Dokumentasi Cara kerja/flow Aplikasi kalkulator Coin Change dan 20 Solver

Komponen utama:

- **Model:** tempat menangani input data menjadi output sesuai algoritma
- **View:** tempat untuk mendapatkan input sekaligus menampilkan output
- **Controller:** tempat mengkoordinasikan antar komponen dan mengendalikan urutan komponen

A. Entry Point (sharedPageView.js)

Class template untuk kedua kalkulator

sharedPageView adalah kelas abstrak yang berfungsi sebagai entry point untuk tampilan halaman yang dibagikan dalam aplikasi. Kelas ini mewarisi dari abstractView dan menyediakan fungsionalitas umum untuk halaman-halaman yang menggunakan algoritma tertentu.

```
export default class sharedPageView extends abstractView
   constructor(algorithmTitle, algorithmDescription){
        super();
        this.algorithmTitle = algorithmTitle;
        this.algorithmDescription = algorithmDescription;
    }
 async getHtml(){
  //return html string
   }

       setInputField(inputFieldHtml){
        this.inputField = inputFieldHtml;
    }

    setController(controller){
        this.controller = controller;
    }

    setHtmlLogs(logsHtml){
        this.logs = logsHtml;
    }

   eventListener(){
     //entry point
     const submitButton = document.getElementById('submitButton');
     submitButton?.addEventListener('click', action => {
        action.preventDefault();

        try{
            document.getElementById("errorMessage").innerHTML = "";
            this.controller(this.getInput(), this)
```

```
        }catch (e){
            document.getElementById("errorMessage").innerHTML = "";
            setTimeout(() => {
                document.getElementById("errorMessage").innerHTML = e;
            }, 200);

        }
    }, false);
1.
```

class sharedPageView memiliki atribut penting yaitu :

- **Konstruktor**

  constructor(algorithmTitle, algorithmDescription)
  - algorithmTitle: Judul algoritma yang akan ditampilkan di halaman.
  - algorithmDescription: Deskripsi algoritma yang akan ditampilkan di halaman.

- **getHtml()**

  Metode ini mengembalikan string HTML yang merepresentasikan struktur halaman. Metode ini menggunakan atribut-atribut kelas seperti algorithmTitle, algorithmDescription, inputField yang bisa di sesuaikan oleh subclass

- **setInputField(inputFieldHtml)**

  Mengatur HTML untuk bagian input pada halaman.
  inputFieldHtml: String HTML yang merepresentasikan bidang input

- **setController(controller)**

  Mengatur fungsi controller untuk halaman.
  controller: Fungsi yang akan dipanggil saat tombol submit ditekan.

- **setHtmlLogs(logsHtml)**

  Mengatur HTML untuk bagian log pada halaman.
  logsHtml: String HTML yang merepresentasikan log.

- **eventListener() <- awal tempat pemanggilan controller**

  Metode ini menambahkan event listener ke tombol submit. Ketika tombol ditekan, metode ini akan mencoba menjalankan controller dengan input this.getInput() dan view itu sendiri. Jika terjadi error, pesan error akan ditampilkan.

sharedPageView menurunkan atributnya ke subCLass di file coinChangeVIew.js dan twentySolver.js yang mendefinisikan tiap page dari kalkulator

20-Solver class : twentySolverView.js

```
export default class extends sharedPageView {
    constructor(){
        super( //konstruktor parent class
          "20 - Solver",
          `20 - Solver adalah kalkulasi yang mencari kombinasi dari empat angka dengan
          operator aritmatika (+, -, *, /) untuk menghasilkan nilai 20.
          Algoritma ini menguji berbagai kombinasi operator dan penempatan tanda kurung
          untuk melihat apakah angka-angka tersebut dapat dioperasikan sehingga hasil
akhirnya adalah 20.
          Jika ada solusi yang ditemukan, maka akan menampilkan daftar ekspresi yang
valid.
          Jika tidak, maka akan menampilkan pesan error yang menyatakan bahwa operasi
tersebut
          tidak memungkinkan. Input 'List' berupa Angka (bilangan bulat). `
        );
        this.setTitle("Twenty Solver");
        this.setInputField(
          //html string mengenai input field
        )


        this.setController(twentySolverController);
    }
```

Coin Change class : CoinChangeView.js

```
export default class extends sharedPageView { //Coin Change
    constructor(){
        super( //konstruktor parent class
          "Coin Changes",
          `Coin Changes adalah kalkulasi menggunakan algoritma greedy yang digunakan untuk
          menentukan kombinasi koin yang tepat untuk mencapai jumlah tertentu menggunakan
          denominasi koin yang diinput.
          Algoritma greedy mengurutkan denominasi koin dari yang terbesar hingga terkecil,
          lalu menghitung berapa banyak koin dari setiap denominasi yang dibutuhkan.
          Jika nilai tersebut tidak dapat dipecah dengan koin yang tersedia,
          fungsi akan mengembalikan pesan error.
          Input 'Coin List' berupa angka (bilangan bulat) yang merepresentasikan jumlah
uang yang
          ingin dipecah menjadi koin. Input 'Target' berupa bilangan bulat yang berisi
denominasi koin.`
        );
        this.setTitle("Coin Changes");
        this.setInputField(
          //html string mengenai input field
        )
```

```
        this.setController(coinChangeController)
    }
1.
```

- **This.getInput() di dalam coinChangeView.js dan twentySolverView.js Terdapat 2 versi**
    1. **Coin Change**

```
    getInput(){
      const listInput = document.getElementById("listInput").value;
      const valueInput = document.getElementById("valueInput").value;
      return {
          body : {
                  X : valueInput,
                  arr : listInput
                }}
    }
```

Metode mengambil nilai dari dua elemen input: "listInput" dan "valueInput".

Mengembalikan objek dengan properti body yang berisi:

- X: nilai dari input "valueInput"
- arr: nilai dari input "listInput"

2. **TwentySolver**

```
    getInput(){
      const form = document.getElementById('listForm');
      const inputs = form.querySelectorAll('input[type="text"]');
      const valuesList = [];
      inputs.forEach(input => {
          valuesList.push(input.value);
      });
    return { body : { arr : valuesList } };

}
```

Metode ini mengambil input dari semua elemen input teks dalam form:

- Mencari semua elemen input teks dalam form dengan ID 'listForm'.
- Mengumpulkan nilai dari setiap input ke dalam array valuesList.

Metode ini mengembalikan objek dengan properti body yang berisi arr (array nilai dari semua input).

B. Controller

Coin Change controller

```
export function coinChangeController(req, view){
    validateCoinChangeInput(req)

    //if validation pass
    view.renderResultContainer();

    const { X, arr } = req.body;

    const validArr = arr.split(",").map(Number);
    const validX = Number(X)

    const results = calculateCoinChange(validX, validArr);

    view.generateResults(results, validX, validArr);
```

Input:

- req: Objek yang berisi:
  - body.X: Jumlah target (string)
  - body.arr: String denominasi koin yang dipisahkan koma
- view: Objek view untuk merender hasil

twenty Solver Controller

```
export function twentySolverController(req, view){
    validateTwentySolverInput(req);

    view.renderResultContainer();

    //if validation pass
    const { arr } = req.body;

    const results = calculateTwentySolver(arr);

    view.generateResults(results, arr);

}
```

Flow controller:

1. Memvalidasi input
2. mengubah input dari client ke list
3. Memanggil fungsi model untuk mentransformasi ouput/hasil
4. trigger view untuk merender output/hasil ke format html untuk di tampilkan di web

Penjelasan Komponen-Komponen yang di panggil Controller

A. Validation

Coin Change Validation : coinChangeValidation.js

```
export function validateCoinChangeInput(req){
    const { X, arr } = req.body;

    try{
        blankInputValidation(arr);
        blankInputValidation(X);
        IllegalInputValidation(X);
        IllegalInputValidation(arr);

        const validArr = arr.split(",").map(Number);
        const validX = Number(X)

        nonZeroValidation(validX)
        onlyIntListValidation(validArr);
        OnlyIntValueValidation(validX);
    }catch(e){
        throw e;
    }
}
```

20 – solver input valdation : twentySolverValidation.js

```
export function validateTwentySolverInput(req){
    const { arr } = req.body;

    try{
        for (const element of arr){
            IllegalInputValidation(element);
            blankInputValidation(element);
        }
        const validArr = arr.map(Number);


        onlyIntListValidation(validArr);
        sizeListValidation(validArr,4);

    }catch(e){
        throw e;
    }
}
```

Penjelasan util-util validasi yang ada di coinChangeValidation.js dan twentySolverValidation.js

validationUtils.js :

```
function onlyIntListValidation(list) {
    for (let element of list) {
        if (element === 0){
            throw Error("tidak boleh ada element list/angka yang 0 atau kosong");
        }

        if (element === null || element === undefined || Number.isNaN(element) ||
!Number.isInteger(element)) {
            throw Error("Input hanya boleh terdiri integer");
        }
    }
}

function OnlyIntValueValidation(input){
    if (!Number.isInteger(input)) {
        throw Error("Input harus integer");
    }
}

function IllegalInputValidation(input){
    let cleanInput = DOMPurify.sanitize(input);
    if ((input !== cleanInput)) {
        throw Error("terdeteksi input illegal");
    }
}

function blankInputValidation(input){
    if (input === ""){
        throw Error("Input tidak boleh kosong");
    }
}

function sizeListValidation(list, size){
    if (list.length !== size){
        throw Error(`Input \"List\" tidak boleh ada yang kosong`)
    }
}

function nonZeroValidation(input){
    if (input === 0){
        throw Error("Input tidak boleh 0")
    }
}
```

- **onlyIntListValidation(list)**: Fungsi ini memvalidasi bahwa elemen-elemen dalam sebuah daftar hanya terdiri dari bilangan bulat (integer). Fungsi akan memeriksa setiap elemen

dalam daftar, dan jika ditemukan elemen yang bernilai 0, null, undefined, atau bukan integer, fungsi akan mengeluarkan pesan kesalahan yang sesuai.

- **OnlyIntValueValidation(input)**: Fungsi ini memastikan bahwa nilai yang diberikan adalah integer. Jika input bukan integer, fungsi ini akan mengeluarkan pesan kesalahan.
- **IllegalInputValidation(input)**: Fungsi ini digunakan untuk mendeteksi input yang tidak aman atau berbahaya. Fungsi akan membersihkan input menggunakan DOMPurify, lalu membandingkan input asli dengan versi yang sudah dibersihkan. Jika ada perbedaan, itu berarti ada karakter ilegal dalam input, dan fungsi akan mengeluarkan pesan kesalahan.
- **blankInputValidation(input)**: Fungsi ini memvalidasi apakah input kosong. Jika input berupa string kosong, fungsi ini akan mengeluarkan pesan kesalahan.
- **sizeListValidation(list, size)**: Fungsi ini memastikan bahwa panjang daftar (list) sesuai dengan ukuran yang diharapkan. Jika panjang daftar tidak sesuai, fungsi akan mengeluarkan pesan kesalahan.
- **nonZeroValidation(input)**: Fungsi ini memvalidasi bahwa input tidak bernilai nol. Jika input bernilai 0, fungsi ini akan mengeluarkan pesan kesalahan.

B. Model

COIN CHANGE

```
export function calculateCoinChange(X, arr){

    // Urutkan denominasi koin dari terbesar ke terkecil
    arr.sort((a,b) => b - a);

    let result = {}; // Objek untuk menyimpan hasil(denomisasi koin dan jumlahnya)

    // Iterasi melalui setiap koin dalam array
    for (let i = 0; i < arr.length; i++) {
        let coinValue = arr[i];

        // Hitung berapa banyak koin ini yang bisa digunakan
        if (X >= coinValue) {
            let count = Math.floor(X/ coinValue); // Jumlah koin yang bisa diambil
            result[coinValue] = count;            // Simpan jumlah koin dalam result
            X -= count * coinValue;               // Kurangi nilai X sesuai dengan koin yang digunakan
        }
    }

    // Jika masih ada sisa X,
    if(X > 0) {
        return { error: "Tidak bisa memberikan kembalian dengan tepat"};
    }

    return {results : result };
}
```

Fungsi calculateCoinChange ini digunakan untuk menghitung kombinasi koin yang dibutuhkan untuk memberikan sejumlah nilai tertentu, berdasarkan denominasi koin yang tersedia. Fungsi ini mencoba memberikan kembalian yang paling efisien dengan menggunakan denominasi koin terbesar terlebih dahulu.

Parameter:

- X (number): Nilai yang ingin dihitung dalam bentuk kembalian.
- arr (array of numbers): Array yang berisi nilai-nilai denominasi koin yang tersedia, contohnya [1, 5, 10, 25].

Proses:

1. **Mengurutkan Denominasi**: Array koin diurutkan dari yang terbesar hingga terkecil agar dapat memberikan kembalian dengan menggunakan koin terbesar terlebih dahulu.
2. **Menghitung Koin yang Dibutuhkan**: Fungsi menghitung berapa banyak koin dari setiap denominasi yang dibutuhkan untuk mencapai nilai X. Jika nilai X masih lebih besar dari atau sama dengan nilai koin, maka jumlah koin yang bisa digunakan dihitung.
3. **Mengurangi Nilai X**: Setelah menghitung jumlah koin yang digunakan, nilai X akan dikurangi dengan jumlah koin yang digunakan dikali nilai koin tersebut.
4. **Memeriksa Sisa**: Jika setelah iterasi ada sisa nilai X yang tidak bisa dipecah dengan koin yang tersedia, fungsi akan mengembalikan error.
5. **Mengembalikan Hasil**: Fungsi akan mengembalikan objek yang berisi denominasi koin sebagai key dan jumlah koin yang digunakan sebagai value.

Return:

- Jika sukses, fungsi mengembalikan objek dalam format
  { results: { <nilai_koin>: <jumlah_koin> }}.
- Jika gagal (kembalian tidak bisa diberikan dengan tepat), fungsi mengembalikan objek error dalam format: { error: "Tidak bisa memberikan kembalian dengan tepat" }.

20 – SOLVER

```
export function calculateTwentySolver(numbers) {
    const operators = ['+', '-', '*', '/'];
    const target = 20;
    let solutions = [];
    let foundSolutions = false;

    // Fungsi untuk menghasilkan semua permutasi dari array
    function generatePermutations(arr) {
        let result = [];

        if (arr.length === 1) {
            return [arr];
        }

        for (let i = 0; i < arr.length; i++) {
            const currentNum = arr[i];
            const remainingNums = arr.slice(0, i).concat(arr.slice(i + 1));
            const remainingPermutations = generatePermutations(remainingNums);

            for (let permutation of remainingPermutations) {
                result.push([currentNum].concat(permutation));
            }
        }

        return result;
    }

```

```
    // Fungsi untuk membuat semua kombinasi operator dengan tanda kurung dinamis
    function bruteForceOperators(numbers, operators, target) {
        const [a, b, c, d] = numbers;

        // Definisikan berbagai template penempatan tanda kurung
        const expressionTemplates = [
            'A op1 B op2 C op3 D',
            '(A op1 B) op2 (C op3 D)',
            '(A op1 (B op2 C)) op3 D',
            '(A op1 B) op2 C op3 D',
            'A op1 (B op2 (C op3 D))',
            'A op1 ((B op2 C) op3 D)',
            '((A op1 B) op2 C) op3 D',
            '(A op1 B op2 (C op3 D))'
        ];

        // Loop untuk setiap kombinasi operator
        for (let i = 0; i < operators.length; i++) {
            for (let j = 0; j < operators.length; j++) {
                for (let k = 0; k < operators.length; k++) {

```

```
                    // Gantikan operator dan angka ke dalam template
                    expressionTemplates.forEach(template => {
                        let expression = template
                            .replace('A', a)
                            .replace('B', b)
                            .replace('C', c)
                            .replace('D', d)
                            .replace('op1', operators[i])
                            .replace('op2', operators[j])
                            .replace('op3', operators[k]);

                        try {
                            // Evaluasi setiap ekspresi yang dihasilkan
                            if (eval(expression) === target) {
                                solutions.push(`${expression} = ${target}`);
                                foundSolutions = true;
                            }
                        } catch (e) {
                            // Abaikan kesalahan seperti pembagian dengan nol
                        }
                    });
                }
            }
        }
    }

    // Dapatkan semua permutasi angka
    const permutations = generatePermutations(numbers);

    // Panggil fungsi brute force untuk setiap permutasi
    permutations.forEach(permutation => {
        bruteForceOperators(permutation, operators, target);
    });

    // Jika tidak ada solusi yang ditemukan, beri pesan bahwa tidak ada solusi
    if (!foundSolutions) {
        return { error: "Operasi ini tidak memungkinkan." };
    } else {
        return { results: solutions }; // Kembalikan semua solusi yang ditemukan
    }
}
```

Fungsi calculateTwentySolver ini dirancang untuk menyelesaikan masalah "24 Game" (atau versi untuk angka target 20) menggunakan empat angka dan operator dasar matematika. Tujuannya adalah menemukan semua ekspresi yang menghasilkan angka 20 dari kombinasi angka dan operator, dengan kemungkinan menggunakan tanda kurung.

Parameter:

- numbers (array of numbers): Array yang berisi empat angka yang akan dioperasikan untuk mencapai target 20. Setiap angka dapat digunakan sekali, dan urutannya dapat diubah.

Proses:

1. **Inisialisasi**:

- operators: Array yang berisi operator dasar matematika yang dapat digunakan, yaitu ['+', '-', '*', '/'].
- target: Nilai yang ingin dicapai, dalam hal ini adalah 20.
- solutions: Array kosong yang akan menampung semua ekspresi yang menghasilkan nilai target.
- foundSolutions: Boolean untuk melacak apakah ada solusi yang ditemukan.

2. **generatePermutations(arr)**:
   - Fungsi ini menghasilkan semua permutasi (pengurutan ulang) dari array numbers. Ini penting karena angka dapat diatur dalam urutan yang berbeda untuk mengeksplorasi semua kemungkinan.
   - Permutasi dihasilkan secara rekursif dengan menggabungkan setiap elemen dalam array dengan sisa elemen.
3. **bruteForceOperators(numbers, operators, target)**:
   - Fungsi ini mencoba berbagai kombinasi operator dan penempatan tanda kurung untuk empat angka yang diberikan. Ini menggunakan template ekspresi dengan berbagai cara penempatan tanda kurung.
   - Setiap kombinasi operator dari ['+', '-', '*', '/'] digantikan ke dalam template, menghasilkan beberapa ekspresi aritmatika berbeda.
   - Ekspresi tersebut dievaluasi menggunakan fungsi eval() untuk melihat apakah hasilnya sama dengan target (20). Jika sama, ekspresi yang valid disimpan dalam array solutions.
   - Kesalahan yang mungkin terjadi, seperti pembagian dengan nol, diabaikan dalam blok try-catch.
4. **Evaluasi Permutasi**:
   - Fungsi ini memanggil bruteForceOperators untuk setiap permutasi dari numbers, memastikan bahwa semua kombinasi angka dan operator diuji.
5. **Mengembalikan Hasil**:
   - Jika tidak ada solusi yang ditemukan, fungsi mengembalikan objek error: { error: "Operasi ini tidak memungkinkan." }.
   - Jika solusi ditemukan, fungsi mengembalikan objek dengan key results, yang berisi semua solusi dalam array.

Return:

- Jika berhasil menemukan solusi, fungsi mengembalikan objek dalam format:

{ results: ['ekspresi1 = 20', 'ekspresi2 = 20', ...] }

- Jika tidak ada solusi, fungsi mengembalikan:

{ error: "Operasi ini tidak memungkinkan." }

C. View.generateResults(result)

Generate semua output yang ada ke web ui

Contoh :