Study diary

Contents

Week 3 - Planning, designing

Suunnittelu

[TOP Back to Contents]

For planning freelance platform were created features Project features (draft) :

- user authentication and profiles,
- Gig Creation and Management,
- Rating and Review System,
- Basic Search and Filters,
- Auditlogs .

Freelance-alustan suunnittelua varten luotiin seuraavat ominaisuudet: Project features (draft) :

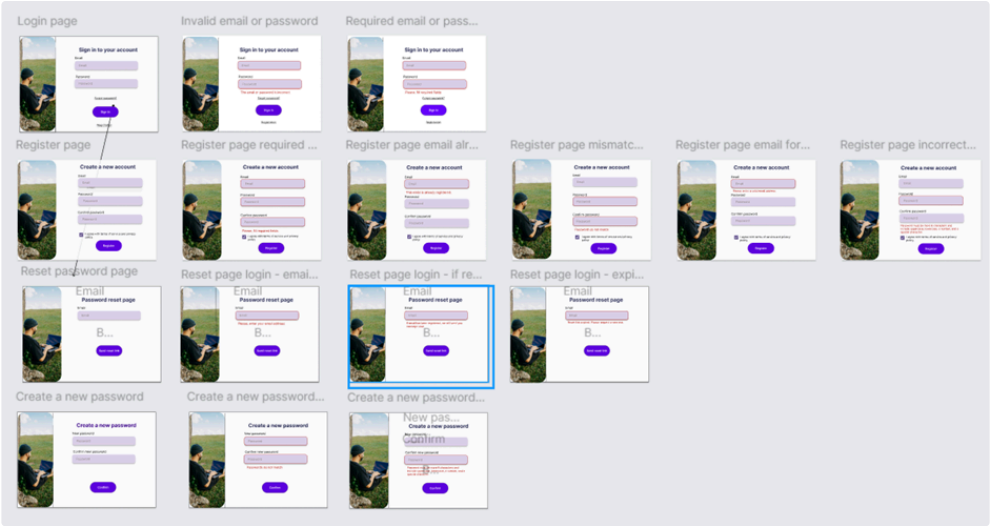
- käyttäjien todennus ja profiilit
- keikkojen luominen ja hallinta
- arviointi- ja arvostelujärjestelmä
- perushaku ja suodattimet
- lokitiedot (auditlogs).

Stack: Project Technology Overview .

Created specific tasks. e.g. - Divided reset password task to smaller tasks. KEKO24-17: Password Recovery Process TO DO and KEKO24-33: Password update form TO DO
Updated design in **full app** page for login and register features in figma: https://www.figma.com/design/LWH8F4qcrSCbaEAT6/DOVDY/Project-wireframe?node-id=927-4811 Connect your Figma account We've done 3 user stories about login and registration process: User Story for login

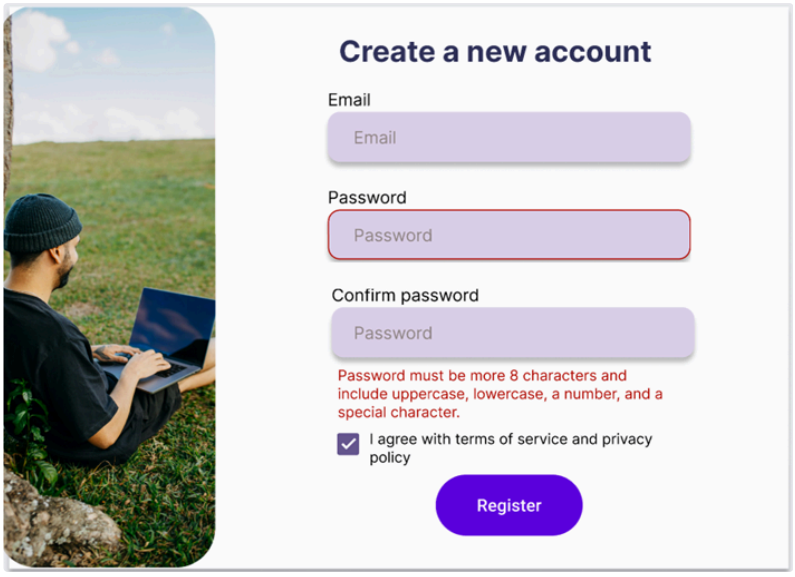
The team's work was to create a prototype and user scenarios for the page: registration form, login form, password recovery and creating new password form. 16 user cases were created for the landing page.

Tiimin tehtävänä oli luoda prototyyppi ja käyttäjäskenaariot seuraaville sivuille: rekisteröitymislomake, kirjautumislomake, salasanan palautus- ja uuden salasanan luomislomake. Landing-sivulle luotiin 16 käyttäjätapausta.



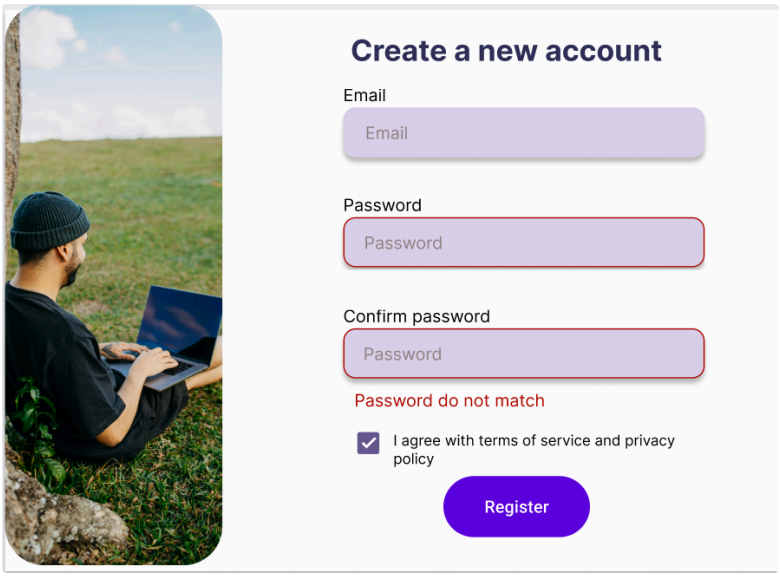
Examples of user cases.
Scenario for an error when a user enters a password that does not meet the requirements of the input field on the account creation page.

Esimerkkejä käyttäjätapauksista:
Skenaario virheestä, kun käyttäjä syöttää salasanan, joka ei täytä vaadittuja ehtoja tilinluontisivulla.



Example of designing a scenario when a user confirm a password and passwords do not match.

Esimerkki skenaariosta, jossa käyttäjä vahvistaa salasanan, mutta salasanat eivät täsmää.



Week 4 - Set up the environment

Ympäristön käyttöönotto

[TOP Back to Contents]

1. Created task for button component KEKO24-37: Button component DONE .
2. Integrated Jira, Bitbucket into VSCode. Cloned project from Bitbucet to my VSCode.

Generated SSH key:
`ssh-keygen -t ed25519 -C "your_email@address.here"`

3. Updated node and npm version.

`nvm install lts/jod`

`nvm use lts/jod`

`node -v` this will check node version should be 22.13.0

`npm update -g npm` This will update npm version

`npm -v` This will check npm version which should be 11.0.0

4. Created brunch for development button component. gigflow: feature/KEKO24-37-button-component

5 Commited and pushed first initial commit: gigflow: 0ee92525 Initialize button component

for commiting in Git set a name and a email:

`git config --global user.name "Your Name"`

`git config --global user.email "youremail@yourdomain.com"`

Week 5 - UI Developing

Käyttöliittymän kehitys

[TOP Back to Contents]

1. Here are the official documentation for testing, stories, react and typescript.

- For unit testing (icon.spec.tsx): Getting Started | Guide | Vitest

- For storybook stories (icon.stories.tsx): Get started with Storybook | Storybook docs
- For react (icon.tsx): React Reference Overview – React
- For typescript: TypeScript: JavaScript With Syntax For Types.
- We are using Radix UI components: Radix UI (Use Themes not Primitives)
- We are using Tailwind: Tailwind CSS - Rapidly build modern websites without ever leaving your HTML.
- Radix themes + tailwind: Styling – Radix Themes
- Extra tutorials: Styling a Radix Dialog with Tailwind CSS (Sam Selikoff from Youtube)

2. Installed Storybook `npx storybook@latest init`
3. Installed vitest `npm install -D vitest` and Vitest Runner in VSCode
4. Installed radix themes

```
npm install @radix-ui/themes
```

5. Created files: button.tsx, button.spec.tsx, button.stories.tsx. Modified button.stories.tsx file according figma design Guide and buttons specification Buttons. Avoided big amount button options, reduced some of them.
6. Made a commit gigflow: ab69eb48 Done
7. Code in files below:

/home/kseniia/git/gigflow/libs/ui-components/src/lib/button/button.tsx

```tsx
import React from 'react';
import { Button as RadixButton, type ButtonProps as RadixButtonProps } from '@radix-ui/themes';

/**
 * Extend the Radix Button props here if you want to add extra fields.
 * Otherwise, you can just re-export RadixButtonProps directly.
 */
export interface ButtonProps extends RadixButtonProps {
  // Example: add your own custom prop
  dataTestId?: string;
}

/**
 * A reusable Button component wrapping Radix UI Themes <Button>.
 */
export const Button = React.forwardRef<HTMLButtonElement, ButtonProps>(
  ({ dataTestId, ...rest }, ref) => {
    return <RadixButton ref={ref} data-testid={dataTestId} {...rest} />;
  }
);

```

```

```

/home/kseniia/git/gigflow/libs/ui-components/src/lib/button/button.stories.tsx

```tsx
/home/kseniia/git/gigflow/libs/ui-components/src/lib/button/button.stories.tsx
// libs/ui-components/src/lib/button/button.stories.tsx
import { Meta, StoryObj } from '@storybook/react';
import { userEvent, within } from '@storybook/testing-library';
import { expect } from '@storybook/jest';

// 1) Import Radix UI's Theme & global styles
import { Theme } from '@radix-ui/themes';
import '@radix-ui/themes/styles.css';
```

```
10
11  import { Button } from './button';
12
13  // Set up the Storybook meta
14  const meta: Meta<typeof Button> = {
15    title: 'Components/Button',
16    component: Button,
17    argTypes: {
18      variant: { options: ['solid', 'soft', 'outline'], control: { type: 'radio' } },
19      highContrast: { control: 'boolean' },
20      radius: { options: ['large', 'full'], control: { type: 'radio' } },
21      size: { options: ['3', '4'], control: { type: 'radio' } },
22      disabled: { control: 'boolean' },
23      onClick: function () {
24        console.log('Button Clicked');
25      },
26    },
27
28    // 2) Decorator: Wrap stories in <Theme>
29    decorators: [
30      Story => (
31        <Theme appearance="light">
32          <Story />
33        </Theme>
34      ),
35    ],
36  };
37  export default meta;
38
39  type Story = StoryObj<typeof Button>;
40
41  // A basic story main large buttons for sign in, register, password reset, new password pages.
42  // For the most important or common action on a screen. Only one primary button per screen.
43  export const SolidButtonPrimary: Story = {
44    args: {
45      children: 'Sign in',
46      variant: 'solid',
47      color: 'violet',
48      radius: 'full',
49      size: '4',
50      disabled: false,
51      onClick: function () {
52        console.log('Button Clicked');
53      },
54    },
55    play: async ({ canvasElement }) => {
56      const canvas = within(canvasElement);
57      const button = canvas.getByRole('button', { name: /Sign in/i });
58      await userEvent.click(button);
59      expect(button).toHaveTextContent('Sign in');
60    },
61  };
62
63  // Specific buttons are less visually prominent than primary buttons, e.g, for header buttons.
64  export const SoftButtonSecondary: Story = {
65    args: {
66      children: 'Join',
67      variant: 'soft',
```

```
    color: 'violet',
    size: '3',
    radius: 'large',
    disabled: false,
    highContrast: true,
    onClick: function () {
      console.log('Button Clicked');
    },
  },

  play: async ({ canvasElement }) => {
    const canvas = within(canvasElement);
    const button = canvas.getByRole('button', { name: /Join/i });
    await userEvent.click(button);
    expect(button).toHaveTextContent('Join');
  },
};

// Important actions that are less visually prominent than primary buttons/
// For add, create, apply, save, confirm buttons.
export const SolidButtonSecondary: Story = {
  args: {
    children: 'Create',
    variant: 'solid',
    color: 'violet',
    size: '3',
    radius: 'large',
    disabled: false,
    onClick: function () {
      console.log('Button Clicked');
    },
  },
  play: async ({ canvasElement }) => {
    const canvas = within(canvasElement);
    const button = canvas.getByRole('button', { name: /Create/i });
    await userEvent.click(button);
    expect(button).toHaveTextContent('Create');
  },
};

// Secondary outline buttons for less actions, requiring attention:
// back, cancel, load more, view all.
export const OutlinedButtonSecondary: Story = {
  args: {
    children: 'Cancel',
    variant: 'outline',
    color: 'violet',
    size: '3',
    radius: 'large',
    disabled: false,
    onClick: function () {
      console.log('Button Clicked');
    },
  },
  play: async ({ canvasElement }) => {
    const canvas = within(canvasElement);
    const button = canvas.getByRole('button', { name: /Cancel/i });
    await userEvent.click(button);
```

```
    expect(button).toHaveTextContent('Cancel');
  },
};

// For actions that may have destructive effects on user data.
// For delete, remove buttons.
export const DangerButton: Story = {
  args: {
    children: 'Delete',
    variant: 'solid',
    color: 'red',
    size: '3',
    radius: 'large',
    highContrast: false,
    disabled: false,
    onClick: function () {
      console.log('Button Clicked');
    },
  },
  play: async ({ canvasElement }) => {
    const canvas = within(canvasElement);
    const button = canvas.getByRole('button', { name: /Delete/i });
    await userEvent.click(button);
    expect(button).toHaveTextContent('Delete');
  },
};

// For buttons, calling to action: Leave a review, Update picture.
// Yellow shades can be used for backlighting and rating.
export const YellowButton: Story = {
  args: {
    children: 'Leave a review!',
    variant: 'solid',
    color: 'yellow',
    radius: 'large',
    size: '4',
    disabled: false,
    onClick: function () {
      console.log('Button Clicked');
    },
  },
  play: async ({ canvasElement }) => {
    const canvas = within(canvasElement);
    const button = canvas.getByRole('button', { name: /Leave a review!/i });
    await userEvent.click(button);
    expect(button).toHaveTextContent('Leave a review!');
  },
};

```

Button stories created in storybook:

Primary button

Join

Soft secondary

Create

Solid secondary

Cancel

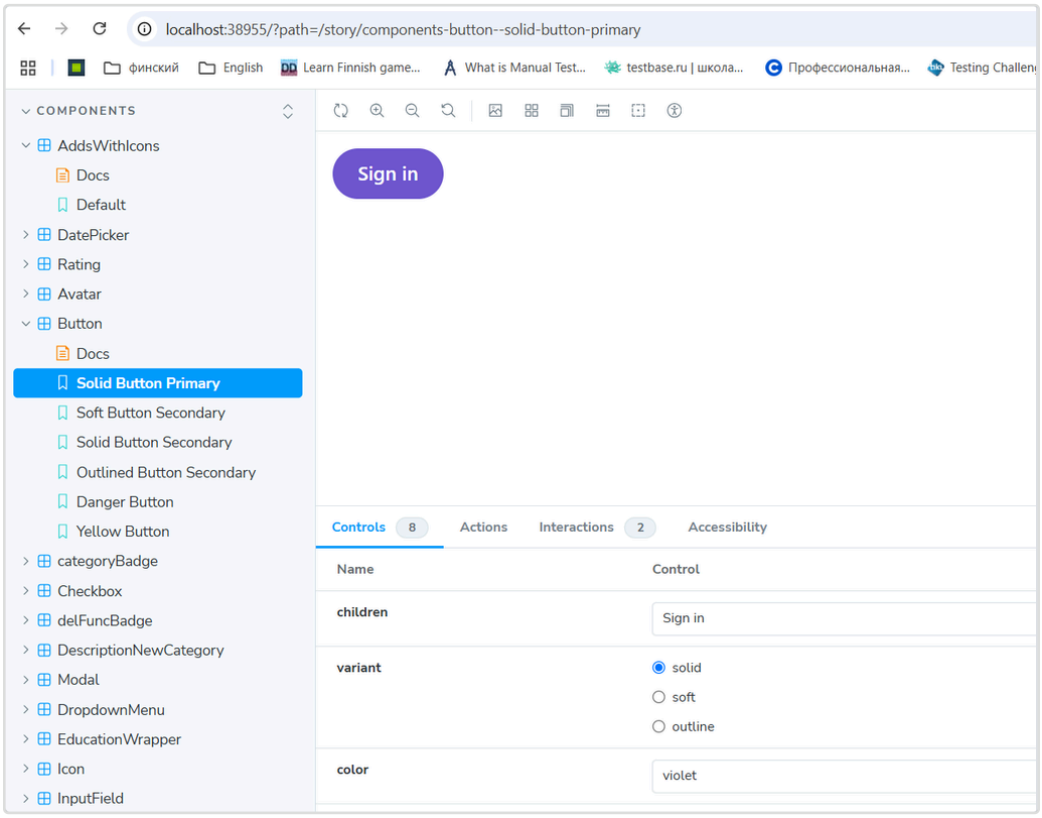
Outlined

Leave a review!

Yellow button

Delete

Danger button



Storybook preview

Week 6 - 7 - UI Testing

UI-testaus

[TOP Back to Contents]

1. Created different unit tests in button.spec.tsx using libraries vitest, react, jest-dom:

/home/kseniia/git/gigflow/libs/ui-components/src/lib/button/button.spec.tsx

```tsx
import { render, screen } from '@testing-library/react';
import { describe, it, expect } from 'vitest';
import { Button } from './button';
import '@testing-library/jest-dom';
import userEvent from '@testing-library/user-event';

// General button tests
describe('General button tests', () => {

  // Button renders succesfully
  it('renders successfully', () => {
    render(<Button>Click me!</Button>);
    expect(screen.getByText('Click me!')).toBeInTheDocument();
  });

  // Button can be disabled
  it('can be disabled', () => {
    render(<Button disabled>Cannot click me!</Button>);
    screen.debug();
    expect(screen.getByRole('button')).toBeDisabled();
  });

  // Button calls onClick when clicked
  it('calls onClick when clicked', async () => {
    const handleClick = vi.fn(); // Mock-function
    render(<Button onClick={handleClick}>Click me!</Button>);
    await userEvent.click(screen.getByText('Click me!'));
    expect(handleClick).toHaveBeenCalledTimes(1); // Check, that onClick called one time
  });
});



// Styles tests
describe('Button variant styles', () => {
  // Button can have Solid style
  it('applies solid variant', () => {
    render(<Button variant="solid">Solid</Button>);
    expect(screen.getByText('Solid')).toHaveClass('rt-variant-solid');
  });

  // Button can have Soft style
  it('applies soft variant', () => {
    render(<Button variant="soft">Soft</Button>);
    expect(screen.getByText('Soft')).toHaveClass('rt-variant-soft');
  });

  // Button can have Outline style
  it('applies outline variant', () => {
    render(<Button variant="outline">Outline</Button>);
    expect(screen.getByText('Outline')).toHaveClass('rt-variant-outline');
  });
});

```

```
55
56
57  // Colour tests
58  describe('Button colors', () => {
59
60    // Sign in button has violet color
61    it('applies violet color', () => {
62      render(<Button color="violet">Sign in</Button>);
63      expect(screen.getByText('Sign in')).toHaveAttribute('data-accent-color', 'violet');
64    });
65
66    // Danger button has red color skipped why???
67    it('applies red color (danger button)', () => {
68      render(<Button color="red">Delete</Button>);
69      expect(screen.getByText('Delete')).toHaveAttribute('data-accent-color', 'red');
70    });
71
72    // Review button has yellow color
73    it('applies yellow color', () => {
74      render(<Button color="yellow">Review</Button>);
75      expect(screen.getByText('Review')).toHaveAttribute('data-accent-color', 'yellow');
76    });
77  });
78
79
80
81  // Sizes tests
82  describe('Button sizes', () => {
83
84    // Button has size 3
85    it('applies size 3', () => {
86      render(<Button size="3">Size 3</Button>);
87      expect(screen.getByText('Size 3')).toHaveClass('rt-r-size-3');
88    });
89
90    // Button has size 4
91    it('applies size 4', () => {
92      render(<Button size="4">Size 4</Button>);
93      expect(screen.getByText('Size 4')).toHaveClass('rt-r-size-4');
94    });
95  });
96
97
98
99  // Radius tests
100 describe('Button radius styles', () => {
101
102   // Button has radius large
103   it('applies large radius', () => {
104     render(<Button radius="large">Large Radius</Button>);
105     expect(screen.getByText('Large Radius')).toHaveAttribute('data-radius', 'large');
106   });
107
108   // Button has radius full
109   it('applies full radius', () => {
110     render(<Button radius="full">Full Radius</Button>);
111     expect(screen.getByText('Full Radius')).toHaveAttribute('data-radius', 'full');
112   });
```

```
});



// Solid button primary test
describe('Solid button primary test', () => {

  // Solid button primary has its properties
  it('Solid button primary', () => {
    render(
    <Button color='violet' variant='solid' radius='full' size={'4'} disabled={false}
dataTestId='solidbuttonprimary'>
      Sign in
    </Button>);
    screen.debug();
    expect(screen.getByTestId('solidbuttonprimary')).toBeInTheDocument();
    expect(screen.getByTestId('solidbuttonprimary')).toHaveAttribute('data-accent-color', 'violet');
    expect(screen.getByTestId('solidbuttonprimary')).toHaveClass('rt-variant-solid');
    expect(screen.getByTestId('solidbuttonprimary')).toHaveAttribute('data-radius', 'full');
    expect(screen.getByTestId('solidbuttonprimary')).toHaveTextContent('Sign in');
  });
});



// Soft button secondary test
describe('Soft button secondary test', () => {

  // Soft button primary has its properties
  it('Soft button secondary', () => {
    render(
    <Button color='violet' variant='soft' radius='large' size={'3'} disabled={false} highContrast={true}
dataTestId='softbuttonsecondary'>
      Join
    </Button>);
    screen.debug();
    expect(screen.getByTestId('softbuttonsecondary')).toBeInTheDocument();
    expect(screen.getByTestId('softbuttonsecondary')).toHaveAttribute('data-accent-color', 'violet');
    expect(screen.getByTestId('softbuttonsecondary')).toHaveClass('rt-variant-soft');
    expect(screen.getByTestId('softbuttonsecondary')).toHaveClass('rt-high-contrast');
    expect(screen.getByTestId('softbuttonsecondary')).toHaveAttribute('data-radius', 'large');
    expect(screen.getByTestId('softbuttonsecondary')).toHaveTextContent('Join');
  });
});



// Solid button secondary test
describe('Solid button secondary test', () => {

  // Solid button secondary has its properties
  it('Solid button secondary', () => {
    render(
    <Button color='violet' variant='solid' radius='large' size={'3'} disabled={false}
dataTestId='solidbuttonsecondary'>
      Create
    </Button>);
    screen.debug();
```

```
    expect(screen.getByTestId('solidbuttonsecondary')).toBeInTheDocument();
    expect(screen.getByTestId('solidbuttonsecondary')).toHaveAttribute('data-accent-color', 'violet');
    expect(screen.getByTestId('solidbuttonsecondary')).toHaveClass('rt-variant-solid');
    expect(screen.getByTestId('solidbuttonsecondary')).toHaveAttribute('data-radius', 'large');
    expect(screen.getByTestId('solidbuttonsecondary')).toHaveTextContent('Create');
  });
});



// Outline button secondary test
describe('Outline button secondary test', () => {

  // Outline button secondary has its properties
  it('Outline button secondary', () => {
    render(
    <Button color='violet' variant='outline' radius='large' size={'3'} disabled={false}
dataTestId='outlinebuttonsecondary'>
      Cancel
    </Button>);
    screen.debug();
    expect(screen.getByTestId('outlinebuttonsecondary')).toBeInTheDocument();
    expect(screen.getByTestId('outlinebuttonsecondary')).toHaveAttribute('data-accent-color', 'violet');
    expect(screen.getByTestId('outlinebuttonsecondary')).toHaveClass('rt-variant-outline');
    expect(screen.getByTestId('outlinebuttonsecondary')).toHaveAttribute('data-radius', 'large');
    expect(screen.getByTestId('outlinebuttonsecondary')).toHaveTextContent('Cancel');
  });
});



// Danger button test
describe('Danger button test', () => {

  // Danger button has its properties
  it('Danger button', () => {
    render(
    <Button color='red' variant='solid' radius='large' size={'3'} disabled={false}
dataTestId='dangerbutton'>
      Delete
    </Button>);
    screen.debug();
    expect(screen.getByTestId('dangerbutton')).toBeInTheDocument();
    expect(screen.getByTestId('dangerbutton')).toHaveAttribute('data-accent-color', 'red');
    expect(screen.getByTestId('dangerbutton')).toHaveClass('rt-variant-solid');
    expect(screen.getByTestId('dangerbutton')).toHaveAttribute('data-radius', 'large');
    expect(screen.getByTestId('dangerbutton')).toHaveTextContent('Delete');
  });
});



// Yellow button test
describe('Yellow button test', () => {

  // Yellow button has its properties
  it('Yellow button', () => {
    render(
```

```
224     <Button color='yellow' variant='solid' radius='large' size={'4'} disabled={false}
    dataTestId='yellowbutton'>
225       Leave a review
226     </Button>);
227     screen.debug();
228     expect(screen.getByTestId('yellowbutton')).toBeInTheDocument();
229     expect(screen.getByTestId('yellowbutton')).toHaveAttribute('data-accent-color', 'yellow');
230     expect(screen.getByTestId('yellowbutton')).toHaveClass('rt-variant-solid');
231     expect(screen.getByTestId('yellowbutton')).toHaveAttribute('data-radius', 'large');
232     expect(screen.getByTestId('yellowbutton')).toHaveTextContent('L');
233   });
234 });
```

These tests verify the behavior, appearance, and properties of a custom `Button` component. They include:

- **General functionality tests**: checking if the button renders correctly, handles `onClick` events, and supports the `disabled` state.
- **Style variant tests**: ensuring the correct CSS classes are applied for `solid`, `soft`, and `outline` variants.
- **Color tests**: confirming that different colors (e.g. `violet`, `red`, `yellow`) correctly set the `data-accent-color` attribute.
- **Size and radius tests**: verifying the button applies the correct size and border radius settings.
- **Combined style tests**: testing specific button configurations (e.g. primary, secondary, danger buttons) to ensure all visual and functional properties are applied as expected.

Nämä testit tarkistavat mukautetun Button-komponentin toiminnan, ulkoasun ja ominaisuudet. Ne sisältävät seuraavat osa-alueet:

- **Yleiset toiminnallisuustestit**: tarkistetaan, että painike renderöityy oikein, käsittelee onClick-tapahtumat ja tukee disabled-tilaa.
- **Tyylivarianttien testaus**: varmistetaan, että oikeat CSS-luokat käytetään eri tyylivaihtoehdoille, kuten solid, soft ja outline.
- **Väritestit**: tarkistetaan, että eri värit (esim. violetti, punainen, keltainen) asettavat `data-accent-color` -attribuutin oikein.
- **Koko- ja kulmatestit**: varmennetaan, että painike käyttää oikeita koko- ja reunapyöristyksen asetuksia.
- **Yhdistetyt tyylitestit**: testataan tiettyjä painikekonfiguraatioita (esim. primary, secondary, danger) ja varmistetaan, että kaikki visuaaliset ja toiminnalliset ominaisuudet toimivat odotetusti.

2. Running tests with `npx vitest run libs/ui-components/src/lib/button/button.spec.tsx`

```
✓ @gigflow/ui-components src/lib/button/button.spec.tsx (19 tests) 165ms
  ✓ General button tests > renders successfully 31ms
  ✓ General button tests > can be disabled 44ms
  ✓ General button tests > calls onClick when clicked 24ms
  ✓ Button variant styles > applies solid variant 3ms
  ✓ Button variant styles > applies soft variant 4ms
  ✓ Button variant styles > applies outline variant 2ms
  ✓ Button colors > applies violet color 3ms
  ✓ Button colors > applies red color (danger button) 3ms
  ✓ Button colors > applies yellow color 6ms
  ✓ Button sizes > applies size 3 3ms
  ✓ Button sizes > applies size 4 2ms
  ✓ Button radius styles > applies large radius 2ms
  ✓ Button radius styles > applies full radius 2ms
  ✓ Solid button primary test > Solid button primary 11ms
  ✓ Soft button secondary test > Soft button secondary 5ms
  ✓ Solid button secondary test > Solid button secondary 4ms
  ✓ Outline button secondary test > Outline button secondary 5ms
  ✓ Danger button test > Danger button 3ms
  ✓ Yellow button test > Yellow button 4ms

Test Files  1 passed (1)
     Tests  19 passed (19)
  Start at  16:22:59
  Duration  2.67s (transform 102ms, setup 0ms, collect 1.19s, tests 165ms, en
```

Screenshot successful passed tests from terminal

3. Made a commit gigflow: cf6fb8f3 Added unit tests in brunch gigflow: feature/KEKO24-37-button-component
4. Documentation - Testing libraries documentation:

[React Testing Library | Testing Library]

[Assertion API | Browser Mode | Vitest]

[GitHub - testing-library/jest-dom: :owl: Custom jest matchers to test the state of the DOM]

[Getting Started | Guide | Vitest]

Flex example from Matti for content (for frontend)

```tsx
import { Flex } from '@radix-ui/themes';

export function SignInForm() {
  return (
    <Flex
      // 1) On small screens, stack everything in a column;
      //    on medium/large, place them side by side (row).
      direction={{ sm: 'column', md: 'row'  }}
      gap="5"
      // Optional: limit total width & center on page
      style={{ maxWidth: '1000px', margin: '0 auto', padding: '1rem' }}
    >
      {/* Left (Profile Picture) */}
      <Flex direction="column" style={{ flex: '1 0 0' }}>
        <div style={{ height: '100%', border: '2px solid black' }}>
          Profile Picture
        </div>
      </Flex>
        {/* Right (Form Fields & Buttons) */}
  <Flex direction="column" gap="5" style={{ flex: '2 0 0' }}>
    {/* Form Fields */}
    <Flex direction="column" gap="3">
      <div style={{ border: '2px solid black' }}>Sign In Your Account</div>
```

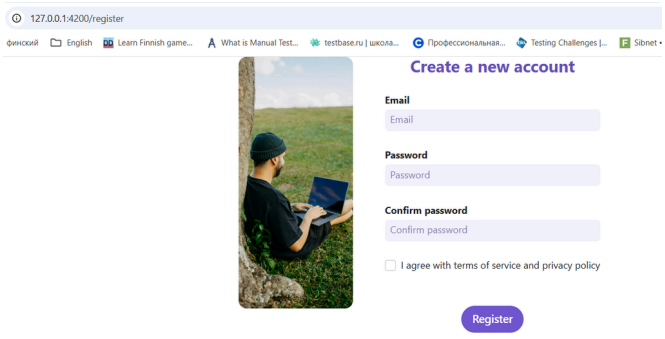
```
      <div style={{ border: '2px solid black' }}>Email</div>
      <input style={{ border: '2px solid black', padding: '0.5rem' }} />
      <div style={{ border: '2px solid black' }}>Password</div>
      <input style={{ border: '2px solid black', padding: '0.5rem' }} />
      <a href="<https://google.com>">Forgot password</a>
    </Flex>

    {/* Buttons */}
    <Flex
      // 2) For buttons, we can make them stack on extra-small screens
      //    but go side by side on small screens and up.
      direction={{ initial: 'column', sm: 'row' }}
      gap="5"
    >
      <button style={{ border: '2px solid black', padding: '0.5rem' }}>
        Cancel
      </button>
      <button style={{ border: '2px solid black', padding: '0.5rem' }}>
        Sign In
      </button>
    </Flex>
  </Flex>
</Flex>
 );
}
```

Week 8-13 - Frontend developing

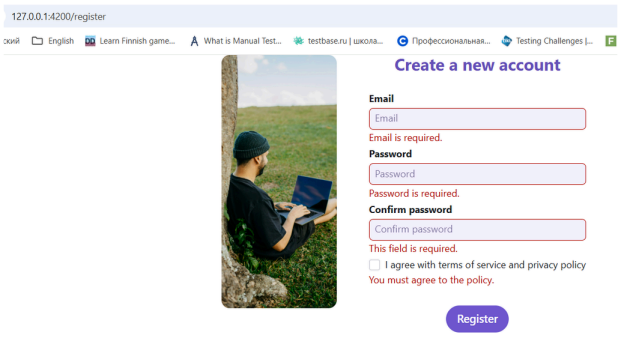
Frontend-kehitys

[TOP Back to Contents]

Created frontend part of registration page with validation scenarios.



Registration page in browser



Registration page - validation scenario if user click button register first

This registration form was built using React and leverages several libraries to handle form functionality, validation, and styling. React Hook Form is used to manage the form's state and interactions, allowing for efficient input registration and validation handling. Zod is integrated as the validation schema language, ensuring that all form fields follow strict rules—such as requiring a valid email format, enforcing a minimum password length, checking if passwords match, and making sure the user accepts the terms of service. The integration between React Hook Form and Zod is done using the `@hookform/resolvers/zod` package.

The visual structure and styling of the form are supported by components from `@radix-ui/themes`, such as layout containers, flexboxes, headings, and text elements. Custom components like `InputTextField`, `Checkbox`, and `Button` are used to render form controls, styled consistently with the overall design system.

Functionally, the form performs real-time validation on blur and change events, but only displays error messages after the field has been interacted with, providing a smoother user experience. On form submission, all fields are validated, and if everything passes, the collected data is logged to the console.

Rekisteröitymissivun frontend-osuus luotiin validointiskenaarioiden kanssa.
Tämä rekisteröitymislomake rakennettiin Reactilla, ja se hyödyntää useita kirjastoja lomakkeen toiminnallisuuden, validoinnin ja tyylien hallintaan. React Hook Form hallinnoi lomakkeen tilaa ja vuorovaikutuksia, mahdollistaen tehokkaan syötteiden rekisteröinnin ja validoinnin käsittelyn.

Zod on integroitu validointiskeemaksi, mikä varmistaa, että kaikki lomakekentät noudattavat tiukkoja sääntöjä — kuten vaadittua sähköpostin muotoa, vähimmäissalasanan pituutta, salasanojen täsmäämistä ja käyttöehtojen hyväksymistä. Integraatio React Hook Formin ja Zodin välillä toteutetaan @hookform/resolvers/zod -paketin avulla.

Lomakkeen visuaalinen rakenne ja tyylit perustuvat @radix-ui/themes-komponentteihin, kuten layout-kontteihin, joustaviin laatikoihin (flexbox), otsikoihin ja tekstielementteihin. Mukautettuja komponentteja, kuten **InputTextField**, **Checkbox** ja **Button**, käytetään lomakekenttien näyttämiseen, ja ne ovat yhdenmukaisia muun suunnittelujärjestelmän kanssa.

Toiminnallisesti lomake suorittaa reaaliaikaista validointia blur- ja change-tapahtumissa, mutta näyttää virheilmoitukset vasta, kun kenttää on käytetty. Tämä tarjoaa sujuvamman käyttökokemuksen. Lomakkeen lähetyksessä kaikki kentät validoidaan, ja jos kaikki on kunnossa, kerätyt tiedot tulostetaan konsoliin.

Code below:

libs/ui-components/src/lib/registration_page/registrationform.tsx

```
import { useState } from 'react';
import { useForm, Controller } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import { z } from 'zod';
import { Flex, Text, Heading, Container } from '@radix-ui/themes';
import { Button, InputTextField, Checkbox } from '../../index';
import image from './assets/login_image.jpg';
```

```
8
9  // Define Zod Schema for Validation
10 const passwordSchema = z
11   .string()
12   .nonempty('Password is required.')
13   .min(8, 'Password must be at least 8 characters.');
14
15 const confirmPasswordSchema = z.string().nonempty('This field is required.');
16
17 export const schema = z
18   .object({
19     email: z.string().nonempty('Email is required.').email('Please, enter a valid email address.'),
20     password: passwordSchema,
21     confirmPassword: confirmPasswordSchema,
22     agree: z.boolean().refine(value => value === true, {
23       message: 'You must agree to the policy.',
24     }),
25   })
26   .refine(data => data.password === data.confirmPassword, {
27     message: 'Passwords do not match.',
28     path: ['confirmPassword'],
29   });
30
31 // 🔹 Define TypeScript Types based on Schema
32 type FormValues = z.infer<typeof schema>;
33
34 export function RegisterForm() {
35   const {
36     register,
37     handleSubmit,
38     control,
39     formState: { errors },
40     trigger,
41   } = useForm<FormValues>({
42     resolver: zodResolver(schema),
43     defaultValues: { agree: false },
44     mode: 'onChange', // Validation if changes
45   });
46
47   // Visited fields
48   const [touchedFields, setTouchedFields] = useState({
49     email: false,
50     password: false,
51     confirmPassword: false,
52     agree: false,
53   });
54
55   const onSubmit = (data: FormValues) => {
56     console.log('Form submitted:', data);
57   };
58
59   // Function for trigger
60   const handleBlur = (fieldName: keyof FormValues) => {
61     setTouchedFields(prev => ({ ...prev, [fieldName]: true }));
62     trigger(fieldName);
63   };
64
65   const handleChange = (fieldName: keyof FormValues) => {
```

```
  if (touchedFields[fieldName]) {
    trigger(fieldName); // Trigger validation if field is changed
  }
};

// Submit handler with full validation
const onFormSubmit = async () => {
  const isValid = await trigger(); // Trigger validation all fields
  if (isValid) {
    handleSubmit(onSubmit)(); // If sucsesfull, call onSubmit
  }
  // Show mistakes for all fields
  setTouchedFields({
    email: true,
    password: true,
    confirmPassword: true,
    agree: true,
  });
};

return (
  <form
    onSubmit={e => {
      e.preventDefault(); // Prevent normal submit
      onFormSubmit(); // Custom handler
    }}
  >
    <Container maxWidth={'600px'}>
      <Flex direction="row" gap="8">
        <div style={{ flexShrink: 0 }}>
          <img
            src={image}
            alt="login illustration"
            width="170px"
            style={{ borderRadius: '12px' }}
          />
        </div>

        <Flex direction="column" gap="3">
          <Heading color="violet" align="center" mb="3">
            Create a new account
          </Heading>

          <Text as="div" size="2" mb="-2" weight="bold">
            Email
          </Text>
          <InputTextField
            {...register('email', {
              onBlur: () => handleBlur('email'),
              onChange: () => handleChange('email'),
            })}
            placeholder="Email"
            color="violet"
            variant="soft"
            className={`border p-2 rounded-md w-full ${
              errors.email && touchedFields.email ? 'border-red-500' : 'border-gray-300'
            }`}
          />
```

```
            <Flex style={{ height: '2px' }}>
              {errors.email && touchedFields.email && (
                <p className="text-red-600" style={{ fontSize: '14px', marginTop: '-10px' }}>
                  {errors.email.message}
                </p>
              )}
            </Flex>

            <Text as="div" size="2" mb="-2" weight="bold">
              Password
            </Text>
            <InputTextField
              {...register('password', {
                onBlur: () => handleBlur('password'),
                onChange: () => handleChange('password'),
              })}
              type="password"
              placeholder="Password"
              color="violet"
              variant="soft"
              className={`border p-2 rounded-md w-full ${
                errors.password && touchedFields.password ? 'border-red-500' : 'border-gray-300'
              }`}
            />

            <Flex style={{ height: '2px' }}>
              {errors.password && touchedFields.password && (
                <p className="text-red-600" style={{ fontSize: '14px', marginTop: '-10px' }}>
                  {errors.password.message}
                </p>
              )}
            </Flex>

            <Text as="div" size="2" mb="-2" weight="bold">
              Confirm password
            </Text>
            <InputTextField
              {...register('confirmPassword', {
                onBlur: () => handleBlur('confirmPassword'),
                onChange: () => handleChange('confirmPassword'),
              })}
              type="password"
              placeholder="Confirm password"
              color="violet"
              variant="soft"
              className={`border p-2 rounded-md w-full ${
                errors.confirmPassword && touchedFields.confirmPassword
                  ? 'border-red-500'
                  : 'border-gray-300'
              }`}
            />

            <Flex style={{ height: '2px' }}>
              {errors.confirmPassword && touchedFields.confirmPassword && (
                <p className="text-red-600" style={{ fontSize: '14px', marginTop: '-10px' }}>
                  {errors.confirmPassword.message}
                </p>
```

```
              )}
            </Flex>

            {/* Connect Checkbox with React Hook Form through Controller */}
            <Controller
              name="agree"
              control={control}
              render={({ field }) => (
                <Text as="label" size="2">
                  <Flex gap="2">
                    <Checkbox
                      checked={field.value}
                      onCheckedChange={checked => {
                        field.onChange(checked);
                        setTouchedFields(prev => ({ ...prev, agree: true }));
                        trigger('agree');
                      }}
                      color="violet"
                    />
                    I agree with terms of service and privacy policy
                  </Flex>
                </Text>
              )}
            />

            <Flex style={{ height: '2px' }}>
              {errors.agree && touchedFields.agree && (
                <p className="text-red-600" style={{ fontSize: '14px', marginTop: '-10px' }}>
                  {errors.agree.message}
                </p>
              )}
            </Flex>

            <Flex gap="3" mt="5" justify="center">
              <Button color="violet" variant="solid" radius="full" size="3" type="submit">
                Register
              </Button>
            </Flex>
          </Flex>
        </Flex>
      </Container>
    </form>
  );
}
```

Merged brunch gigflow: feature/KEKO24-37-button-component into main brunch. Debugging.

Created new task KEKO24-58: Registration page DONE . Created new brunch https://bitbucket.org/kekory/%7B0cf51169-0224-4ea2-8380-7ef6d16dee62%7D/branch/feature/KEKO24-58-registration-page Can't find link Created files registrationform.tsx, registrationform.stories.tsx, registrationform.spec.tsx. Commits: gigflow: 13bff0d5 registration in progress

Week 15-18 - Backend developing

Taustajärjestelmän testaus

[TOP Back to Contents]

In this system, the following functionalities are implemented: automatic logging of all HTTP requests and responses, tracking of user actions (such as CREATE, UPDATE, DELETE), recording of request metadata (HTTP method, path, IP address, user agent), and masking of sensitive data (e.g., passwords, payment details). The audit log also captures user context, including user ID, email, and role, and records which entities and records were affected by each action. The system supports extraction of entity names from request paths, handles nested routes and API versioning, and logs errors with corresponding messages and status codes. For data management, the audit log provides pagination, filtering by date, user, action type, and status, as well as sorting capabilities.

Tässä järjestelmässä on toteutettu seuraavat toiminnot: kaikkien HTTP-pyyntöjen ja -vastausten automaattinen kirjaaminen, käyttäjätoimintojen seuranta (kuten LUO, PÄIVITÄ, POISTA), pyyntöjen metatietojen tallennus (HTTP-metodi, polku, IP-osoite, käyttäjäagentti) sekä arkaluontoisten tietojen, kuten salasanojen ja maksutietojen, peittäminen.

Audit-loki tallentaa myös käyttäjän kontekstin, mukaan lukien käyttäjätunnus, sähköpostiosoite ja rooli, ja kirjaa, mitkä entiteetit ja tietueet muuttuivat kunkin toiminnon seurauksena. Järjestelmä tukee entiteettien nimien poimimista pyyntöpoluista, käsittelee sisäkkäiset reitit ja API-versioinnin, sekä kirjaa virheet niiden viestien ja tilakoodeiden kanssa.

Tietojen hallintaa varten audit-loki tarjoaa sivutuksen, suodatuksen päivämäärän, käyttäjän, toimintatyypin ja tilan mukaan sekä lajittelutoiminnot.

The following libraries and frameworks were used in the implementation:

- **NestJS** (`@nestjs/common`, `@nestjs/core`, `@nestjs/typeorm`) serves as the main application framework, providing modular structure and dependency injection.
- **TypeORM** is used for database operations, entity definition, and repository pattern implementation.
- **RxJS** is utilized for asynchronous request handling and side effects management via Observables and operators like `tap`.
- **class-validator** and **class-transformer** are applied for data validation and transformation.
- **Express** is used as the HTTP server layer for request and response handling.

The database schema is based on an `AuditLog` entity, which includes fields for unique identification, timestamps, user references, action types, entity references, status, and descriptions.

TypeORM is an ORM library for TypeScript and JavaScript (Node.js) that helps you work with databases conveniently. ORM stands for Object-Relational Mapping — that is, TypeORM allows you to work with a database as with ordinary objects, rather than manually writing a bunch of SQL queries.

Jira task: KEKO24-87: Audit Logging DONE

Run docker: `docker-compose up -d`

TypeORM will create a table `AuditLog` with columns `id`, `createdBy`, `createdAt`, `action`, `affectedId`, `affectedTable`, `status`, `description`.

/home/kseniia/git/gigflow/backend/src/audit-log/entities/audit-log.entity.ts

```
import { Entity, PrimaryGeneratedColumn, Column, CreateDateColumn, Index } from 'typeorm';

export enum ActionType {
  CREATE = 'CREATE',
  UPDATE = 'UPDATE',
  DELETE = 'DELETE',
  // Add other action types as needed
}

@Entity()
@Index(['affectedId', 'affectedTable'])
@Index(['createdBy'])
export class AuditLog {
  @PrimaryGeneratedColumn()
```

```
  id!: number;

  @Column({ type: 'int', nullable: true })
  createdBy!: number | null; // User ID of the performer

  @CreateDateColumn({ type: 'timestamp' })
  createdAt!: Date;

  @Column({
    type: 'enum',
    enum: ActionType,
    default: ActionType.CREATE,
  })
  action!: ActionType;

  @Column({ type: 'int', nullable: true }) // Make the column nullable in the database
  affectedId?: number | null; // ID of the affected entity, now optional

  @Column()
  affectedTable!: string; // Name of the affected table/entity

  @Column()
  status!: string; // Could be 'SUCCESS', 'FAILED', etc.

  @Column('text')
  description!: string;
}

```

A service with two main methods:

createAuditLog() - creates a new audit record
getAuditLogs() - gets audit records with filtering and pagination.

Interface AuditLogFilter describes filters that can be applied when you want to get logs. For example: how much to skip, how much to take, by which user (userId), by date, etc.

/home/kseniia/git/gigflow/backend/src/audit-log/audit-log.service.ts

```
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { AuditLog, ActionType } from './entities/audit-log.entity';
import { CreateAuditLogDto } from './dto/create-audit-log.dto';

interface AuditLogFilter {
  skip?: number;
  take?: number;
  startDate?: Date;
  endDate?: Date;
  userId?: number;
  actionType?: ActionType;
  status?: string;
}

@Injectable()
export class AuditLogService {
  constructor(
    @InjectRepository(AuditLog)
```

```
    private readonly auditLogRepository: Repository<AuditLog>,
  ) {}

  /**
   * Creates an audit log entry for a request or action.
   * Supports AC1 (log request details), AC2 (user context), AC4 (secure persistence), AC6 (expects pre-masked data).
   */
  async createAuditLog(dto: CreateAuditLogDto): Promise<void> {
    const auditLog = this.auditLogRepository.create({
      createdBy: dto.createdBy,
      action: dto.action,
      affectedId: dto.affectedId,
      affectedTable: dto.affectedTable,
      status: dto.status,
      description: dto.description,
    });

    await this.auditLogRepository.save(auditLog, { reload: false });
  }

  /**
   * Retrieves audit logs with pagination, filtering, and sorting.
   * Supports AC5 (pagination, date filtering, sorting).
   */
  async getAuditLogs(params: AuditLogFilter): Promise<{ logs: AuditLog[]; total: number }> {
    const {
      skip = 0,
      take = 10,
      startDate,
      endDate,
      userId,
      actionType,
      status,
    } = params;

    const query = this.auditLogRepository.createQueryBuilder('auditLog');

    // Apply filters
    if (startDate) {
      query.andWhere('auditLog.createdAt >= :startDate', { startDate });
    }
    if (endDate) {
      query.andWhere('auditLog.createdAt <= :endDate', { endDate });
    }
    if (userId) {
      query.andWhere('auditLog.createdBy = :userId', { userId });
    }
    if (actionType) {
      query.andWhere('auditLog.action = :actionType', { actionType });
    }
    if (status) {
      query.andWhere('auditLog.status = :status', { status });
    }

    // Apply pagination and sorting
    query.skip(skip).take(take).orderBy('auditLog.createdAt', 'DESC');

```

```
    // Execute query
    const [logs, total] = await query.getManyAndCount();

    return { logs, total };
  }
}

```

Service class. The @Injectable() annotation tells NestJS that it can be used via Dependency Injection.

```
@Injectable()
export class AuditLogService {
```

The AuditLog repository is being implemented. auditLogRepository can now be used to work with the database (create, save, find, queryBuilder, etc.).

```
  constructor(
    @InjectRepository(AuditLog)
    private readonly auditLogRepository: Repository<AuditLog>,
  ) {}
```

Method: Create an audit record

dto is an object with the data to be recorded (who did it, what they did, what they did it on, and the result).

this.auditLogRepository.create(...) — creates a new log record object (does NOT save it to the database yet).

this.auditLogRepository.save(auditLog, { reload: false }) — saves the record to the database.

{ reload: false } — after saving, do not reload the record back from the database (saves queries).

➡️ Result: the log is created in the database, nothing is returned (void).

```
  async createAuditLog(dto: CreateAuditLogDto): Promise<void> {
    const auditLog = this.auditLogRepository.create({
      createdBy: dto.createdBy,
      action: dto.action,
      affectedId: dto.affectedId,
      affectedTable: dto.affectedTable,
      status: dto.status,
      description: dto.description,
    });

    await this.auditLogRepository.save(auditLog, { reload: false });
  }
```

Method: Get a list of logs with filters

Method for getting a list of logs by filters and with pagination. Returns: logs — array of records. total — total number of records excluding skip and take. Values are extracted from the filter object. skip defaults to 0, take defaults to 10 (if not passed). TypeORM builds a SQL query via QueryBuilder. 'auditLog' is a table name.

```
async getAuditLogs(params: AuditLogFilter): Promise<{ logs: AuditLog[]; total: number }> {
    const {
      skip = 0,
      take = 10,
      startDate,
      endDate,
      userId,
      actionType,
      status,
```

```
    } = params;

    const query = this.auditLogRepository.createQueryBuilder('auditLog');
```

Applying filters: If a filter is passed, AND WHERE is added to SQL. Filters are only added if they are actually passed.

```
if (startDate) {
  query.andWhere('auditLog.createdAt >= :startDate', { startDate });
}
if (endDate) {
  query.andWhere('auditLog.createdAt <= :endDate', { endDate });
}
if (userId) {
  query.andWhere('auditLog.createdBy = :userId', { userId });
}
if (actionType) {
  query.andWhere('auditLog.action = :actionType', { actionType });
}
if (status) {
  query.andWhere('auditLog.status = :status', { status });
}
```

Pagination and sorting: skip — how many records to skip (for example, to go to the next page). take — how many records to take. orderBy — sort by creation date in order from new to old.

```
query.skip(skip).take(take).orderBy('auditLog.createdAt', 'DESC');
```

A database query is executed. logs — array of found logs. total — total number of matching records without skip/take restrictions.

```
const [logs, total] = await query.getManyAndCount();
    return { logs, total };
```

AuditLogController (audit-log.controller.ts)

1 Request for GET /admin/audit-logs is sent with a token
2 Nest checks the token via AuthGuard('jwt')
3 Controller gets the user from the token
4 Checks: if the user is not an admin — error 403
5 If an admin — calls the service and gets logs with filters
6 Returns the logs to the client

/home/kseniia/git/gigflow/backend/src/audit-log/audit-log.controller.ts

```
import { Controller, Get, Query, UseGuards, Req, ForbiddenException } from '@nestjs/common';
import { AuthGuard } from '@nestjs/passport';
import type { Request } from 'express';
import { AuditLogService } from './audit-log.service';
import { GetAuditLogsDto } from './dto/get-audit-log.dto';
import { AuditLog } from './entities/audit-log.entity';
//import { RolesGuard } from '../auth/guards/roles.guard';
//import { Roles } from '../auth/decorators/roles.decorator';
//import { Role } from '../users/entities/role.enum';

@Controller('admin/audit-logs')
//@UseGuards(AuthGuard('jwt'), RolesGuard)
//@Roles(Role.Admin)
export class AuditLogController {
  constructor(private readonly auditLogService: AuditLogService) {}

```

```
  @Get()
  async getAuditLogs(
    @Query() query: GetAuditLogsDto
  ): Promise<{ logs: AuditLog[]; total: number }> {
    return this.auditLogService.getAuditLogs(query);
  }
}

```

@Controller('admin/audit-logs') — sets the base path for all methods inside the class.
➔ All requests will be on the path like: GET /admin/audit-logs.

@UseGuards(AuthGuard('jwt')) — protects all controller methods by checking the JWT token. ➔ Without a valid token, access will be denied. The AuditLogService service is implemented through the constructor in order to then call its methods.

```
@Controller('admin/audit-logs')
@UseGuards(AuthGuard('jwt'))
export class AuditLogController {
  constructor(private readonly auditLogService: AuditLogService) {}
```

@Get() — binds the method to the HTTP GET request.

AuditLogInterceptor (audit-log-interceptor.ts)

/home/kseniia/git/gigflow/backend/src/audit-log/audit-log-interceptor.ts

```
import { Injectable, NestInterceptor, ExecutionContext, CallHandler } from '@nestjs/common';
import { Observable } from 'rxjs';
import { tap } from 'rxjs/operators';
import { Request } from 'express';
import { AuditLogService } from './audit-log.service';
import { ActionType } from './entities/audit-log.entity';

declare module 'express' {
    export interface Request {
      user?: {
        id: number;
        email: string;
        role: string;
      };
    }
  }

// Define type for body and response objects
interface RequestData {
  id?: number;
  [key: string]: unknown;
}

@Injectable()
export class AuditLogInterceptor implements NestInterceptor {
  // Array of sensitive fields that should be masked in logs
  private readonly sensitiveFields = ['password', 'paymentDetails', 'token', 'secret', 'creditCard',
'ssn'];

  constructor(private readonly auditLogService: AuditLogService) {}

  intercept(context: ExecutionContext, next: CallHandler): Observable<unknown> {
    const request = context.switchToHttp().getRequest<Request>();
```

```
    const { method, path, ip, body, headers } = request;
    const user = request.user as { id: number; email: string; role: string } | undefined; // Assumes Passport sets this

    // Mask sensitive fields in the request body (e.g., password, payment details)
    const safeBody = this.maskSensitiveData(body);

    return next.handle().pipe(
      tap({
        next: (response: unknown) => {
          const statusCode = context.switchToHttp().getResponse().statusCode;
          const description = this.buildDescription({
            method,
            path,
            ip,
            userAgent: headers['user-agent'],
            statusCode,
            body: safeBody,
            userEmail: user?.email as string,
            userRole: user?.role as string,
          });

          this.auditLogService.createAuditLog({
            createdBy: user?.id,
            action: this.mapMethodToAction(method),
            affectedId: this.extractAffectedId(path, body as RequestData, response as RequestData),
            affectedTable: this.extractAffectedTable(path),
            status: statusCode >= 400 ? 'FAILED' : 'SUCCESS',
            description,
          });
        },
        error: (error) => {
          const statusCode = error.status || 500;
          const description = this.buildDescription({
            method,
            path,
            ip,
            userAgent: headers['user-agent'],
            statusCode,
            body: safeBody,
            userEmail: user?.email,
            userRole: user?.role,
            errorMessage: error.message,
          });

          this.auditLogService.createAuditLog({
            createdBy: user?.id,
            action: this.mapMethodToAction(method),
            affectedId: this.extractAffectedId(path, body as RequestData),
            affectedTable: this.extractAffectedTable(path),
            status: 'FAILED',
            description,
          });
        },
      }),
    );
  }

```

```
  /**
   * Masks sensitive data in the request body, such as passwords or payment details.
   */
  private maskSensitiveData(data: unknown): unknown {
    if (!data || typeof data !== 'object') return data;

    // Recursively mask sensitive fields in nested objects
    const maskObject = (obj: unknown): unknown => {
      if (!obj || typeof obj !== 'object') return obj;

      if (Array.isArray(obj)) {
        return obj.map(item => maskObject(item));
      }

      // For non-array objects
      const result: Record<string, unknown> = { ...obj as Record<string, unknown> };

      for (const key in result) {
        if (Object.prototype.hasOwnProperty.call(result, key)) {
          if (typeof result[key] === 'object' && result[key] !== null) {
            result[key] = maskObject(result[key]);
          } else if (this.sensitiveFields.some(field => key.toLowerCase().includes(field.toLowerCase()))) {
            result[key] = '***';
          }
        }
      }

      return result;
    };

    return maskObject(data);
  }

  /**
   * Builds a detailed description of the request for logging.
   */
  private buildDescription(data: {
    method: string;
    path: string;
    ip: string | undefined;
    userAgent: string | undefined;
    statusCode: number;
    body: unknown;
    userEmail?: string;
    userRole?: string;
    errorMessage?: string;
  }): string {
    return JSON.stringify({
      method: data.method,
      path: data.path,
      ip: data.ip,
      userAgent: data.userAgent,
      statusCode: data.statusCode,
      body: data.body,
      userEmail: data.userEmail,
      userRole: data.userRole,
      errorMessage: data.errorMessage,
```

```
      timestamp: new Date().toISOString(),
    });
  }

  /**
   * Maps HTTP methods to audit log action types.
   */
  private mapMethodToAction(method: string): ActionType {
    switch (method.toUpperCase()) {
      case 'POST': return ActionType.CREATE;
      case 'PUT':
      case 'PATCH': return ActionType.UPDATE;
      case 'DELETE': return ActionType.DELETE;
      case 'GET':
      default: return ActionType.CREATE; // Fallback for GET and other methods
    }
  }

  /**
   * Extracts the ID of the affected entity from the path, body, or response.
   * Will convert string IDs to numbers or return undefined if not found.
   */
  private extractAffectedId(path: string, body: RequestData, response?: RequestData): number | undefined {
    // Try to extract numeric ID from path (e.g., /users/123)
    const numericIdMatch = path.match(/\/(\d+)(?:\/|$)/);
    if (numericIdMatch) return parseInt(numericIdMatch[1], 10);

    // Check body for id field (ensure it's a number)
    if (body?.id && typeof body.id === 'number') return body.id;
    if (body?.id && typeof body.id === 'string' && !isNaN(Number(body.id))) {
      return parseInt(body.id, 10);
    }

    // Check response for id field (ensure it's a number)
    if (response?.id && typeof response.id === 'number') return response.id;
    if (response?.id && typeof response.id === 'string' && !isNaN(Number(response.id))) {
      return parseInt(response.id as string, 10);
    }

    // For UUID or other non-numeric IDs, we can't use them directly
    // Consider adding a UUID field to the AuditLog entity if needed

    return undefined;
  }

  /**
   * Extracts the affected table/entity name from the path.
   * Handles API versioning and nested routes.
   */
  private extractAffectedTable(path: string): string {
    // Remove query parameters
    const pathWithoutQuery = path.split('?')[0];

    // Split path and remove empty segments
    const segments = pathWithoutQuery.split('/').filter(segment => segment.length > 0);

    // Skip API version segments (e.g., 'api', 'v1')
    let startIndex = 0;
```

```
    if (segments[0]?.toLowerCase() === 'api') startIndex++;
    if (segments[startIndex]?.toLowerCase().match(/^v\d+$/)) startIndex++;

    // Return first non-version segment as the resource/table name
    if (segments.length > startIndex) {
      return segments[startIndex];
    }

    return 'unknown';
  }
}
```

intercept — a mandatory method in any Interceptor. context — access to the request and response. next — continuation of the request execution.

```
intercept(context: ExecutionContext, next: CallHandler): Observable<any> {
    const request = context.switchToHttp().getRequest<Request>();
    const { method, path, ip, body, headers } = request;
    const user = request.user as { id: number; email: string; role: string } | undefined; // Assumes Passport sets this

    // Mask sensitive fields in the request body (e.g., password, payment details)
    const safeBody = this.maskSensitiveData(body);
```

If the request body contains passwords, cards, etc., replace them with ***.

```
const safeBody = this.maskSensitiveData(body);
```

next.handle() starts the request to the controller itself. tap — "interfere" with the flow and:

if the response is successful (next) — log success, if an error (error) — log failure.

```
return next.handle().pipe(
  tap({
    next: (response) => {
      ...
    },
    error: (error) => {
      ...
    },
  }),
);

```

Week 19 - Backend Testing

Taustajärjestelmän kehitys

[TOP Back to Contents]

HTTP tests are used for manual or automated verification of your backend API functionality. They help ensure that the main user scenarios (registration, login, profile retrieval) work correctly. You can test registering a new user via a POST request, to test user login and token retrieval, to test access to protected routes using a JWT token (for example, retrieving the user profile).

How to perform testing:

1. Execute the requests: GET, POST, UPDATE, DELETE.
2. Use the `@auth_token` variable to insert the authorization token into protected requests.
3. Analyze the server responses: make sure the expected statuses and data are returned.

4. You can quickly check API functionality using Postman.

API tests for users

```
### Register a new user
POST http://localhost:3000/api/users/register
Content-Type: application/json

{
  "email": "test2@example.com",
  "password": "password123"
}

### Login with the registered user
POST http://localhost:3000/api/auth/login
Content-Type: application/json

{
  "email": "test2@example.com",
  "password": "password123"
}

### Get user profile (requires auth token from login response)
GET http://localhost:3000/api/users/profile
Authorization: Bearer {{auth_token}}

### Variables
@auth_token =
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6InRlc3QyQGV4YW1wbGUuY29tIiwic3ViIjoyLCJpYXQiOjE3NDU4MzM0MjYsImV4cCI6MTc0NTgzNzAyNn0.Qkg7LDzNH0aRmgEYTm5r_NvGj6KO6iEBywQ5pDlO_T0
```

API tests for reviews

```
### Create a review (requires auth)
POST http://localhost:3000/api/reviews
Authorization: Bearer {{auth_token}}
Content-Type: application/json

{
  "title": "Great Service!",
  "content": "Everything was perfect.",
  "rating": 5
}

### Get all reviews
GET http://localhost:3000/api/reviews

### Get a review by ID
GET http://localhost:3000/api/reviews/REVIEW_ID

### Update a review (requires auth)
PATCH http://localhost:3000/api/reviews/REVIEW_ID
Authorization: Bearer {{auth_token}}
Content-Type: application/json

{
  "title": "Updated Title",
  "content": "Updated content.",
  "rating": 4
```

```
}

### Delete a review (requires auth)
DELETE http://localhost:3000/api/reviews/REVIEW_ID
Authorization: Bearer {{auth_token}}
```

API tests for jobs

```
### Create a job (requires auth)
POST http://localhost:3000/api/jobs
Authorization: Bearer {{auth_token}}
Content-Type: application/json

{
  "title": "Frontend Developer",
  "description": "React, TypeScript, remote",
  "salary": 3000
}

### Get all jobs
GET http://localhost:3000/api/jobs

### Get a job by ID
GET http://localhost:3000/api/jobs/JOB_ID

### Update a job
PUT http://localhost:3000/api/jobs/JOB_ID
Content-Type: application/json

{
  "title": "Updated Job Title",
  "description": "Updated job description.",
  "salary": 3500
}

### Delete a job
DELETE http://localhost:3000/api/jobs/JOB_ID
```

API tests for audit logs

```
### Get audit logs (admin only, requires auth)
GET http://localhost:3000/api/admin/audit-logs?limit=10&offset=0
Authorization: Bearer {{auth_token}}

### Get audit logs with filters (example)
GET http://localhost:3000/api/admin/audit-logs?userId=1&action=CREATE&from=2024-01-01&to=2024-12-31
Authorization: Bearer {{auth_token}}
```

Pull request for registration page gigflow: #31 Added register full page MERGED

Readme - project's tree:

User-auth-client/reportWebVitals.js file is used to measure and collect web application performance metrics. It allows you to track key performance indicators (Web Vitals) such as loading time, responsiveness, and interface stability. This data can be useful for analyzing and optimizing application performance.

libs/ui-components/package.json The package.json file is a configuration file used to manage a Node.js project or library. In this case, it describes the @gigflow/ui-components library. This file contains information about the project, its dependencies, entry points, and other settings.

libs/ui-components/tsconfig.json Delegates compilation tasks to other configuration files.

libs/ui-components/eslint.config.cjs The eslint.config.cjs file is a configuration file for ESLint, a static code analysis tool. It is used to configure linting rules (checking the code quality) in the ui-components library.

libs/ui-components/src/index.ts exports components that can use styles defined in styles.css. If a component (such as rating) uses classes such as .star svg, those styles will be applied automatically if styles.css is included in the project.

The vite.config.ts file is a configuration file for Vite, which is used to build and develop the `ui-components` library. It defines settings for developing, building, testing, and generating TypeScript types.

libs/ui-components/.storybook/preview.tsx

- `main.ts`: The main Storybook configuration file (stories, addons, build).
- `override.css`: Overrides styles to display components correctly.
- `preview.tsx`: Customizes the preview of stories (global styles, decorators, theme).
-

infra/docker/postgres/init.sql These files are used to configure and run PostgreSQL and PgAdmin in Docker containers. They automate the process of deploying and managing the database.

init.sql: Contains SQL queries for initial database setup.
.env: Stores environment variables for PostgreSQL and PgAdmin configuration.
docker-compose.yml: Defines how to run PostgreSQL and PgAdmin in Docker containers.

frontend/index.html The index.html file is the main HTML document for the front-end application. It serves as the entry point for the browser and provides the basic structure of the page into which the React application will be mounted.

app.tsx: The main component of the application, includes the theme and routes.
app.spec.tsx: Tests for the App component.
main.tsx: The entry point of the application, mounts App into the DOM.
index.tsx (in routes): Defines the application's routes.

backend/webpack.config.js: Configures how to build the server application using Webpack and Nx.
jest.config.ts: Configures how to test the server application using Jest.

- `backend/src/app/app.service.ts`: Contains business logic.
- `app.service.spec.ts`: Tests the correctness of the service.
- `app.module.ts`: The root module of the application, combines modules and configures the database.
- `app.controller.ts`: Processes HTTP requests and returns responses.
- `app.controller.spec.ts`: Tests the correctness of the controller.

Week 20 - Demo meeting

[TOP Back to Contents]

Links

Button links

☑ KEKO24-37: Button component DONE

Guide

Storybook
Vitests run tests `npx vitest run libs/ui-components/src/lib/button/button.spec.tsx`

Register page links

User Authentication and Profiles

User Story for login

KEKO24-58: Registration page DONE

https://www.figma.com/design/LWH8F4qcrSCbaEAT6DOVDY/Project-wireframe?node-id=927-4811&p=f&t=rQdUtK1SXZxfLios-0 Connect your Figma account

Frontend scenarios

Backend

Audit logs

KEKO24-87: Audit Logging DONE