FRONTEND SOFTWARE ARCHITECT

Technical Assessment

Presented To

MARC VILJOEN

Presented By

KUDAKWASHE NYENGERA

github.com/kudanmedia/huble

mckuda@gmail.com

+27 621 727 342

INTRODUCTION

I would like to extend my heartfelt gratitude to the entire team at Huble for affording me the opportunity to participate in the Frontend Software Architect technical assessment. This experience has been an exceptional opportunity to showcase my skills and passion for frontend development.

I am excited to take on the challenge and demonstrate my creative problem-solving abilities. Your trust in my capabilities is deeply appreciated, and I am committed to delivering exceptional solutions that align with Huble's ethos of excellence and innovation.

In this assessment, I am eager to tackle two engaging tasks that will not only test my technical prowess but also allow me to immerse myself in the world of frontend architecture. I wholeheartedly embrace these challenges and look forward to exceeding your expectations.

Please allow me to express my sincere gratitude once again for this incredible opportunity, and I am eager to present the results of my technical assessment.

TASK 1: MONOLITH HERO BANNER WITH LOGO SLIDER

For this task, I was asked to recreate the "Monolith hero banner with logo slider" module from the Monolith AI website. The client has provided the following feedback:
"Is it possible to reduce the spacing between the two CTAs?"

TASK 1: MONOLITH HERO BANNER WITH LOGO SLIDER

For this task, I was asked to recreate the "Monolith hero banner with logo slider" module from the Monolith AI website. The client has provided the following feedback:

1. "Is it possible to reduce the spacing between the two CTAs?"
2. "Can the logo slider constantly rotate and pause when the mouse hovers over a logo?"

IMPLEMENTATION

Atom Design Pattern

In alignment with Huble's design principles, I followed the Atom design pattern throughout the development of this module. This approach ensures modularity and reusability of components, making it easier to maintain and scale the codebase.

Tailwind CSS Integration

To enhance the styling and responsiveness of the module, I integrated the Tailwind CSS framework. Tailwind CSS allows for rapid development and customization of user interfaces, ensuring a polished and modern appearance.

Spacing Adjustment

To address the first client feedback regarding the spacing between the two Call-to-Action (CTA) buttons, I made the necessary adjustments to reduce the spacing while maintaining a visually appealing layout.

Logo Slider Behavior

To fulfill the second client feedback, I implemented the following behavior for the logo slider:

- The logo slider continuously rotates through logos.
- It pauses rotation when the mouse cursor hovers over a logo.
- Resumes rotation when the mouse cursor leaves the logo.

Reusable Components

In the implementation of the Hero module, I have leveraged reusable components for various UI elements, following best practices for modularity and maintainability.

LogoSlider Component

The LogoSlider component is used to display a slider of logos. It allows for easy inclusion and management of logo images.

```
import LogoSlider from '../../components/molecules/LogoSlider';

// ...

<LogoSlider logos={logos} />
```

Label Component

The Label component is used for displaying text with different intents, such as headings, subheadings, and descriptions.

```
import Label from '../atoms/Label';

// ...

<Label text="Monolith" intent="subHeading" />
<Label text="Use AI software to test less and learn more." intent="heading" />
<Label text="Spend less time running expensive, repetitive tests and more time learning from your
engineering data to predict the exact tests to run." intent="description" />
```

Button Component

The Button component is used to create buttons with different intents and sizes.

```
import Button from '../atoms/Button';

// ...

<Button label="REQUEST DEMO" intent={"primary"} size={"medium"}/>
<Button label="SPEAK TO OUR TEAM" intent={"secondary"} size={"medium"}/>
```

By encapsulating these UI elements into reusable components, we achieve better code organization, maintainability, and the ability to use these components across different parts of the application, promoting consistency in design and functionality.

Task 2: Calculator

Module Description

For the second task, I was asked to recreate a calculator using JavaScript, React, Vue, or jQuery, using your own code.

Implementation

Technology Stack

I chose to implement the calculator using React. This choice was made to ensure an optimal user experience and code maintainability.

Functionality

The calculator I developed includes the following features:

- Arithmetic operations
- Support for decimal numbers.
- Real-time display of the current input and result.

```
// Function to handle changes in the input field
const handleInputChange = (event) => {
  const newValue = event.target.value;

  if (!isNaN(newValue) || newValue === '') {
    setInput(newValue);
    setInputError('');
    if (operator && prevInput !== '') {
      calculateResult(prevInput, newValue, operator);
    }
  } else {
    setInputError('Please enter a valid numeric value.');
  }
};
```

Explanation:

The handleInputChange function is responsible for updating the input field and handling user input validation. It checks if the entered value is a valid numeric input and updates the input state accordingly. If an operator is already selected and there is a previous input, it also calculates the result in real-time.

```
// Function to handle operator button clicks
const handleOperatorClick = (selectedOperator) => {
  if (input !== '') {
    if (operator && prevInput !== '') {
      // If an operator is already selected and previous input exists, perform the previous operation
      calculateResult(prevInput, input, operator);
      setPrevInput(result.toString());
    } else {
      // If no operator is selected, set the current input as the previous input
      setPrevInput(input);
    }

    // Clear the input field, error message and set the new operator
    setInput('');
    setInputError('');
    setOperator(selectedOperator);
  }
};
```

Explanation:

The handleOperatorClick function handles operator button clicks. It checks if there is input in the input field and, based on the conditions, performs the following actions:

- If an operator is already selected and there's a previous input, it calculates the result of the previous operation and updates the previous input.
- If no operator is selected, it sets the current input as the previous input.
- It then clears the input field, clears any input validation errors, and sets the new operator.

These code snippets demonstrate the core functionality of the calculator, including real-time input validation, operator handling, and result calculation.

Please note that this is only a portion of the complete source code for the Calculator module.

Design Process in Alignment with Atomic Design

In the initial stages of this project, I adhered to the Atomic Design pattern to create a design system that promotes modularity and consistency in UI elements. Here's how the design process aligned with Atomic Design principles:

Atomic Design Components Library in Figma

- **Atoms:** At the atomic level, I identified and designed individual UI elements such as buttons, labels, and icons. Each atom represented a fundamental building block with its unique properties.
- **Molecules:** I composed molecules by combining multiple atomic elements, ensuring that their design remained consistent across the application. This approach facilitated reusability and maintained a unified look and feel.

- **Organisms:** Organisms were created by assembling molecules into larger UI components, such as the Monolith Hero Banner with Logo Slider. These organisms represented complex yet reusable sections of the user interface.

Wireframes and Prototypes

- **Wireframes:** Wireframes were initially designed to outline the layout and composition of organisms. These wireframes emphasized the arrangement of molecules within each organism while preserving the principles of Atomic Design.
- **Prototypes:** Prototypes were developed to bring organisms to life, showcasing their interactive behavior. This phase allowed for testing and validation of the user experience, ensuring that the design system components seamlessly integrated and functioned as expected.

Estimations and Costing

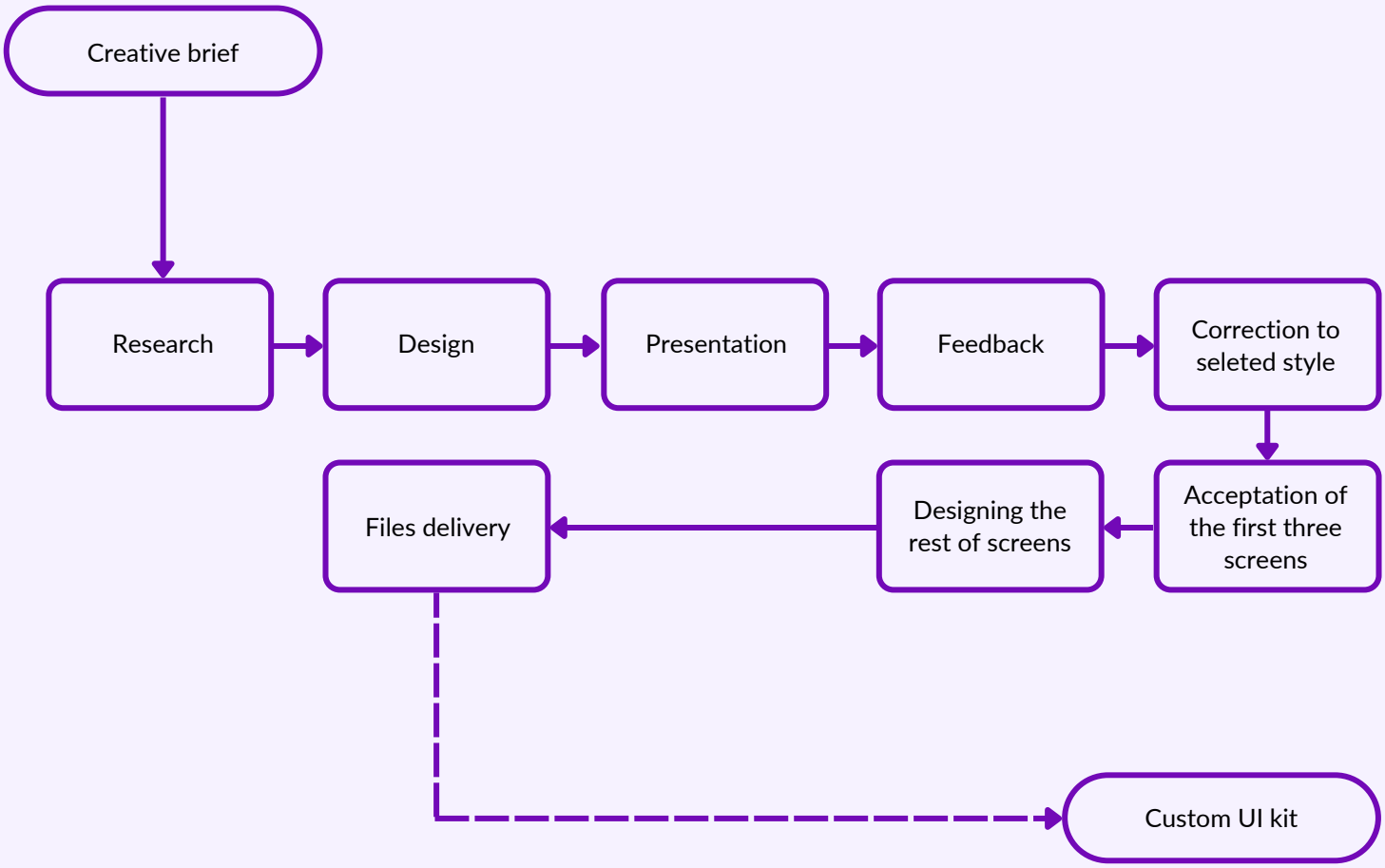
To estimate project timelines and costs while adhering to Atomic Design principles, the following steps were taken:

- **Granular Task Breakdown:** Each design and development task was broken down into atomic, molecular, or organism-level components. This granular approach ensured that estimations accounted for the modularity of the design system.
- **Resource Allocation:** Resources were allocated not only based on the tasks but also on the specific atomic, molecular, or organism components required. This approach facilitated efficient utilization of design and development resources.

- **Time and Cost Estimation:** Estimations considered the complexity of creating and integrating atomic components into molecules, which in turn formed organisms. This hierarchical estimation process ensured that both design and development efforts were accurately assessed.

By aligning the design process with Atomic Design principles, I aimed to create a cohesive and scalable design system. This system allowed for the seamless integration of UI components at the atomic, molecular, and organism levels, promoting consistency and reusability throughout the project.

UI Design Flow



Conclusion

In completing these tasks, I aimed to showcase my skills and expertise as a Frontend Software Architect. The alignment with Atomic Design principles, integration of Tailwind CSS, and adherence to best practices in development and design were key aspects of my approach.

Thank you for considering my application, and I look forward to discussing my solutions and the technical assessment in further detail.