

Framer Motion

1. Animating with css transition

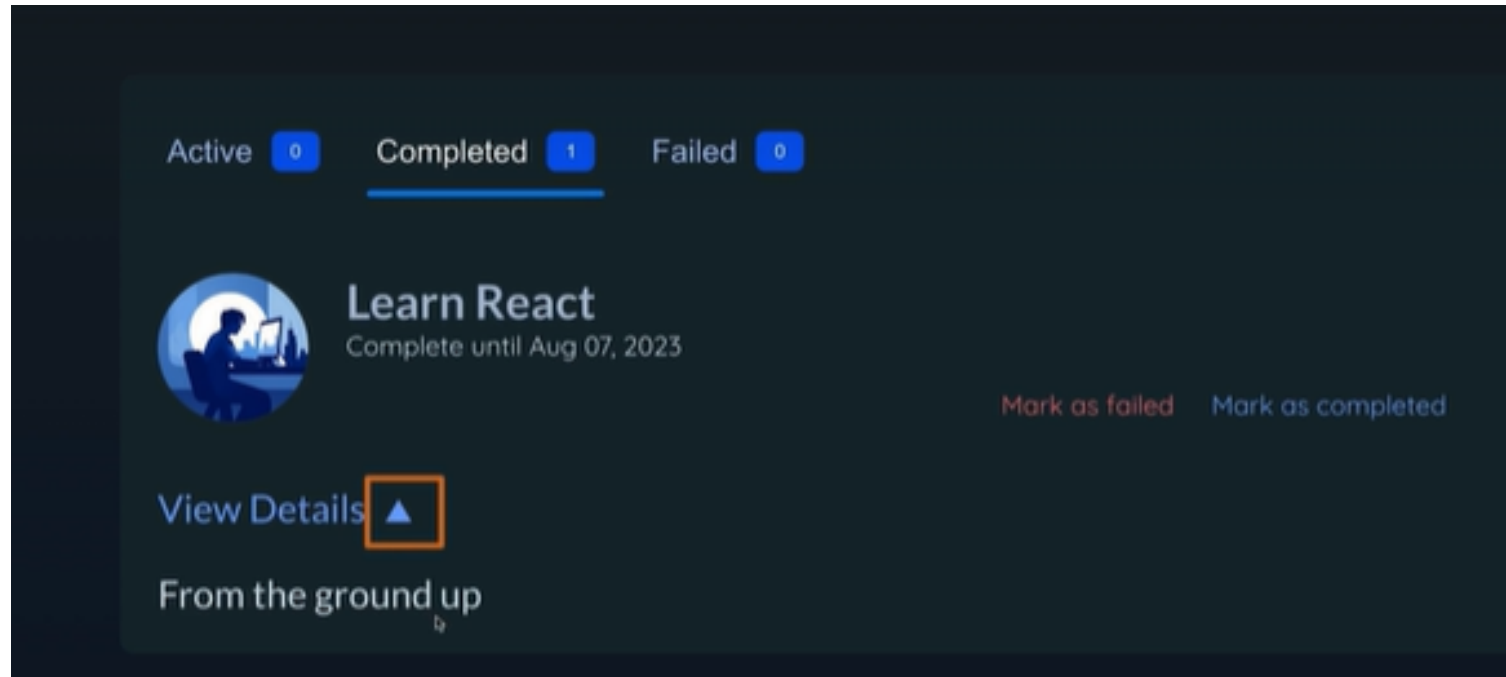
1. Animating with css transition

- Di css kita juga bisa membuat animasi walau fiturnya terbatas
- Dalam kasus tertentu menggunakan animasi css saja sudah cukup sehingga tidak perlu menginstal beberapa library

1. Animating with css transition

- Kita akan menganimasikan tombol dropown yang ada di gambar
- Ketika tombol ditekan , lebar komponen di sekitar akan meluas namun icon dropdownnya tidak pernah berubah
- Kita akan membuat iconnya kebawah ketika detailnya diperluas dan menghadap keatas ketika tidak
- Ini adalah salah satu fitur yang bisa dianimasikan hanya dengan css

1. Animating with css transition



1. Animating with css transition

- Kode tersebut ada di challengeItem.jsx dan classnya challenge-item-details-icon

```
489. Animating with CSS transitions
ChallengeItem.jsx X
43   |   </div>
44   | </header>
45   | <div className="challenge-item-details">
46   |   <p>
47   |     <button onClick={onViewDetails}>
48   |       View Details{' '}
49   |       <span className="challenge-item-details-icon">⌂</span>
50   |     </button>
51   |   </p>
52
```

1. Animating with css transition

- Kita sudah menulis kode untuk membalikkan icon walaupun belum dianimasikan
- Dan agar dapat di animasikan clas challenge-item-details harus memiliki class expanded dan bertetangga dengan challenge-item-details-icon

```
.challenge-item-details-icon {  
  display: inline-block;  
  font-size: 0.85rem;  
  margin-left: 0.25rem;  
}  
  
.challenge-item-details.expanded .challenge-item-details-icon {  
  transform: rotate(180deg);  
}
```

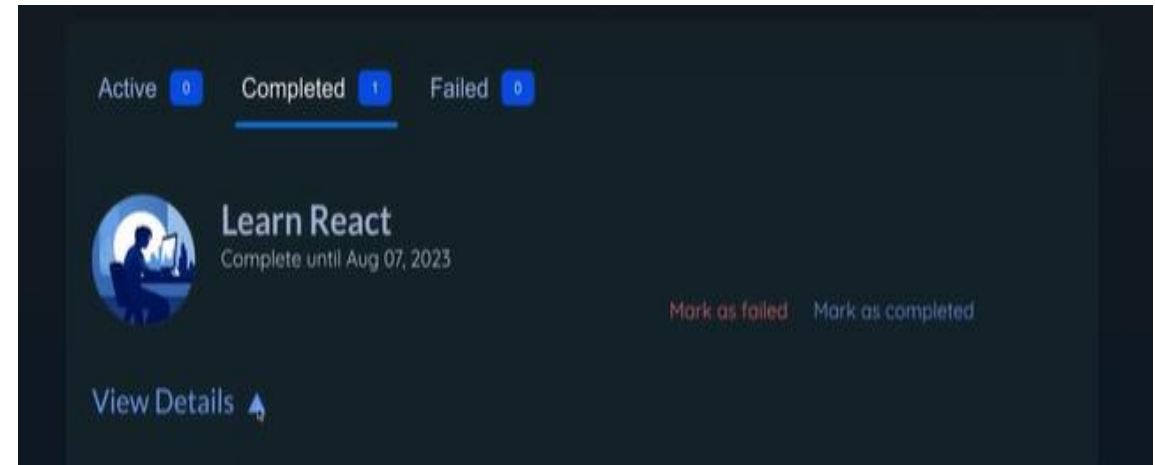
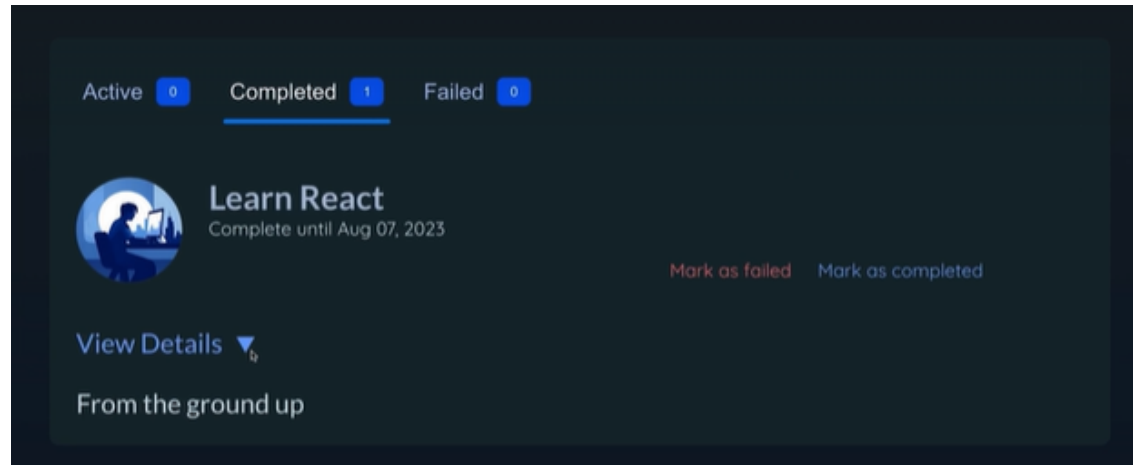
1. Animating with css transition

- Di file challenge-item.jsx ada prop isExpanded yang memvalidasi apakah elemen diperluas atau tidak
- tambahkan class challenge-item-details dengan pengkondisian

```
</header>
<div className={`challenge-item-details ${isExpanded ? 'expanded' : ''}`>
  <p>
    <button onClick={onViewDetails}>
      View Details{' '}
      <span className="challenge-item-details-icon">⌵</span>
    </button>
  </p>
```


1. Animating with css transition

- Sekarang tombolnya bisa berputar walaupun belum dianimasikan



1. Animating with css transition

- Kita akan menganimasikan property transform di class challenge-item-details-icon, untuk menganimasikan childrennya

```
.challenge-item-details.expanded .challenge-item-details-icon {  
  transform: rotate(180deg);  
}
```

- Kita tambahkan property transition, yang dianimasikan property transform dengan waktu animasi 0.3s dan gaya animasi ease-out

```
.challenge-item-details-icon {  
  display: inline-block;  
  font-size: 0.85rem;  
  margin-left: 0.25rem;  
  transition: transform 0.3s ease-out;  
}  
  
.challenge-item-details.expanded .challenge-item-details-icon {  
  transform: rotate(180deg);  
}
```

1. Animating with css transition

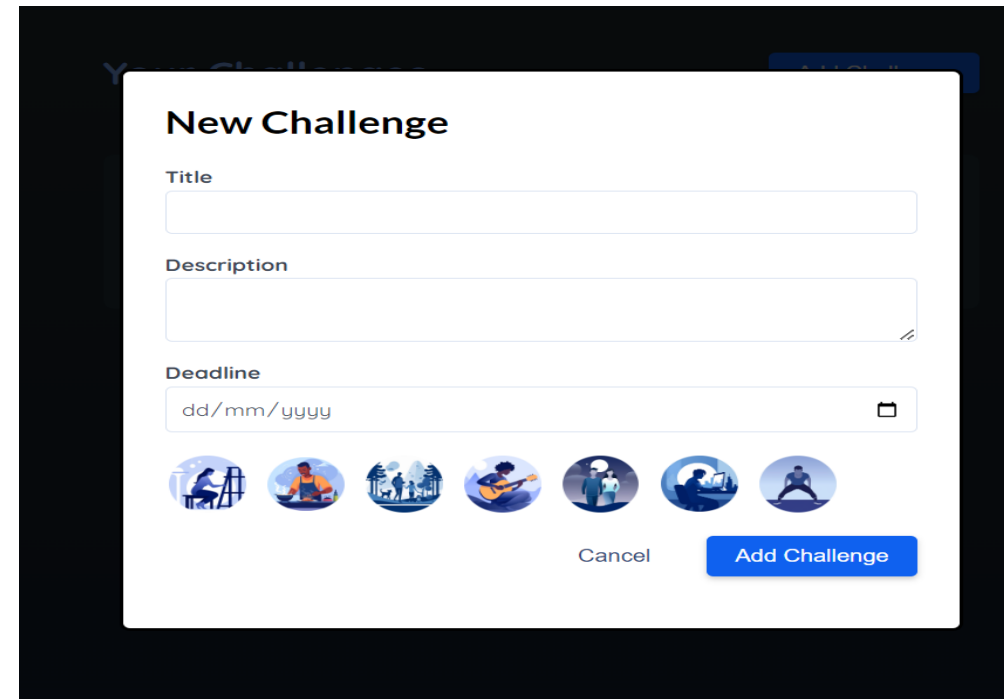
- Tampilan html yang dianimasikan

```
<div
  className={`challenge-item-details ${isExpanded ? "expanded" : ""}`}
>
  <p>
    <button onClick={onViewDetails}>
      View Details{" "}
      <span className="challenge-item-details-icon">#9650</span>
    </button>
  </p>
```

2. Animating with css animations

2. Animating with css animations

- Pada saat kita menekan add challenge yang memunculkan modal, alangkah baiknya jika modal tersebut muncul akan menimbulkan efek animasi seperti meluncur kebawah atau memudar saat dihilangkan



2. Animating with css animations

- Hal itu bisa dicapai dengan css animation, namun tidak menggunakan property transition
- Karena modal tersebut bukan bgian dari dom, melainkan menggunakan react contract dan komponen ini juga dimunculkan berdasarkan kondisi state
- Oleh karena itu kita butuh css animation, dengan css animation kita bisa menentukan posisi awal dan akhir animasi

```
@keyframes slide-up-fade-in {  
  9% {  
  }  
  100% {  
  }  
}
```

2. Animating with css animations

- Di kondisi 0% posisi awal, posisinya 30px lebih bawah dan opacitynya nol sehingga elemen tidak terlihat
- Dii kondisi 100% , posisinya akan kembali seperti semula dan opacitynya 1 sehingga terlihat
- Tambahkan animaasinya di class modal durasi 300ms ease-out

```
@keyframes slide-up-fade-in {  
  9% {  
    transform: translateY(30px);  
    opacity: 0;  
  }  
  100% {  
    transform: translateY(0);  
    opacity: 1;  
  }  
}
```

```
.modal {  
  top: 10%;  
  border-radius: 6px;  
  padding: 1.5rem;  
  width: 30rem;  
  max-width: 90%;  
  z-index: 10;  
  animation: slide-up-fade-in 0.3s ease-out forwards;  
}
```

2. Animating with css animations

- Sehingga halaman HTML nya seperti ini

```
return (

<Modal title="New Challenge" onClose={onDone}>  
  <form id="new-challenge" onSubmit={handleSubmit}>  
    <p>  
      <label htmlFor="title">Title</label>  
      <input ref={title} type="text" name="title" id="title" />  
    </p>  
    <p>  
      <label htmlFor="description">Description</label>  
      <textarea ref={description} name="description" id="description" />  
    </p>  
    <p>  
      <label htmlFor="deadline">Deadline</label>  
      <input ref={deadline} type="date" name="deadline" id="deadline" />  
    </p>  
    <ul id="new-challenge-images">  
      {images.map((image) => (  
        <li  
          key={image.alt}  
          onClick={() => handleSelectImage(image)}  
          className={selectedImage === image ? 'selected' : undefined}  
        >  
          <img {...image} />  
        </li>  
      ))}  
    </ul>  
    <p className="new-challenge-actions">  
      <button type="button" onClick={onDone}>  
        Cancel  
      </button>  
      <button>Add Challenge</button>  
    </p>  
  </form>  
</Modal>  
</div>  
</div>


```

```
import { createPortal } from 'react-dom';  
  
Codeium: Refactor | Explain | Generate JSDoc | X  
export default function Modal({ title, children, onClose }) {  
  return createPortal(  
    <div className="backdrop" onClick={onClose} />  
    <dialog open className="modal">  
      <h2>{title}</h2>  
      {children}  
    </dialog>  
  </>,  
    container: document.getElementById('modal')  
  );  
}
```


3. Pengenalan Framer-motion

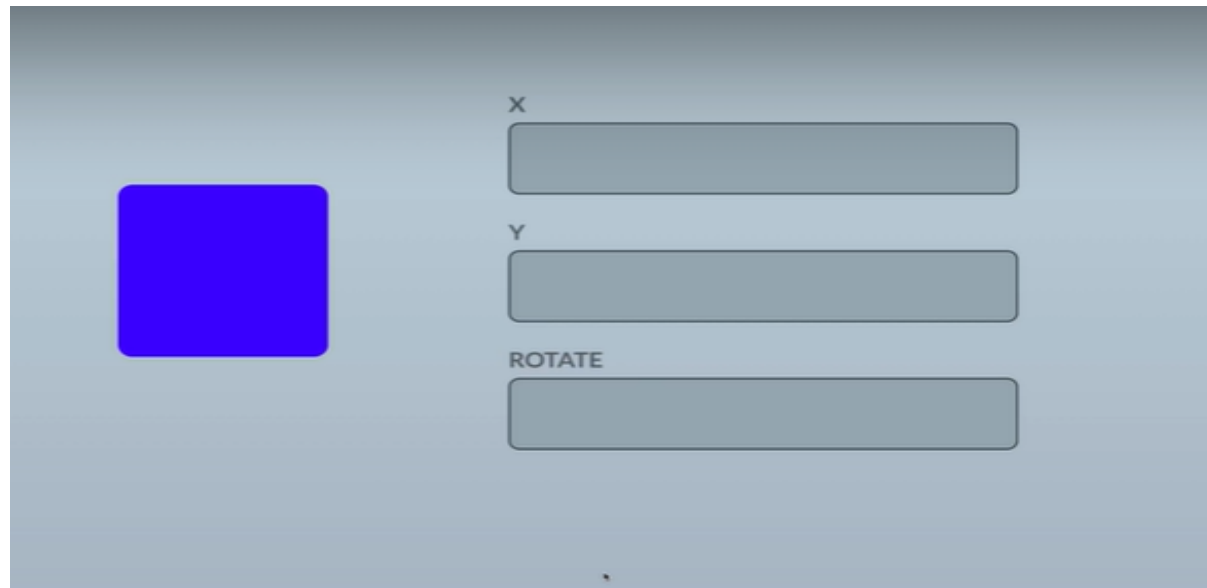
3. Pengenalan Framer-motion

- CSS transition dan css animation sangat hebat namun memiliki banyak keterbatasan, seperti tidak mampu membuat animasi ketika elemen menghilang dari dom
- Untuk animasi yang kompleks seperti ketika navbar beralih ke tab yang berbeda itu juga akan menyebabkan error
- Oleh karena itulah anda butuh library khusus animasi seperti framer-motion
- Install framer-motion dengan `npm i framer-motion`

4. Framer Motion Basics and Fundamental

4. Framer Motions Basic And Fundamental

- Kita bermain menggunakan file playground, bermain menganimasikan kotak biru dengan memasukkan nilai input
- Hal tersebut harus dicapai dengan framer-motion supaya posisi x,y,dan rotate bisa berpindah



4. Framer Motions Basic And Fundamental

- Di App.jsx kita harus mengimport motion, semua hasil inputan akan kita masukan ke state, komponen motion adalah komponen bawaan framer-motion

```
<motion.div id="box" />
```

- Ini akan merender div namun dikontrol oleh framer-motion untuk menghasilkan animasi dengan performa tinggi, selain div semua elemen html apat dianimasikan dengan motion

4. Framer Motions Basic And Fundamental

- Kita bisa menambahkan banyak special props di komponen motion
- Props animate berisi objek yang isinya banyak animasi yang diinginkan seperti merubah warna, posisi x,y,dll.
- X menandakan posisi elemen di sumbu horizontal dan Y menandakan elemen di sumbu vertikal
- Ketika isi dari property animate berubah maka react akan melakukan animasi, begitulah cara kerja framer-motion
- Property transition berguna untuk mengkonfigurasi animasi yang diinginkan, property ini berisi objek yang dapat mengatur berbagai macam hal seperti durasi animasi berisi angka dalam detik,

4. Framer Motions Basic And Fundamental

- Selain duration di property transition ada juga property type yang mengontrol animasi apa yang harus dimainkan, defaultnya adalah spring, ini adalah animasi dengan lompatan,
- Selain type, ada juga property bounce yang berisi integer berapa kali lompatan animasi terjadi, defaultnya adalah nol, jika type animasi anda adalah tween maka anda akan mendapatkan animasi sederhana tanpa lompatan

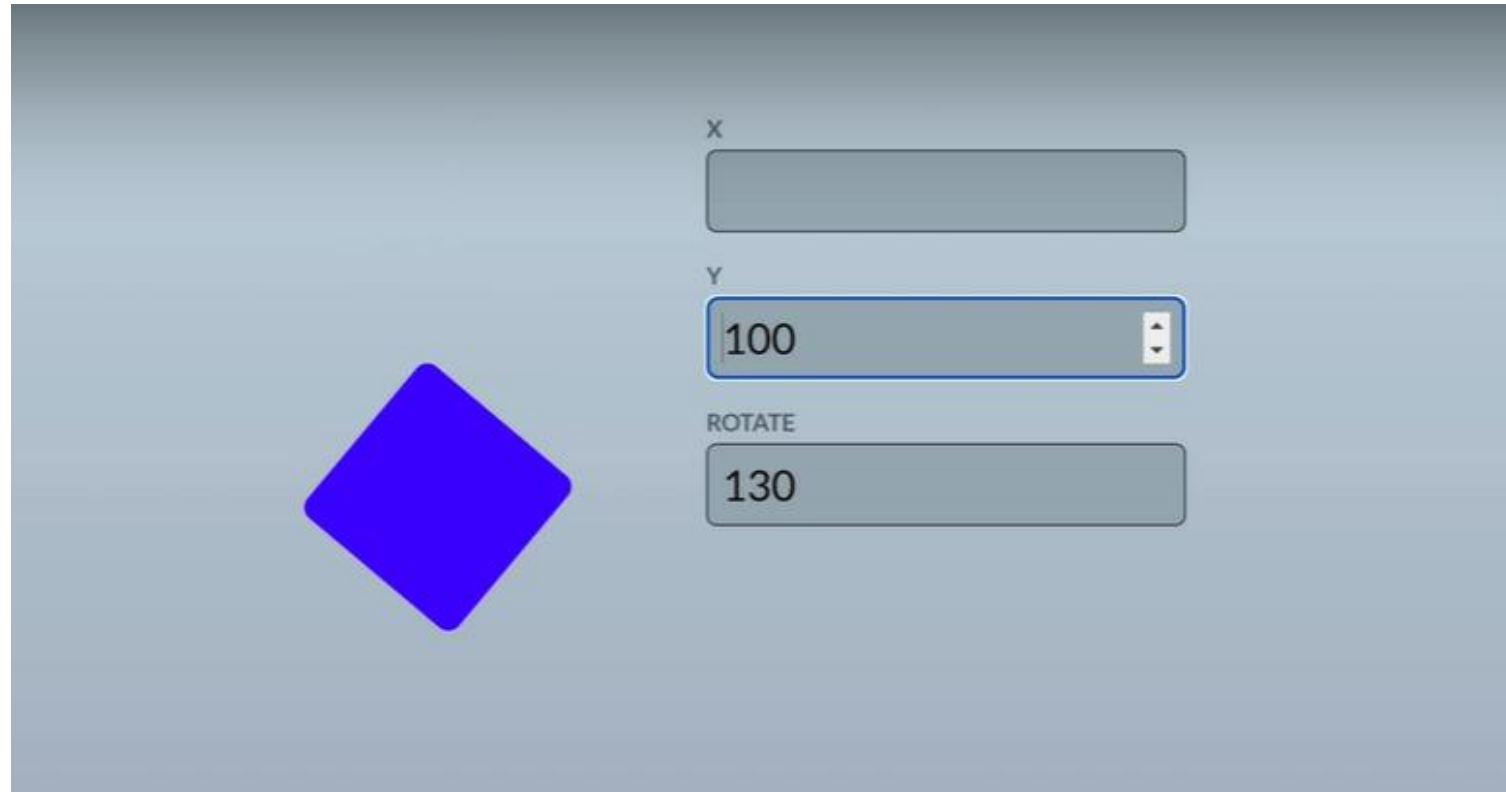
4. Framer Motions Basic And Fundamental

```
Codeium: Refactor | Explain | Generate JSDoc | X
function App() {
  const [x, setX] = useState( initialState: 0);
  const [y, setY] = useState( initialState: 0);
  const [rotate, setRotate] = useState( initialState: 0);

  return (
    <div id="demo">
      <motion.div
        id="box"
        animate={{ x, y, rotate }}
        transition={{
          duration: 0.3,
          type: "spring",
        }}
      />
      <div id="inputs">
        <p>
          <label htmlFor="x">X</label>
          <input
            type="number"
            id="x"
            onChange={(event) => setX(+event.target.value)}
          />
        </p>
        <p>
          <label htmlFor="y">Y</label>
          <input
            type="number"
            id="y"
            onChange={(event) => setY(+event.target.value)}
          />
        </p>
        <p>
          <label htmlFor="rotate">Rotate</label>
          <input
            type="number"
            id="rotate"
            onChange={(event) => setRotate(+event.target.value)}
          />
        </p>
      </div>
    </div>
  );
}

export default App;
```


4. Framer Motions Basic And Fundamental



5. Animating Between Conditional Values

5. Animating Between Conditional Values

- Sekarang kita akan menganimasikan project awal namun menggunakan framer-motion
- Buka challengeItem.js dan di animasi yang memutar icon, hapus pengkondisian dan hapus css transsinya di file index.jsx

```
8
9  ✓ .challenge-item-details-icon {
10    display: inline-block;
11    font-size: 0.85rem;
12    margin-left: 0.25rem;
13  }
14
15  ✓ .challenge-item-details.expanded .challenge-item-details-icon {
16    transform: rotate(180deg);
17  }
```

```
<div className={`challenge-item-details`} >
  <p>
```

5. Animating Between Conditional Values

- Di file challengeItem, tambahkan component motion Buka challengeItem.js dan di animasi yang memutar icon, hapus pengkondisian dan hapus css transsinya di file index.jsx
- motion nya ditaruh di tag span, tempat symbol dropdown berada, property animate nya terdapat rotate dengan nilai 0 jika pengkondisian gagal atau 180 (di rotate) jika pengkondisian bernilai true

```
<motion.span  
  animate={{ rotate: isExpanded ? 180 : 0 }}  
  className="challenge-item-details-icon"  
>  
  &#9650;  
</motion.span>
```

6. Adding Entry Animations

6. Adding Entry Animations

- Kita akan menganimasikan animasi masuk modal dari css ke framer-motion
- Di file modal.jsx di komponen modal kita harus ganti tag dialog menjadi motion.dialog
- Lalu ke file indxe.css di class modal hapus animasinya

```
import { createPortal } from "react-dom";
import { motion } from "framer-motion";

Codeium: Refactor | Explain | Generate JSDoc | X
export default function Modal({ title, children, onClose }) {
  return createPortal(
    children: <>
    <div className="backdrop" onClick={onClose} />
    <motion.dialog open className="modal">
      <h2>{title}</h2>
      {children}
    </motion.dialog>
  </>,
    container: document.getElementById("modal")
  );
}
```

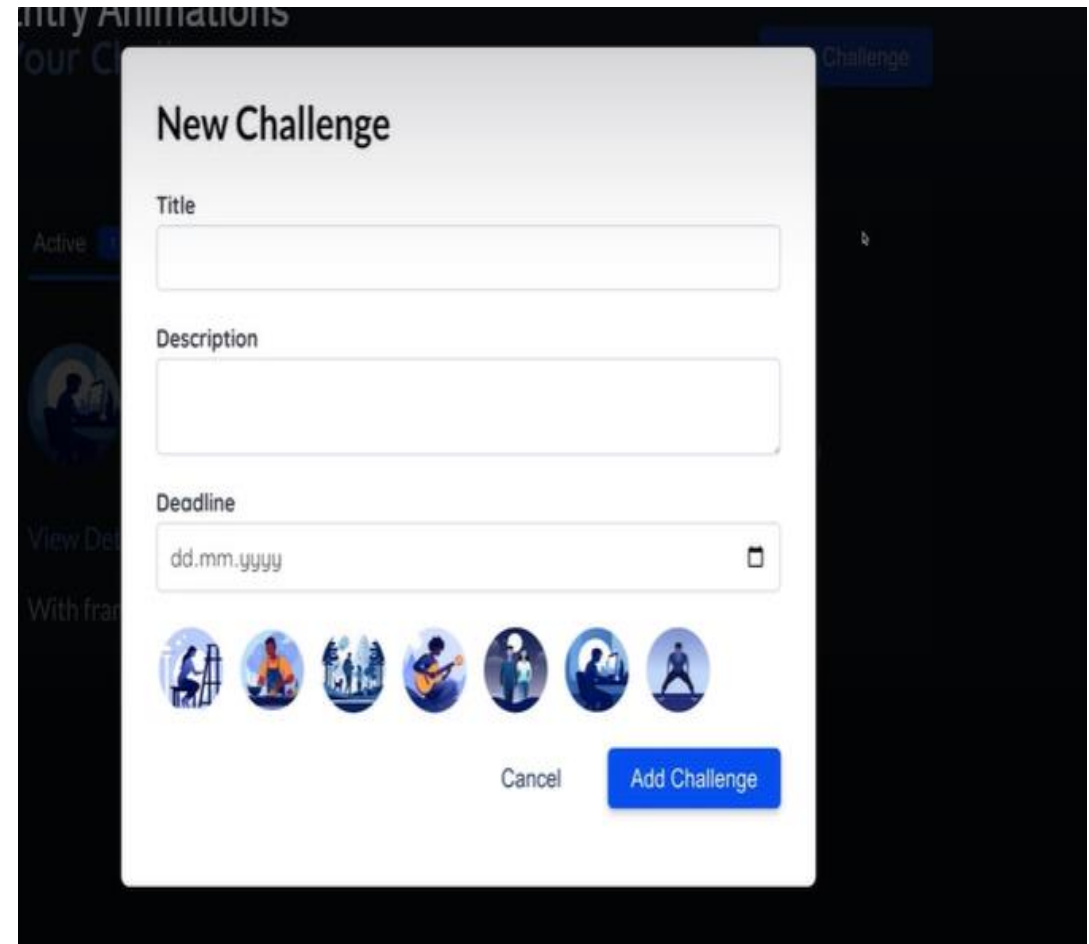
```
.modal {
  top: 10%;
  border-radius: 6px;
  padding: 1.5rem;
  width: 30rem;
  max-width: 90%;
  z-index: 10;
}
```

6. Adding Entry Animations

- Di motion.modal gunakan property animate untuk menganimasikan modal
- Karena modal ini muncul bukan karena pengkondisian atau state, sehingga untuk menganimasikannya
- Diperlukan property initial yang memberitahu kondisi awal sebelum animasi Di file modal.jsx
- Di property initial kita atur opacity nya 0 dan posisi y nya 30px
- Dan di property animasi opacitynya 1 serta posisinya nol
- Itulah cara memberitahu framer-motion mengenai kondisi awal animasi di tag initial dan kondisi akhir animasi di tag animate

6. Adding Entry Animations

```
export default function Modal({ title, children, onClose }) {  
  return createPortal(  
    children: <>  
    <div className="backdrop" onClick={onClose} />  
    <motion.dialog  
      open  
      className="modal"  
      initial={{ opacity: 0, y: 30 }}  
      animate={{ opacity: 1, y: 0 }}  
    >  
      <h2>{title}</h2>  
      {children}  
    </motion.dialog>  
  </>,  
  container: document.getElementById("modal")  
);  
}
```



The image shows a 'New Challenge' modal form. The form includes the following elements:

- Title:** A text input field.
- Description:** A text input field.
- Deadline:** A date input field with a placeholder 'dd.mm.yyyy' and a calendar icon.
- Activity Selection:** A row of seven circular icons representing different activities: a person climbing a ladder, a person sitting at a desk, a group of people, a person playing guitar, a person standing, a person sitting, and a person running.
- Buttons:** 'Cancel' and 'Add Challenge' buttons at the bottom right.

7. Animating Elements Dissappearances

7. Animating Elements Dissappearances

- Untuk menggunakan animasi karena hilangnya elemen dari dom , kita bisa menambahkan prop lain dari komponen motion yaitu prop exit()
- Prop exit berisi objek animasi

```
import { createPortal } from "react-dom";
import { motion } from "framer-motion";

Codeium: Refactor | Explain | Generate JSDoc | X
export default function Modal({ title, children, onClose }) {
  return createPortal(
    children: <>
      <div className="backdrop" onClick={onClose} />
      <motion.dialog
        open
        className="modal"
        initial={{ opacity: 0, y: 30 }}
        animate={{ opacity: 1, y: 0 }}
        exit={{ opacity: 0, y: 30 }}
      >
        <h2>{title}</h2>
        {children}
      </motion.dialog>
    </>,
    container: document.getElementById("modal")
  );
}
```

7. Animating Elements Disappearances

Namun jika kita hanya menggunakan motion component itu tidak akan mengubah react menangani element

Ini dikareakan komponen NewChallenge yang menjadai parentnya di file Header.jsx tidak akan menjadi return, karena kondisi nya bernilai false dan langsung dihapus oleh dom

```
Codeium: Refactor | Explain | Generate JSDoc | X
export default function Header() {
  const [isCreatingNewChallenge, setIsCreatingNewChallenge] = useState();

  Codeium: Refactor | Explain | Generate JSDoc | X
  function handleStartAddNewChallenge() {
    setIsCreatingNewChallenge(true);
  }

  Codeium: Refactor | Explain | Generate JSDoc | X
  function handleDone() {
    setIsCreatingNewChallenge(false);
  }

  return (
    <>
      {isCreatingNewChallenge && <NewChallenge onDone={handleDone} />}
      <header id="main-header">
        <h1>Your Challenges</h1>
        <button onClick={handleStartAddNewChallenge} className="button">
          Add Challenge
        </button>
      </header>
    </>
  );
}
```

7. Animating Elements Dissappearances

- Oleh krena itu file Header.jsx kita harus gunakan yaitu `AnimatePresence`, komponen itu memungkus kode yang secara kondisional menampilkan atau menghapus komponen,
- `Framer-motion` akan memastikan bhawa jika kode dieksekusi, komponen tersebut tidak langsung hilang dari dom , melainkan `framer-motion` akan mengecek terlebih dahulu apakah terdapat child komponen yang menggunakan motion dengan property `exit`?
- Jika ada maka `framer-motion` akan memainkan animasi terlebih dahulu sebelum menghapusnya

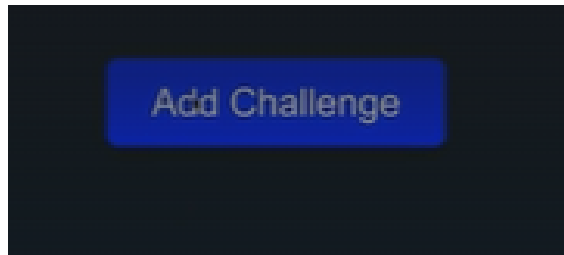
7. Animating Elements Dissappearances

```
<AnimatePresence>  
  {isCreatingNewChallenge && <NewChallenge onDone={handleDone} />}  
</AnimatePresence>
```

8. Making Elements Pop With Hover Animation

8. Making Elements Pop With Hover Animation

- Sekarang tombol newChallenge ini kita animasikan ketika di hover



```
<header id="main-header">
  <h1>Your Challenges</h1>
  <motion.button onClick={handleStartAddNewChallenge} className="button">
    Add Challenge
  </motion.button>
</header>
</>
```

8. Making Elements Pop With Hover Animation

- Kita tidak bisa menggunakan property animate
- Untuk itu kita perlu menggunakan property lain yang berawalan while
- Property while ada whileHover, whileTap, dll kita gunakan whileHover yang isinya berupa animasi
- Sekarang scale button meningkat tapi tidak memantul

```
<motion.button  
  whileHover={{ scale: 1.1 }}  
  onClick={handleStartAddNewChallenge}  
  className="button"  
>  
  Add Challenge  
</motion.button>
```


8. Making Elements Pop With Hover Animation

- Kita tidak bisa menggunakan property animate
- Untuk itu kita perlu menggunakan property lain yang berawalan while
- Property while ada whileHover, whileTap, dll kita gunakan whileHover yang isinya berupa animasi
- Sekarang scale button meningkat tapi tidak memantul, oleh karena itu kita juga perlu menambahkan property transition dengan type spring, kita juga bisa menambahkan stiffness (kekakuan) untuk menambah efek bouncing dari type :spring yang makin tinggi makin tiba-tiba bouncing nya

```
<motion.button
  whileHover={{ scale: 1.1 }}
  transition={{ type: "spring", stiffness: 500 }}
  onClick={handleStartAddNewChallenge}
  className="button"
>
  Add Challenge
</motion.button>
```

9. Reusing Animation States

9. Reusing Animation States

- Kembali ke komponen modal
- Animasi awal dan akhir itu sama, sehingga kita bisa menambahkan property agar tidak perlu menulis animasi berulang
- Kita bisa menambahkan property variants di komponen motion
- Variants disini berbeda dengan variants di bab 10
- Variants berisi objek dengan property apa saja untuk key, key tersebut berisi objek animasi

```
<div className="backdrop" onClick={onClose} />
<motion.dialog
  initial={{ opacity: 0, y: 30 }}
  animate={{ opacity: 1, y: 0 }}
  exit={{ opacity: 0, y: 30 }}
  open
  className="modal"
```

9. Reusing Animation States

- Disini kita menggunakan dua key yaitu key hidden dan visible
- Key itu hanya sebuah kunci, jika key dipanggil dalam property animate, whileHover, dsb, maka animasi dari key itulah yang akan ditampilkan

```
<div className="backdrop" onClick={onClose} />
<motion.dialog
  initial={{ opacity: 0, y: 30 }}
  animate={{ opacity: 1, y: 0 }}
  exit={{ opacity: 0, y: 30 }}
  open
  className="modal"
```

```
<motion.dialog
  open
  variants={{
    hidden: { opacity: 0, y: 30 },
    visible: { opacity: 1, y: 0 },
  }}
  className="modal"
  initial="hidden"
  animate="visible"
  exit="hidden"
>
  <h2>{title}</h2>
  {children}
</motion.dialog>
```

10. Nested Animations And Variants

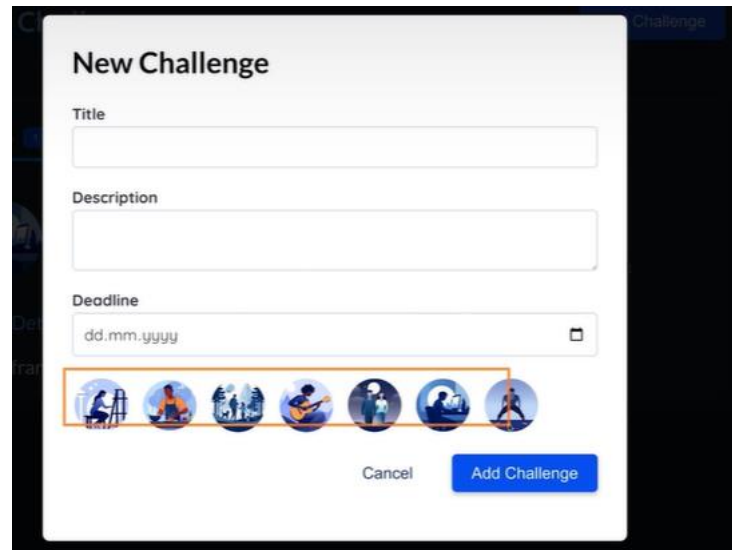
10. Nested Animations And Variants

- Variant tidak hanya berfungsi mendefinisikan dan menggunakan animasi
- Variant juga memicu animasi jauh didalam component tree hanya menggunakan component induk
- Di mosal komponen kita menggunakan variant

```
<motion.dialog
  variants={{
    hidden: { opacity: 0, y: 30 },
    visible: { opacity: 1, y: 0 }
  }}
  initial="hidden"
  animate="visible"
  exit="hidden"
```

10. Nested Animations And Variants

- Kita bisa menganimasikan tag li yang handle gambar, dengan variant yang sama tanpa harus mengaturnya, karena variant yang dideklarasikan di parent component juga akan diaktifkan di child component



The image shows a 'New Challenge' form with the following fields:

- Title:
- Description:
- Deadline:

Below the fields is a row of seven circular icons representing different activities: a person sitting at a desk, a person running, a group of people, a person playing guitar, a person standing, a person sitting, and a person running. The first icon is highlighted with an orange box.

At the bottom of the form are two buttons: 'Cancel' and 'Add Challenge'.

10. Nested Animations And Variants

- Ubah tag li menjadi motion.li
- Untuk menambahkan animasi yang sama kita tidak perlu menggunakan animate dan initial, kecuali ingin menampilkan animasi yang berbeda
- Kita hanya perlu mendefinisikan variant yang sama dengan induknya yaitu hidden dan visible karena akan otomatis diaktifkan oleh komponen induknya, dimana initial akan diisi otomatis dengan hidden, animate diisi visible dan exit diisi hidden
- Isi property hidden dan value boleh berbeda dan itu tidak masalah

10. Nested Animations And Variants

- Ubah tag li menjadi motion.li
- Untuk menambahkan animasi yang sama kita tidak perlu menggunakan animate dan initial, kecuali ingin menampilkan animasi yang berbeda
- Kita hanya perlu mendefinisikan variant yang sama dengan induknya yaitu hidden dan visible karena akan otomatis diaktifkan oleh komponen induknya, dimana initial akan diisi otomatis dengan hidden, animate diisi visible dan exit diisi hidden
- Isi property hidden dan value boleh berbeda dan itu tidak masalah

10. Nested Animations And Variants

```
<motion.li
  variants={{
    hidden: { opacity: 0, scale: 0.5 },
    visible: { opacity: 1, scale: 1 },
  }}
  transition={{ type: "spring" }}
  key={image.alt}
  onClick={() => handleSelectImage(image)}
  className={selectedImage === image ? "selected" : undefined}
>
  <img {...image} />
</motion.li>
))}
```

10. Nested Animations And Variants

- Namun ada masalah, ketika kita menutup modal
- Perlu waktu beberapa saat agar backdrop menghilang
- Hal ini dikarenakan property exit juga diturunkan ke child, sehingga animasi di child berjalan terlebih dahulu baru animasi di parentnya
- Satu-satunya cara adalah menimpa property exit dengan objek animasi berbeda dan sesuai dengan kondisi animasi akhir di property visible
- Namun kita tidak bisa property exit dengan property visible, karena ketika modal dibuka kembali animasi awalnya di childnya bukan hidden lagi tapi berganti menjadi visible sehingga animasi di child tidak berjalan

10. Nested Animations And Variants

- Cara satu-satunya adalah mencopy isi property variant visible kedalam property exit, sehingga kita benar-benar menimasi di child komponent, dan ketika modal ditutup, child komponen tidak menjalankan animasi karena state exit telah sama dengan state animasinya

11. Animating Staggered List

11. Animating Staggered List

- Framer-motion juga membantu kita menganimasikan item daftar list seperti ini



- Ketika kita ingin agar gambar tidak muncul secara bersamaan dalam waktu yang sama
- Tapi anda ingin membuat animasi gambar muncul satu demi satu
- Teknik ini disebut staggering

11. Animating Staggered List

- Sekarang pergi ke parent elemen dari tag motion.li yaitu tag ul dan ubah tag ul menjadi motion.ul

```
</p>
<motion.ul
  id="new-challenge-images"
  variants={{
    visible: {},
  }}
>
  {images.map((image) => (
    <motion.li
      variants={{
        hidden: { opacity: 0, scale: 0.5 },
        visible: {
          opacity: 1,
          scale: 1,
        },
        transition: { type: "spring" },
      }}
      key={image.alt}
      onClick={() => handleSelectImage(image)}
      className={selectedImage === image ? "selected" : undefined}
    >
      <img {...image} />
    </motion.li>
  ))}
</motion.ul>
```

11. Animating Staggered List

- Di variant visible nya kita tidak ingin mengatur animasi, di variant visible di motion.ul kita gunakan property transition, memasukan property transition didalam variants visible berarti mengatur transition khusus untuk property visible
- Di property transition kita gunakan property staggered Children, property ini akan mengatur delay elemen anak akan memulai animasinya, secara default bernilai 0, jika kita mengubah nilainya ke selain nol
- Maka setiap children akan dianimasikan setelah sekian detik setelah elemen sebelumnya dirender

11. Animating Staggered List

```
<motion.ul
  id="new-challenge-images"
  variants={{
    visible: {
      transition: { staggerChildren: 0.05 },
    },
  }}
>
  {images.map((image) => (
    <motion.li
      variants={{
        hidden: { opacity: 0, scale: 0.5 },
        visible: {
          opacity: 1,
          scale: 1,
          transition: { type: "spring" },
        },
      }}
      key={image.alt}
      onClick={() => handleSelectImage(image)}
      className={selectedImage === image ? "selected" : undefined}
    >
      <img {...image} />
    </motion.li>
  ))}
</motion.ul>
```

12. Animating Colors & Working with Keyframes

12. Animating Colors & Working with Keyframes

- Selain mengisi nilai property seperti opacity, scale, x, y, dll dengan angka, kita juga bisa mengisi nilai property dengan string atau function bawaan css seperti , “hex”, “calc()”, dll seperti motion button ini

12. Animating Colors & Working with Keyframes

```
Codeium: Refactor | Explain | Generate JSDoc | X
export default function Header() {
  const [isCreatingNewChallenge, setIsCreatingNewChallenge] = useState();

  Codeium: Refactor | Explain | Generate JSDoc | X
  function handleStartAddNewChallenge() {
    setIsCreatingNewChallenge(true);
  }

  Codeium: Refactor | Explain | Generate JSDoc | X
  function handleDone() {
    setIsCreatingNewChallenge(false);
  }

  return (
    <>
      <AnimatePresence>
        {isCreatingNewChallenge && <NewChallenge onDone={handleDone} />}
      </AnimatePresence>

      <header id="main-header">
        <h1>Your Challenges</h1>
        <motion.button
          whileHover={{ scale: 1.1, backgroundColor: "#8b11f0" }}
          transition={{ type: "spring", stiffness: 500 }}
          onClick={handleStartAddNewChallenge}
          className="button"
        >
          Add Challenge
        </motion.button>
      </header>
    </>
  );
}
```

12. Animating Colors & Working with Keyframes

- Selain itu kita juga mengisi value property dengan sebuah array
- Ke NewChallenge.jsx di motion.li I varian visible property scale, kalau angkanya diganti ke array[0.8,1.3,1]
- Artinya scale akan beralih ke 0.8 lalu ke 1.3 lalu terakhir ke posisi 1

```
57 >
58 {images.map((image) => (
59   <motion.li
60     variants={{
61       hidden: { opacity: 0, scale: 0.5 },
62       visible: {
63         opacity: 1,
64         scale: [0.8, 1.3, 1],
65       },
66       transition: { type: "spring" },
67     }},
68     key={image.alt}
69     onClick={() => handleSelectImage(image)}
70     className={selectedImage === image ? "selected" : undefined}
71   >
72     <img {...image} />
73   </motion.li>
74 )
75 )}
76 </motion.ul>
```

13. Imperative Animation

13. Imperative Animation

- Selama ini kita mendefinisikan animasi secara deklaratif dalam kode JSX kita motion props
- Namun terkadang kita ingin memulai animasi secara imperatif , bukan deklaratif seperti membuat animasi berdasarkan pengkondisian yang lebih rapi,
- Untuk itu kita perlu mengimport useAnimate, hook ini mengembalikan array dengan 2 data. Yaitu [scope,animate]
- Elemen scope adalah ref yang bisa ditambahkan di elemen
- Dan animate adalah fungsi yang bisa digunakan untuk memicu animasi

13. Imperative Animation

- [scope,animate]

```
const [scope,animate] = useAnimate();
```

- Misal kdi file newChallenge , kita ingin jika form tidak valid maka kita ingin mengguncang seluruh tag input

```
if (
  !challenge.title.trim() ||
  !challenge.description.trim() ||
  !challenge.deadline.trim() ||
  !challenge.image
) {
  animate();
  return;
}
```


13. Imperative Animation

- Argumen pertama animate adalah string
- Berisi tag elemen, atau class yang ingin dianimasikan
- Argumen kedua adalah objek yang berisi sama dengan property animate
- Argumen ketiga adalah konfigurasi bagaimana animasi akan dimainkan, ini akan sama seperti dengan property transition
- Di property transition yaitu delay kita punya fungsi stagger bawaan framer-motion
- Stagger ini merupakan function yang berfungsi sama seperti staggerChildren dan berisi jeda waktu
- Stagger juga merupakan fungsi penting jika kita ingin menggunakan animasi imperative dimana animasinya berfungsi pada children

13. Imperative Animation

```
    if (  
      !challenge.title.trim() ||  
      !challenge.description.trim() ||  
      !challenge.deadline.trim() ||  
      !challenge.image  
    ) {  
      animate(  
        "input , text-area",  
        {  
          x: [-10, 0, 10, 0],  
        },  
        {  
          type: "spring",  
          duration: 0.2,  
          delay: stagger( duration: 0.05),  
        }  
      );  
      return;  
    }  
  }  
}
```

13. Imperative Animation

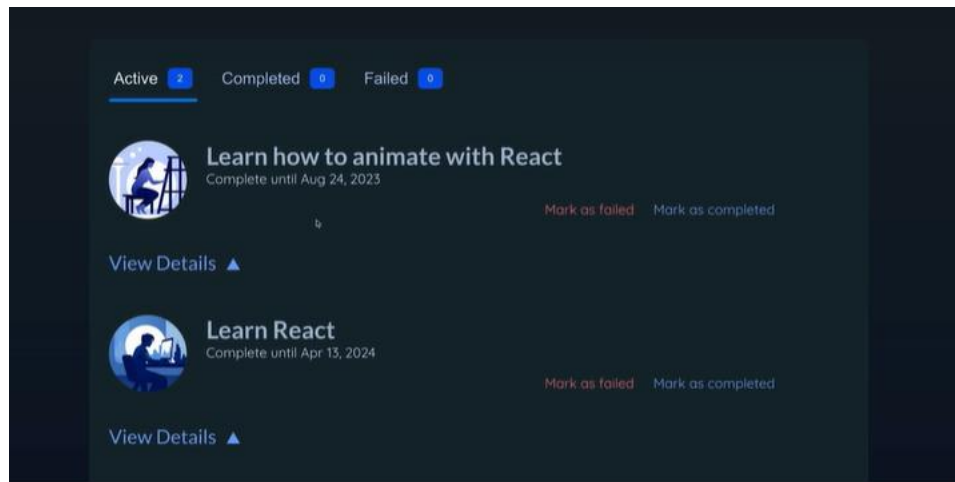
- Dan function scope untuk refnya akan dimasukkan kedalam ref di parent yang mau dianimasikan, dalam hal ini tag form
- Spring animasi hanya bisa menerima animasi yang hanya ada start dan end nya saja
- Jika ada animasi yang error kita bisa mengaktifkan show animation di windows 10

```
<Modal title="New Challenge" onClose={onDone}>
  <form id="new-challenge" onSubmit={handleSubmit} ref={scope}>
    <p>
      <label htmlFor="title">Title</label>
      <input ref={title} type="text" name="title" id="title" />
    </p>
    <p>
      <label htmlFor="description">Description</label>
      <textarea ref={description} name="description" id="description" />
    </p>
    <p>
      <label htmlFor="deadline">Deadline</label>
      <input ref={deadline} type="date" name="deadline" id="deadline" />
    </p>
    <motion.ul
```

14. Animating Layout Changes

14. Animating Layout Changes

- Kita akan menganimasikan beberapa item di tab, jika user menekan mark as completed maka komponen akan berpindah ke tab complete dan ketika menekan mark as failed akan berpindah ke tab failed
- Ketika komponen children berpindah baik ke tab completed ataupun failed, maka komponen di bawahnya akan mengisi ruang kosong dari komponennya yang berpindah



14. Animating Layout Changes

- Ini dapat dicapai dengan membuka `challengeItem` komponen dan ada tag `li`, kita akan ganti tag `li` menjadi `motion.li` dan menambahkan property `layout`
- Jika komponen ini ditambahkan, maka ketika salah satu child komponen dihapus dari dom, child komponen dibawahnya akan bergerak keatas mengisi child komponen yang dihapus, menciptakan efek gerak

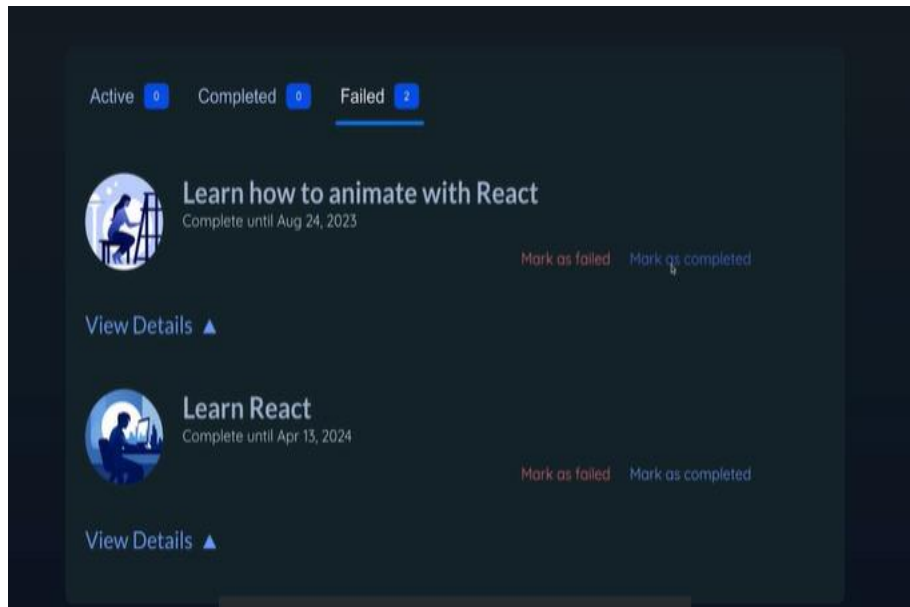
14. Animating Layout Changes

```
<motion.li layout>
  <article className="challenge-item">
    <header>
      <img {...challenge.image} />
      <div className="challenge-item-meta">
        <h2>{challenge.title}</h2>
        <p>Complete until {formattedDate}</p>
        <p className="challenge-item-actions">
          <button onClick={handleCancel} className="btn-negative">
            Mark as failed
          </button>
          <button onClick={handleComplete}>Mark as completed</button>
        </p>
      </div>
    </header>
    <div className={`challenge-item-details`} >
      <p>
        <button onClick={onViewDetails}>
          View Details{" "}
          <motion.span
            (property) className?: string | undefined
            className="challenge-item-details-icon"
          >
            &#9650;
          </motion.span>
        </button>
      </p>
      {isExpanded && (
        <div>
          <p className="challenge-item-description">
            {challenge.description}
          </p>
        </div>
      )}
    </div>
  </article>
</motion.li>
```

15. Orchestrating Multi-Element Animations

15. Orchestrating Multi-Element Animations

- Kita bisa menganimasikan child element yang lebih kompleks seperti ketika elemenitem dibawah hilang, elemen bawahnya akan naik keatas, lalu ketika elemen item tidak ada maka akan muncul teks yang naik keatas



15. Orchestrating Multi-Element Animations

Di file challenges.jsx disitulah kita merender semua list item

```
return (  
  <div id="challenges">  
    <ChallengeTabs  
      challenges={filteredChallenges}  
      onSelectType={handleSelectType}  
      selectedType={selectedType}  
    >  
      {displayedChallenges.length > 0 && (  
        <ol className="challenge-items">  
          {displayedChallenges.map((challenge) => (  
            <ChallengeItem  
              key={challenge.id}  
              challenge={challenge}  
              onViewDetails={() => handleViewDetails( id: challenge.id)}  
              isExpanded={expanded === challenge.id}  
            />  
          ))}  
        </ol>  
      )}  
      {displayedChallenges.length === 0 && <p>No challenges found.</p>}  
    </ChallengeTabs>  
  </div>  
);
```

15. Orchestrating Multi-Element Animations

- Karena kita ingin menganimasikan komponen yang hilang dan muncul, kita memerlukan `AnimatePresence` untuk membungkus component `ChallengeItem`

```
<div id="challenges">
  <ChallengeTabs
    challenges={filteredChallenges}
    onSelectType={handleSelectType}
    selectedType={selectedType}
  >
    {displayedChallenges.length > 0 && (
      <ol className="challenge-items">
        <AnimatePresence>
          {displayedChallenges.map((challenge) => (
            <ChallengeItem
              key={challenge.id}
              challenge={challenge}
              onViewDetails={() => handleViewDetails( id: challenge.id)}
              isExpanded={expanded === challenge.id}
            />
          ))}
        </AnimatePresence>
      </ol>
    )}
    {displayedChallenges.length === 0 && <p>No challenges found.</p>}
  </ChallengeTabs>
</div>
);
```

15. Orchestrating Multi-Element Animations

- Di ChallengeItem.jsx yang merupakan list yang di WraperAnimatePresence motion.li kita bisa menambahkan property exit untuk menganimasikan komponent yang dihapus

```
<motion.li layout exit={{ y: -30, opacity: 0 }}>
  <article className="challenge-item">
    <header>
      <img {...challenge.image} />
      <div className="challenge-item-meta">
        <h2>{challenge.title}</h2>
        <p>Complete until {formattedDate}</p>
        <p className="challenge-item-actions">
          <button onClick={handleCancel} className="btn-negative">
            Mark as failed
          </button>
          <button onClick={handleComplete}>Mark as completed</button>
        </p>
      </div>
    </header>
    <div className={`challenge-item-details`} >
      <p>
        <button onClick={onViewDetails}>
          View Details{" "}
          <motion.span
            animate={{ rotate: isExpanded ? 180 : 0 }}
            className="challenge-item-details-icon"
          >
            &#9650;
          </motion.span>
        </button>
      </p>
      {isExpanded && (
        <div>
          <p className="challenge-item-description">
            {challenge.description}
          </p>
        </div>
      )}
    </div>
  </article>
</motion.li>
```

15. Orchestrating Multi-Element Animations

- Ketika dianimasikan, dan ada 2 list, sebelum 1 komponen hilang dari dom, komponen 1 akan bergerak keatas dan perlahan menghilang
- Namun ketika komponen terakhir dihapus dari dom, animasi tidak bekerja
- Hal ini karena is ChallengeItem.jsx merupakan child dari komponen AnimatePresence di Challenge.jsx
- Dan ketika seluruh child hilang dari dom, tidak akan ada animasi karena challengeItem berasal dari array bersyarat yang tidak ada Animate Presence nya

15. Orchestrating Multi-Element Animations

- Untuk melakukannya kita harus membungkus pengkondisian dengan `AnimatePresence`, pembungkusan berakhir setelah tag paragraf yang menampilkan `No Challenges found`
- Di tag `ol` kita ubah jadi `motion.ol` dan tulis property `exit` dengan isi yang sama pada `ChallengeItem`, agar walaupun seluruh nilai array kosong, komponen terakhir juga akan dianimasikan, begitu juga di tag `p` ubah menjadi `motion.p`
- Sekarang `AnimatePresence` membungkus tag `motion.ol` dan `motion.p`
- Walaupun keduanya memiliki animasi yang berbeda, `framer-motion` akan menjalankan animasi tersebut secara bersamaan

15. Orchestrating Multi-Element Animations

- Untuk membedakannya kita perlu menambahkan property key di tag `motion.ol` dan tag `AnimatePresence`
- Karena tag `AnimatePresence` pertama memiliki dua child elemen yang dianimasikan yaitu `motion.ol` dan `motion.p`, maka kita harus menambahkan key unik di setiap child yang dianimasikan agar animasi tidak berjalan bersamaan
- Di `AnimatePresence` yang membungkus kita harus menambahkan property `mode="wait"` agar animasi selanjutnya tidak dijalankan sebelum animasi pertama selesai

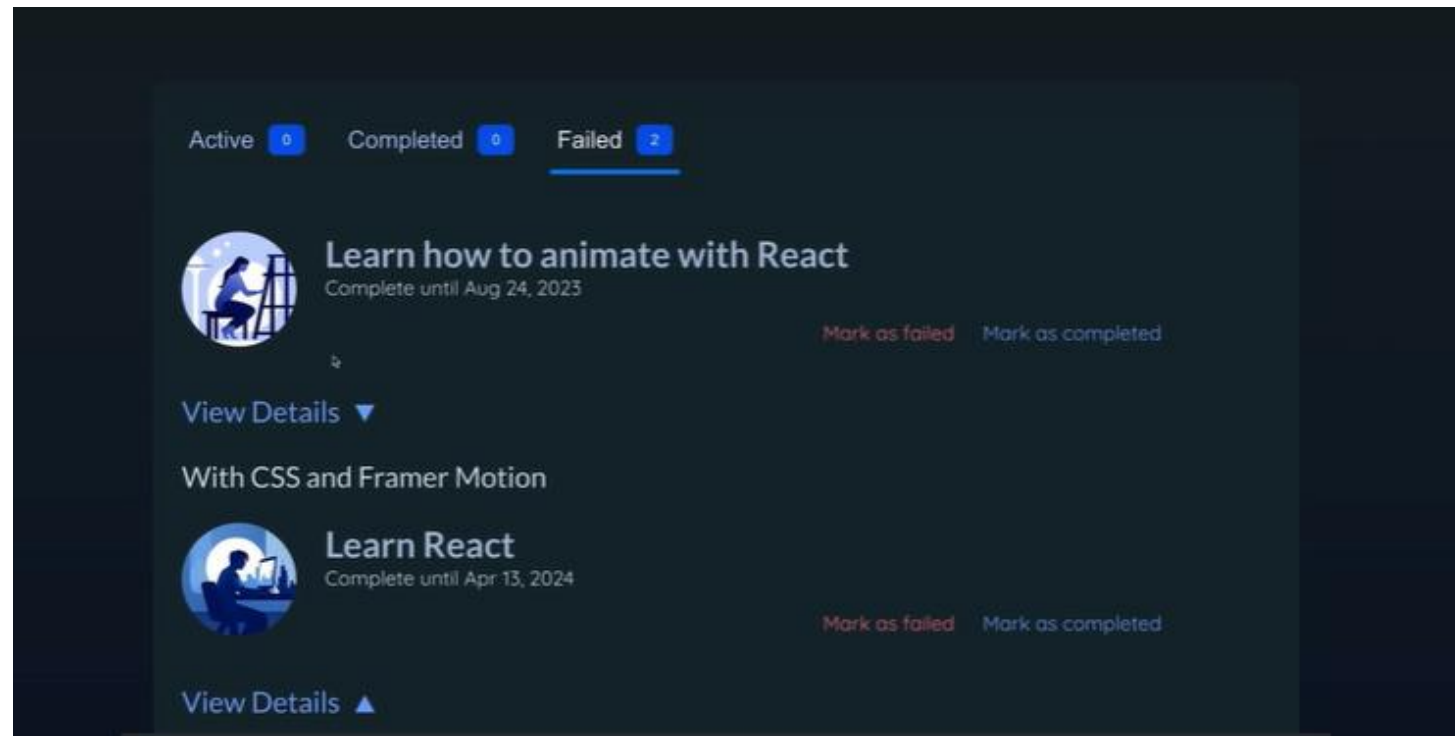
15. Orchestrating Multi-Element Animations

```
return (  
  <div id="challenges">  
    <ChallengeTabs  
      challenges={filteredChallenges}  
      onSelectType={handleSelectType}  
      selectedType={selectedType}  
    >  
      <AnimatePresence mode="wait">  
        {displayedChallenges.length > 0 && (  
          <motion.ol  
            key="list"  
            exit={{ y: -30, opacity: 0 }}  
            className="challenge-items"  
          >  
            <AnimatePresence>  
              {displayedChallenges.map((challenge) => (  
                <ChallengeItem  
                  key={challenge.id}  
                  challenge={challenge}  
                  onViewDetails={() => handleViewDetails( id: challenge.id)}  
                  isExpanded={expanded === challenge.id}  
                >  
                </>  
              </>  
            </>  
          </AnimatePresence>  
        </motion.ol>  
      )}  
      {displayedChallenges.length === 0 && (  
        <motion.p  
          key="fallback"  
          initial={{ opacity: 0, y: -20 }}  
          exit={{ y: -20, opacity: 0 }}  
          animate={{ opacity: 1, y: 0 }}  
        >  
          No challenges found.  
        </motion.p>  
      )}  
    </AnimatePresence>  
  </ChallengeTabs>  
</div>  
</>  
);  
}
```


16. Combining Animations with Layout Animations

16. Combining Animations with Layout Animations

- Ada perilaku aneh jika tombol view detail ditekan , item didalamnya akan bergoyang-goyang



16. Combining Animations with Layout Animations

- Perilaku ini disebabkan karena ChallengeItem di tag motion.li memiliki property layout
- Jika kita menghapus property layout, perilaku anehnya tidak muncul tetapi animasi sliding nya juga hilang, karena property layout menyebabkan perubahan tata letak pada ChallengeItem, dalam hal ini perubahan akibat ketinggian, itulah yang terjadi ketika menekan tombol views details

16. Combining Animations with Layout Animations

```
return (
  <motion.li layout exit={{ y: -30, opacity: 0 }}>
    <article className="challenge-item">
      <header>
        <img {...challenge.image} />
        <div className="challenge-item-meta">
          <h2>{challenge.title}</h2>
          <p>Complete until {formattedDate}</p>
          <p className="challenge-item-actions">
            <button onClick={handleCancel} className="btn-negative">
              Mark as failed
            </button>
            <button onClick={handleComplete}>Mark as completed</button>
          </p>
        </div>
      </header>
      <div className={`challenge-item-details`} >
        <p>
          <button onClick={onViewDetails}>
            View Details{" "}
            <motion.span
              animate={{ rotate: isExpanded ? 180 : 0 }}
              className="challenge-item-details-icon"
            >
              &#9650;
            </motion.span>
          </button>
        </p>
        {isExpanded && (
          <div>
            <p className="challenge-item-description">
              {challenge.description}
            </p>
          </div>
        )}
      </div>
    </article>
  </motion.li>
);
```

16. Combining Animations with Layout Animations

- Solusi masalah ini adalah menggunakan animasi di paragraf untuk menampilkan deskripsi

```
{isExpanded && {  
  <div>  
    <p className="challenge-item-description">  
      {challenge.description}  
    </p>  
  </div>  
}
```

16. Combining Animations with Layout Animations

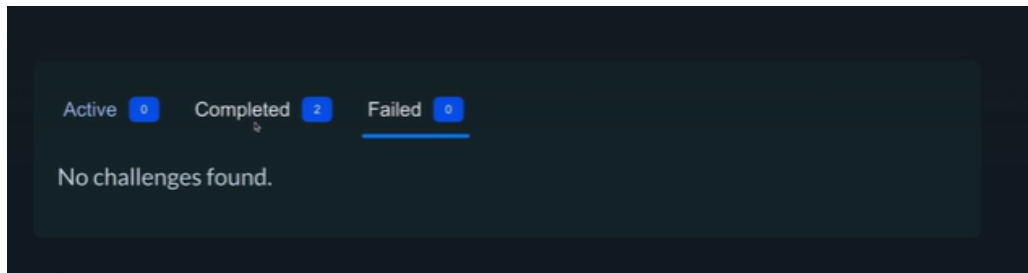
- Kita harus mengubah divnya menjadi motion.div dan berikan kondisi initial, animate, dan exit animasi dan karena kita menggunakan pengkondisian untuk memunculkan elemen, kita juga harus membungkusnya dengan AnimatePresence

```
56 </p>
57 <AnimatePresence>
58   {isExpanded && (
59     <motion.div
60       initial={{ height: 0, opacity: 0 }}
61       animate={{ height: "auto", opacity: 1 }}
62       exit={{ height: 0, opacity: 0 }}
63     >
64       <p className="challenge-item-description">
65         {challenge.description}
66       </p>
67     </motion.div>
68   )}
69 </AnimatePresence>
70 </div>
```

17. Animating Shared Elements

505. Animating Shared Elements

- Bagaimana jika kita ingin menganimasikan permindahan tab dengan underline?
- Misal kita ingin berpindah dari tab active ke tab lain atau sebagainya



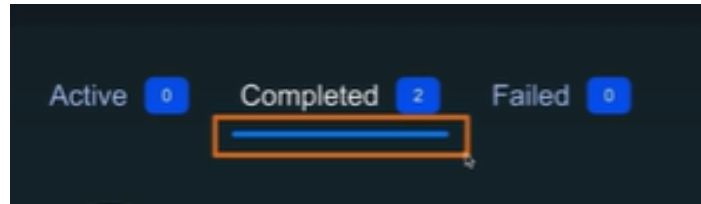
- Pertama di Challenges.jsx di motion.ol kita juga menambahkan initial dan animate dan exit agar dianimasikan ketika elemennya menghilang atau pertama kali muncul

505. Animating Shared Elements

```
<AnimatePresence mode="wait">
  {displayedChallenges.length > 0 && (
    <motion.ol
      key="list"
      initial={{ opacity: 0, y: -20 }}
      exit={{ y: -20, opacity: 0 }}
      animate={{ opacity: 1, y: 0 }}
      className="challenge-items"
    >
      <AnimatePresence>
        {displayedChallenges.map((challenge) => (
          <ChallengeItem
            key={challenge.id}
            challenge={challenge}
            onViewDetails={() => handleViewDetails( id: challenge.id)}
            isExpanded={expanded === challenge.id}
          />
        ))}
      </AnimatePresence>
    </motion.ol>
  )}
```

505. Animating Shared Elements

- Sekarang kita focus ke perpindahan tab bar ini



- Underline yang warna biru ini ada pada komponen div di file ChallengeTabs.jsx,

```
{isSelected && <div className="active-tab-indicator" />}
```

505. Animating Shared Elements

- Kita animasikan Tabsnya menjadi motion.div
- Kita bisa menggunakan special props yaitu layoutId yang bisa diisi string apa saja misalnya tab-indicator
- Ini secara otomatis mendeteksi ketika kita merender elemen lain dengan layoutId yang sama disuatu tempat pada satu halaman

```
{isSelected && (  
  <motion.div layoutId="tab-indicator" className="active-tab-indicator" />  
)}
```

18. Re-triggering Animations via Keys

18. Re-triggering Animations via Keys

- Sekarang kita akan belajar cara memutar ulang animasi , setelah animasi itu berputar
- Seperti kita akan memunculkan animasi setiap badges angka ini berubah



18. Re-triggering Animations via Keys

- Untuk mencapai hal ini kita perlu pergi ke komponen Badge
- Lalu berikan animasi ke span dengan motion.span

```
import {motion} from "framer-motion"

Codeium: Refactor | Explain | Generate JSDoc | X
export default function Badge({ caption }) {
  return <motion.span

    animate={{ scale : [1,1.2,1] }}
    transition={{ duration : 0.3 }}
    className="badge">{caption}</motion.span>
  }
}
```

18. Re-triggering Animations via Keys

- Agar komponen Badge bisa dianimasikan ulang, maka komponen Badge nya harus dirender ulang di komponen/page yang diperlukan
- Dalam hal ini challengeTabs komponen , cukup tambahkan property key pada komponen Badge dengan nilai dinamis, dalam hal ini child nya
- Ketika nilai pada key berubah, react akan menghancurkan instance komponen dan merender ulang komponen dengan key baru, sehingga animasi bisa dimulai kembali

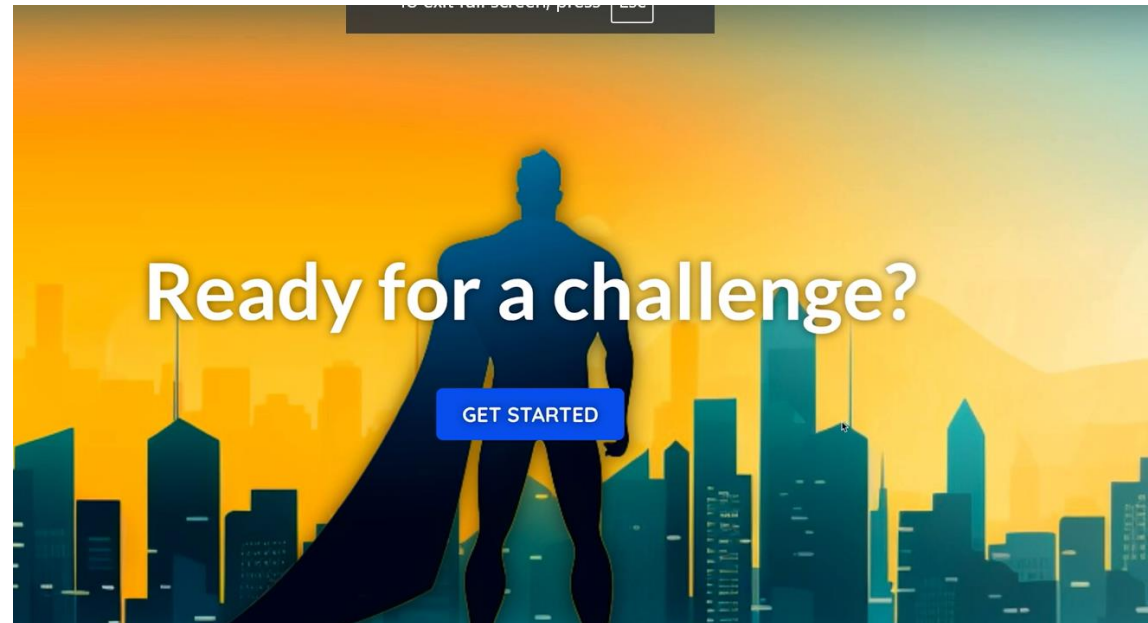
18. Re-triggering Animations via Keys

```
ChallengeTabs.jsx src/components/ChallengeTabs.jsx ChallengeTabs
1 import Badge from "../Badge.jsx";
2 import { motion } from "framer-motion";
Codeium: Refactor | Explain | Generate JSDoc | X
3 function Tab({ isSelected, onSelect, badgeCaption, children }) {
4   return (
5     <li>
6       <button
7         className={isSelected ? "selected" : undefined}
8         onClick={onSelect}
9       >
10        {children}
11        <Badge key={badgeCaption} caption={badgeCaption}></Badge>
12      </button>
13      {isSelected && (
14        <motion.div layoutId="tab-indicator" className="active-tab-indicator" />
15      )}
16    </li>
17  );
18 }
19
Codeium: Refactor | Explain | Generate JSDoc | X
```


19. Scroll Based Animations

19. Scroll Based Animations

- Kita melihat di halaman awal ada gambar hero lengkap dengan city nya, serta tulisan dan tombol get started
- Agar tidak membosankan kita akan menambahkan berbagai animasi disini



19. Scroll Based Animations

- Kita akan membuat efek parallax yang berarti setiap kita menscroll halaman, semua gambar ini akan bergerak di kecepatan yang berbeda untuk menghasilkan efek yang bagus dan menarik
- Sekarang kita perlu ke Welcome.jsx di halaman inilah kita akan menampilkan animasi

```
<div id="welcome-header-content">  
  <h1>Ready for a challenge?</h1>  
  <Link id="cta-link" to="/challenges">  
    Get Started  
  </Link>  
</div>
```

```
<img  
  src={cityImg}  
  alt="A city skyline touched by sunlight"  
  id="city-image"  
>
```

```
<img src={heroImg} alt="A superhero wearing a cape" id="hero-image" />  
</header>
```

19. Scroll Based Animations

```
export default function WelcomePage() {
  return (
    <>
      <header id="welcome-header">
        <div id="welcome-header-content">
          <h1>Ready for a challenge?</h1>
          <Link id="cta-link" to="/challenges">
            Get Started
          </Link>
        </div>
        <img
          src={cityImg}
          alt="A city skyline touched by sunlight"
          id="city-image"
        />
        <img src={heroImg} alt="A superhero wearing a cape" id="hero-image" />
      </header>
      <main id="welcome-content">
        <section>
          <h2>There's never been a better time.</h2>
          <p>
            With our platform, you can set, track, and conquer challenges at
            your own pace. Whether it's personal growth, professional
            achievements, or just for fun, we've got you covered.
          </p>
        </section>
        <section>
          <h2>Why Challenge Yourself?</h2>
          <p>
            Challenges provide a framework for growth. They push boundaries,
            test limits, and result in genuine progress. Here, we believe
            everyone has untapped potential, waiting to be unlocked.
          </p>
        </section>
        <section>
          <h2>Features</h2>
          <ul>
            <li>Custom challenge creation: Set the rules, define your pace.</li>
            <li>
              Track your progress: See your growth over time with our analytics
              tools.
            </li>
            <li>
              Community Support: Join our community and get motivated by peers.
            </li>
          </ul>
        </section>
      </main>
    </>
  );
}
```

19. Scroll Based Animations

```
</section>
<section>
  <h2>Join Thousands Embracing The Challenge</h2>
  <p>
    "I never realized what I was capable of until I set my first
    challenge here. It's been a transformative experience!" - Alex
    P.
  </p>
  { /* You can add more testimonials or even a carousel for multiple testimonials */ }
</section>
</main>
</>
);
}
```

19. Scroll Based Animations

- Kita ubah semua elemen yang ingin dianimasikan dengan motion

```
<motion.div id="welcome-header-content">
  <h1>Ready for a challenge?</h1>
  <Link id="cta-link" to="/challenges">
    Get Started
  </Link>
</motion.div>
```

```
<motion.img
  src={cityImg}
  alt="A city skyline touched by sunlight"
  id="city-image"
/>
<motion.img
  src={heroImg}
  alt="A superhero wearing a cape"
  id="hero-image"
/>
```

19. Scroll Based Animations

- Kita tidak akan menggunakan property `animate` secara asal karena kita ingin animasi terjadi bergantung kepada seberapa dalam kita melakukan scroll
- Kita harus mengimport dua hook yaitu `useScroll` dan `useTransform`
- `useScroll` menghasilkan objek berisi seberapa dalam kita melakukan scroll
- `useTransform` yang berguna mengubah dan mengatur nilai transformasi berdasarkan sesuatu seperti kedalaman scroll, yang bisa kita isi animasinya berdasarkan nilai transformasinya

19. Scroll Based Animations

- `useScroll` umumnya menghasilkan objek, seperti `scrollY` yang berfungsi mengembalikan nilai posisi scroll kita dalam sumbu vertikal dalam satuan pixel
- Kita juga punya scroll untuk sumbu horizontal
- Kita juga punya `scrollXProgress` dan `scrollYProgress` untuk mendapatkan nilai scroll yang relatif seperti 0-1 misalnya, dimana 1 berarti pengguna scroll sampai bawah dan 0 berarti pengguna tidak melakukan scroll sama sekali

19. Scroll Based Animations

- useTransform merupakan sebuah hook function dengan parameter pertama berisi nilai apa yang harus di transformasikan, dalam hal ini kita menggunakan scrollY, parameter kedua adalah array sebagai breakpoint
- Parameter ketiga berisi array yang isinya satuan animasi yang akan dijalankan ketika posisi breakpoint tercapai atau diantara breakpoint saat ini dan breakpoint setelahnya

19. Scroll Based Animations

- Di useTransform argumen pertama kita isi scrollY,
- Argumen kedua diisi array dari 0,200,dst
- Argumen ketiga kita isi array [1,0.5] dst. artinya ketika scroll di posisi 0 atau $0 < \text{scroll} < 200$, maka nilai animasinya adalah 1. 1 ini akan diisi kedalam property style (bukan animate) di tag motion, jika diisi sebagai nilai dari property opacity maka ketika scroll di posisi 0 atau $0 < \text{scroll} < 200$ nilai opacitynya adalah 1, begitupun ketika posisi scroll di 200, maka nilai opacitynya bernilai 0.5, dst
- Menggunakan useTransform kita tidak mengakibatkan komponen melakukan rerender
- Jangan lupa useTransform harus di assign kesebuah variabel, misalnya opacityCity untuk menganimasikan gambar kota

19. Scroll Based Animations

- useTransform secara tidak langsung mengubah nilai css dan secara otomatis menjalankan animasi

```
const { scrollY } = useScroll();
const opacityCity = useTransform(
  value: scrollY,
  inputRange: [0, 200, 300, 500],
  outputRange: [1, 0.5, 0.5, 0]
);
```

```
<motion.img
  style={{
    opacity: opacityCity,
  }}
  src={cityImg}
  alt="A city skyline touched by sunlight"
  id="city-image"
/>
```

- Kita juga akan menggunakan useTransform untuk elemen lainnya

19. Scroll Based Animations

```
const { scrollY } = useScroll();

const yCity = useTransform( value: scrollY, inputRange: [0, 200], outputRange: [0, -100]);
const opacityCity = useTransform(
  value: scrollY,
  inputRange: [0, 200, 300, 500],
  outputRange: [1, 0.5, 0.5, 0]
);

const yHero = useTransform( value: scrollY, inputRange: [0, 200], outputRange: [0, -150]);
const opacityHero = useTransform( value: scrollY, inputRange: [0, 300, 500], outputRange: [1, 1, 0]);

const scaleText = useTransform( value: scrollY, inputRange: [0, 300], outputRange: [1, 1.5]);
const yText = useTransform( value: scrollY, inputRange: [0, 200, 300, 500], outputRange: [0, 50, 50, 300]);
return {
```

```
<motion.div
  id="welcome-header-content"
  style={{
    scale: scaleText,
    y: yText,
  }}
>
  <h1>Ready for a challenge?</h1>
  <Link id="cta-link" to="/challenges">
    Get Started
  </Link>
</motion.div>
<motion.img
  style={{
    opacity: opacityCity,
    y: yCity,
  }}
  src={cityImg}
  alt="A city skyline touched by sunlight"
  id="city-image"
/>
<motion.img
  style={{
    y: yHero,
    opacity: opacityHero,
  }}
  src={heroImg}
  alt="A superhero wearing a cape"
  id="hero-image"
/>
</header>
```