performance

- Core Web Vitals

- Largest Contentful Paint
- First Input Delay
- Cumulative Layout Shift

The time it takes for the **largest** element of a page (e.g. an image, paragraph) to be rendered (shown) to the user.

The time it takes for the application to react to an input event from the user (e.g. clicking on a link or a button, ...).

A value describing stability of the user interface - used in efforts to minimize unnecessary moving of layout components.

- Also a part of Google Lighthouse performance scoring

Demo: Google Lighthouse

Note! Run in incognito mode to avoid influence from browser plugins.

Demo: Performance Insights

Second Course Project

Second Course Project

- In the second course project, your task is to create a Jamstack-like web application used for practicing programming.
- The web application should feature:
 - A main page with a list of programming exercises (we'll provide these). Opening an exercise shows a handout, a textarea into which a solution (code) can be written, and a button that can be used to submit the solution for grading.
 - Randomly created user token on opening the application for the first time. The user token is stored in localStorage and is used to identify the user in the future.
 - Grading of submissions using a Docker image (we'll provide the image). When a solution has been submitted for grading, the user is shown the result once the grading has finished.
 - A database for storing user-specific submissions and grading results.
 - Handling submission peaks consisting of thousands of code submissions within a minute by storing submissions into a queue that is processed whenever resources are available.
 - A way visually distinguish user's completed exercises from non-completed exercises.

Second Course Project

- In the second course project, your task is to create a Jamstack-like web application used for practicing programming.
- The web application should feature:
 - A main page with a list of programming exercises (we'll provide these). Opening an exercise shows a handout, a textarea into which a solution (code) can be written, and a button that can be used to submit the solution for grading.
 - Randomly created user token on opening the application for the first time. The user token is stored in localStorage and is used to identify the user in the future.
 - Grading of submissions using a Docker image (we'll provide the image). When a solution has been submitted for grading, the user is shown the result once the grading has finished.
 - A database for storing user-specific submissions and grading results.
 - Handling submission peaks consisting of thousands of code submissions within a minute by storing submissions into a queue that is processed whenever resources are available.
 - A way visually distinguish user's completed exercises from non-completed exercises.

Possible user flow:

1. User opens the application
 - a. If the user has previously opened the application, token is retrieved from localStorage
 - b. If the user has not previously opened the application, a token is created and stored to localStorage
2. User is shown a list of programming exercises
3. User clicks on a programming exercise
4. User is shown the handout for the exercise, a textarea for writing a solution, and a button for submitting the solution
5. User types in a solution and submits it
6. User sees solution correctness (can take a while)
7. User navigates back to main page (goto 2)

Second Course Project - Passing Requirements

- A working Jamstack-like implementation returned in a format that allows running it easily locally on Windows, Linux and Mac (i.e. a docker-compose configuration or similar for running the application).
 - Recommended: Separate docker services for client and server. Can have more services (and should have as e.g. a database is needed).
- Core Web Vitals tests for the application (e.g. using Google Lighthouse).
- Performance tests (e.g. with K6) for the main page and the API endpoint used for submitting exercises. In the tests, record the average requests per second and the median, 95th percentile, and 99th percentile HTTP request duration. Run the tests with a sensible number of concurrent users for 10 seconds.
- Lighthouse Performance score of at least 70/100 for the main page and the exercise page.
- Summary report.

We will provide a starter template for the project, including a Docker image used for "grading" programming exercises.

Second Course Project - Passing Requirements / Report

- A markdown-formatted document (no binary content) with:
 - Brief guidelines for running the application (and performance tests if they have been ran with scripts).
 - Core web vitals and performance test results.
 - A brief reflection (5-10 sentences) on the present performance of the application.
 - A brief list of suggestions (5-10 sentences) for improving the performance of the application.

We will provide a starter template for the project, including a Docker image used for "grading" programming exercises.

Second Course Project - Passing With Merits

- In addition to fulfilling the passing requirements:
 - The exercise list on the main page shows which exercises the user has completed.
 - The main page lists always at most three non-completed exercises (and all completed exercises).
 - The application features a cache of exercise submissions and the corresponding grading results. The cache is used to avoid unnecessary grading of submissions that match submitted codes already present in the cache (you can use exact string matching when checking whether a code is already in the cache – do account for different exercises!).
 - Lighthouse Performance score at least 80/100 for the main page and the exercise page.

We will provide a starter template for the project, including a Docker image used for "grading" programming exercises.