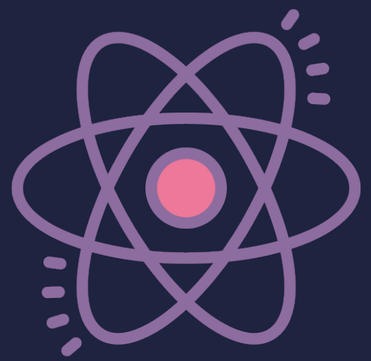
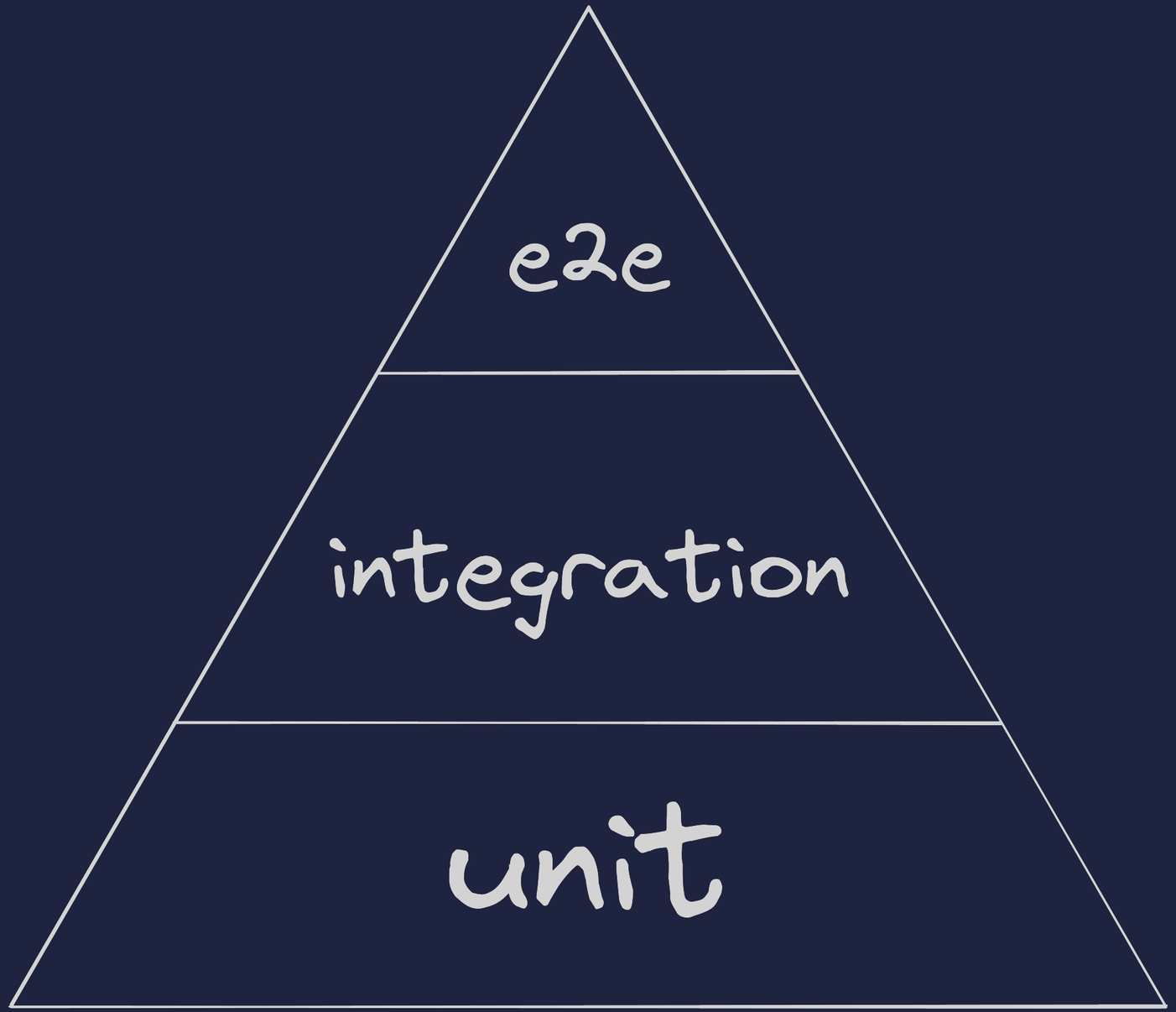
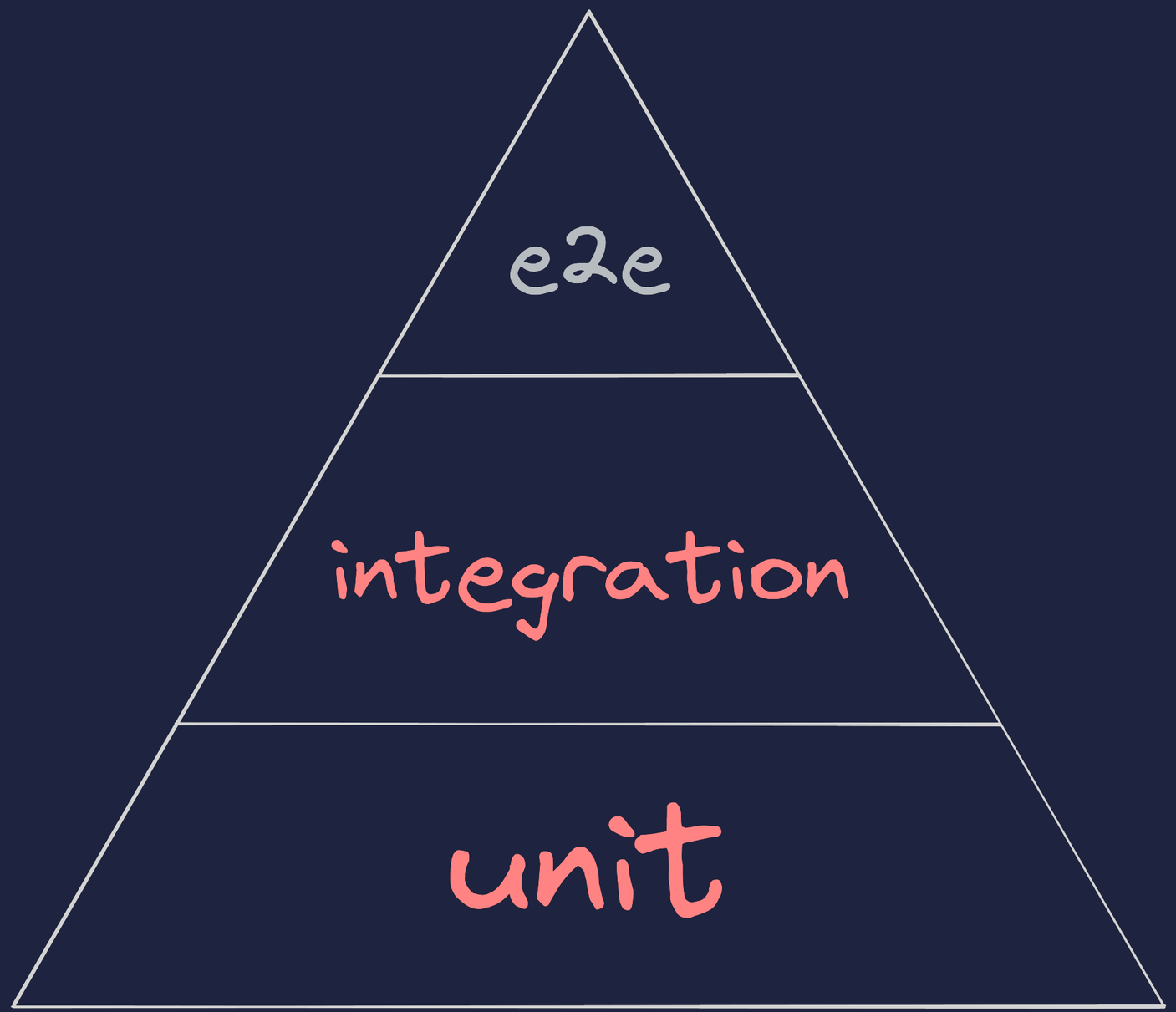
HOW TO WRITE UNBREAKABLE



by Samuel Farkas

OPINION ALERT


e2e
integration
unit


e2e
integration
unit

GOOD CODE IS THE CODE YOU CAN WRITE TEST FOR WITHOUT CRYING

- Simple and predictable
- Easy to maintain and understand
- Well-defined interfaces
- Loose coupling and separation of concerns
- Self-documenting

GREAT, BUT HOW?

- Leverage pure functions
- Make side-effects noticeable and easily fakeable
- Limit cyclomatic complexity
- Leverage single responsibility principle, separation of concerns and dependency injection (if it makes sense)

```tsx
export function TodoList() {
  const [todos, setTodos] = useState<Todo[]>([]);
  const [todoInput, setTodoInput] = useState("");
  const [selectedTodos, setSelectedTodos] = useState<Todo[]>([]);

  const [isLoading, setLoading] = useState(true);
  const [hasError, setError] = useState<Error>();

  useEffect(() => {
    fetch(API_URL).then((response) => {
      response
        .json()
        .then((data) => {
          setTodos(data);
          setLoading(false);
        })
        .catch((e) => {
          setError(e);
          setLoading(false);
        });
    });
  }, []);

  if (isLoading) {
    return <div>Loading...</div>;
  }

  if (hasError) {
    return <div>Error: {hasError.message}</div>;
  }


  const update = ({
    action,
    title,
    id,
  }: {
    action: "DELETE" | "ADD" | "COMPLETED";
    title?: string;
    id?: number;
  }) => {
    if (action === "DELETE") {
        // ...fetch..
        .then(() => {
          setTodos(response.data)
        })
    } else if (action === "ADD" && title) {
        fetch(API_URL, {
            body: JSON.stringify({ title: title.replace(/<[^>]*>?/gm, "")
            .replace(/<img[^>]*src="([^"]*)"[^>]*>/g, "[Image]") }),
        })
        .then((response) => {
            setTodos(response.data)
        })
    } else if (action === "COMPLETED" && id) {
        // ...fetch..
        .then((response) => {
            setTodos(response.data)
        })
    }
  };

  const select = (todo: Todo | Todo[]) => {
    setSelectedTodos((prevSelectedTodos) => {
      if (prevSelectedTodos.includes(id)) {
        return prevSelectedTodos.filter((selectedTodo) => selectedTodo !== id);
      }
      return [...prevSelectedTodos, id];
    });
  };


  return (
    <>
      <input
        type="text"
        value={todoInput}
        onChange={(e) => setTodoInput(e.target.value)}
      />
      <button
        onClick={() =>
          update({
            action: "ADD",
            title: todoInput,
          })
        }
      >
        Add
      </button>
      <ul>
        {todos.map((todo) => (
          <li key={todo.id}>
            <input
              type="checkbox"
              checked={todo.completed}
              onChange={() =>
                update({
                  action: "COMPLETED",
                  id: todo.id,
                })
              }
            />
            {todo.title}
            <button
              onClick={() =>
                update({
                  action: "DELETE",
                })
              }
            >
              Delete
            </button>
          </li>
        ))}
      </ul>
    </>
  );
}
```

```
useEffect(() => {
  fetch(API_URL).then((response) => {
    response
      .json()
      .then((data) => {
        setTodos(data);
        setLoading(false);
      })
      .catch((e) => {
        setError(e);
        setLoading(false);
      });
  });
}, []);

const update = ({
    action,
    title,
    id,
  }: {
    action: "DELETE" | "ADD" | "COMPLETED";
    title?: string;
    id?: number;
  }) => {
    if (action === "DELETE") {
        // ...fetch..
        .then(() => {
          setTodos(response.data)
        })
    } else if (action === "ADD" && title) {
        // ...fetch..
          setTodos(response.data)
        })
    } else if (action === "COMPLETED" && id) {
        // ...fetch..
        .then((response) => {
            setTodos(response.data)
        })
    }
 };
```

- Directly embedded side-effect into component logic
- Almost impossible to fake
- Harder to maintain

```
useEffect(() => {
  fetch(API_URL).then((response) => {
    response
      .json()
      .then((data) => {
        setTodos(data);
        setLoading(false);
      })
      .catch((e) => {
        setError(e);
        setLoading(false);
      });
  });
}, []);

const update = ({
    action,
    title,
    id,
}: {
    action: "DELETE" | "ADD" | "COMPLETED";
    title?: string;
    id?: number;
}) => {
    if (action === "DELETE") {
         // ...fetch..
        .then(() => {
          setTodos(response.data)
        })
    } else if (action === "ADD" && title) {
        fetch(API_URL, {
            body: JSON.stringify({ title: title.replace(/<[^>]*>?/gm, "")
            .replace(/<img[^>]*src="([^"]*)"[^>]*>/g, "[Image]") }),
        })
        .then((response) => {
            setTodos(response.data)
        })
    } else if (action === "COMPLETED" && id) {
        // ...fetch..
        .then((response) => {
            setTodos(response.data)
        })
    }
};
```

- Hard to read
- Higher cyclomatic complexity
- Violating single responsibility principle and separation of concerns (UI updates, Business Logic, Network calls)
- Directly embedded side-effects in function's logic

```
  const select = (id: number) => {
    setSelectedTodos((prevSelectedTodos) => {
      if (prevSelectedTodos.includes(id)) {
        return prevSelectedTodos.filter((selectedTodo) => selectedTodo !== id);
      }
      return [...prevSelectedTodos, id];
    });
  };


  return (
    <>
      <input
        type="text"
        value={todoInput}
        onChange={(e) => setTodoInput(e.target.value)}
      />
      <button
        onClick={() =>
          update({
            action: "ADD",
            title: todoInput,
          })
        }
      >
        Add
      </button>
      <ul>
        {todos.map((todo) => (
          <li key={todo.id}>
            <input
              type="checkbox"
              checked={todo.completed}
              onChange={() =>
                update({
                  action: "COMPLETED",
                  id: todo.id,
                })
              }
            />
            {todo.title}
            <button
              onClick={() =>
                update({
                  action: "DELETE",
                })
              }
            >
              Delete
            </button>
          </li>
        ))}
      </ul>
    </>
  );
}
```

- Contains side-effect that makes it untestable
- Handles untestable internal logic

```
  const select = (todo: Todo | Todo[]) => {
     setSelectedTodos((prevSelectedTodos) => {
      if (prevSelectedTodos.includes(id)) {
        return prevSelectedTodos.filter((selectedTodo) => selectedTodo !== id);
      }
      return [...prevSelectedTodos, id];
    });
  }


  return (
    <>
      <input
        type="text"
        value={todoInput}
        onChange={(e) => setTodoInput(e.target.value)}
      />
      <button
        onClick={() =>
          update({
            action: "ADD",
            title: todoInput,
          })
        }
      >
        Add
      </button>
      <ul>
        {todos.map((todo) => (
          <li key={todo.id}>
            <input
              type="checkbox"
              checked={todo.completed}
              onChange={() =>
                update({
                  action: "COMPLETED",
                  id: todo.id,
                })
              }
            />
            {todo.title}
            <button
              onClick={() =>
                update({
                  action: "DELETE",
                })
              }
            >
              Delete
            </button>
          </li>
        ))}
      </ul>
    </>
  );
}
```

- Hard to read
- Violates single responsibility principle

SO, CAN WE FIX IT?

```tsx
const [todos, setTodos] = useState<Todo[]>([]);
const [todoInput, setTodoInput] = useState("");
const [selectedTodos, setSelectedTodos] = useState<Todo[]>([]);

const [isLoading, setLoading] = useState(true);
const [hasError, setError] = useState<Error>();

useEffect(() => {
  fetch(API_URL).then((response) => {
    response
      .json()
      .then((data) => {
        setTodos(data);
        setLoading(false);
      })
      .catch((e) => {
        setError(e);
        setLoading(false);
      });
  });
}, []);
```

```tsx
const {
  data,
  isLoading: areTodosLoading,
  isError: hasTodosError,
} = useTodosQuery();

const [addTodo, { isLoading: isAddingTodo }] = useAddTodoMutation();
const [deleteTodo, { isLoading: isDeletingTodo }] = useDeleteTodoMutation();
const [updateTodo, { isLoading: isUpdatingTodo }] = useUpdateTodoMutation();
```

```tsx
const [todos, setTodos] = useState<Todo[]>([]);
const [todoInput, setTodoInput] = useState("");
const [selectedTodos, setSelectedTodos] = useState<Todo[]>([]);

const [isLoading, setLoading] = useState(true);
const [hasError, setError] = useState<Error>();

useEffect(() => {
  fetch(API_URL).then((response) => {
    response
      .json()
      .then((data) => {
        setTodos(data);
        setLoading(false);
      })
      .catch((e) => {
        setError(e);
        setLoading(false);
      });
  });
}, []);
```

```tsx
const {
  data,
  isLoading: areTodosLoading,
  isError: hasTodosError,
} = useTodosQuery();

const [addTodo, { isLoading: isAddTodoLoading }] = useAddTodoMutation();
const [deleteTodo, { isLoading: isDeleteTodoLoading }] =
  useDeleteTodoMutation();
const [updateTodo, { isLoading: isUpdateTodoLoading }] =
  useUpdateTodoMutation();
```

- Moved data management concern into separate hook (RTKQ, react-query or whatever twitter likes today)
- Easily fakeable (mockable and stubable)
- Easy to understand, read and declarative (idc about inner implementation)

```
const update = ({
    action,
    title,
    id,
}: {
    action: "DELETE" | "ADD" | "COMPLETED";
    title?: string;
    id?: number;
}) => {
    if (action === "DELETE") {
         // ...fetch..
        .then(() => {
          setTodos(response.data)
        })
    } else if (action === "ADD" && title) {
        fetch(API_URL, {
            body: JSON.stringify({ title: title.replace(/<[^>]*>?/gm, "")
            .replace(/<img[^>]*src="([^"]*)"[^>]*>/g, "[Image]") }),
        })
        .then((response) => {
            setTodos(response.data)
        })
    } else if (action === "COMPLETED" && id) {
        // ...fetch..
        .then((response) => {
            setTodos(response.data)
        })
    }
};
```

```
const handleAddTodo = (text: string) => {
  const todo = parseMedia(sanitizeText(text));
  return addTodo(todo).catch(handleApiError);
};

const handleDeleteTodo = (id: number) => deleteTodo(id).catch(handleApiError);
const handleCompletedTodo = (id: number) =>
  updateTodo({ id, completed: true }).catch(handleApiError);
```

```
const update = ({
    action,
    title,
    id,
}: {
    action: "DELETE" | "ADD" | "COMPLETED";
    title?: string;
    id?: number;
}) => {
    if (action === "DELETE") {
         // ...fetch..
        .then(() => {
          setTodos(response.data)
        })
    } else if (action === "ADD" && title) {
        fetch(API_URL, {
            body: JSON.stringify({ title: title.replace(/<[^>]*>?/gm, "")
            .replace(/<img[^>]*src="([^"]*)"[^>]*>/g, "[Image]") }),
        })
        .then((response) => {
            setTodos(response.data)
        })
    } else if (action === "COMPLETED" && id) {
        // ...fetch..
        .then((response) => {
            setTodos(response.data)
        })
    }
};
```

```
const handleAddTodo = (text: string) => {
  const todo = parseMedia(sanitizeText(text));
  return addTodo(todo).catch(handleApiError);
};

const handleDeleteTodo = (id: number) => deleteTodo(id).catch(handleApiError);
const handleCompletedTodo = (id: number) =>
  updateTodo({ id, completed: true }).catch(handleApiError);
```

- Easily noticable isolated side-effects
- Self-documenting and readable
- Simple and predictable

```
fetch(API_URL, {
  body: JSON.stringify({
    title: title
      .replace(/<[^>]*>?/gm, "")
      .replace(/<img[^>]*src="([^"]*)"[^>]*>/g, "[Image]"),
  }),
});
```

```
const handleAddTodo = (text: string) => {
  const todo = parseMedia(sanitizeText(text));
  return addTodo(todo).catch(handleApiError);
};
```

- Extracted logic to pure functions
- Easily testable and declarative
- Simple and easy to read

```
  return (
    <>
      <input
        type="text"
        value={todoInput}
        onChange={(e) => setTodoInput(e.target.value)}
      />
      <button
        onClick={() =>
          update({
            action: "ADD",
            title: todoInput,
          })
        }
      >
        Add
      </button>
      <ul>
        {todos.map((todo) => (
          <li key={todo.id}>
            <input
              type="checkbox"
              checked={todo.completed}
              onChange={() =>
                update({
                  action: "COMPLETED",
                  id: todo.id,
                })
              }
            />
            {todo.title}
            <button
              onClick={() =>
                update({
                  action: "DELETE",
                })
              }
            >
              Delete
            </button>
          </li>
        ))}
      </ul>
    </>
  );
}
```

```
return (
  <>
    <AddTodoForm onAddTodo={handleAddTodo} />
    <List
      data={data}
      onCompleted={handleCompleteTodo}
      onDeleted={handleDeleteTodo}
      onSelect={handleSelectTodo}
    />
  </>
);
```

```
  return (
    <>
      <input
        type="text"
        value={todoInput}
        onChange={(e) => setTodoInput(e.target.value)}
      />
      <button
        onClick={() =>
          update({
            action: "ADD",
            title: todoInput,
          })
        }
      >
        Add
      </button>
      <ul>
        {todos.map((todo) => (
          <li key={todo.id}>
            <input
              type="checkbox"
              checked={todo.completed}
              onChange={() =>
                update({
                  action: "COMPLETED",
                  id: todo.id,
                })
              }
            />
            {todo.title}
            <button
              onClick={() =>
                update({
                  action: "DELETE",
                })
              }
            >
              Delete
            </button>
          </li>
        ))}
      </ul>
    </>
  );
}
```

```
return (
  <>
    <AddTodoForm onAddTodo={handleAddTodo} />
    <List
      data={data}
      onCompleted={handleCompleteTodo}
      onDeleted={handleDeleteTodo}
      onSelect={handleSelectTodo}
    />
  </>
);
```

- Leverage single responsibility principle
- Presentational components might be considered pure functions
- Leverages dependency injection = fakeable, spyable
- Simple and easy to read

```tsx
export function TodoList() {
  const [selectedTodos, setSelectedTodos] = useState([]);

  const {
    data,
    isLoading: areTodosLoading,
    isError: hasTodosError,
  } = useTodosQuery();

  const [addTodo, { isLoading: isAddTodoLoading }] = useAddTodoMutation();
  const [deleteTodo, { isLoading: isDeleteTodoLoading }] =
    useDeleteTodoMutation();
  const [updateTodo, { isLoading: isUpdateTodoLoading }] =
    useUpdateTodoMutation();

  const handleAddTodo = (text: string) => {
    const todo = parseMedia(sanitizeText(text));
    return addTodo(todo).catch(handleApiError);
  };

  const handleDeleteTodo = (id: number) => deleteTodo(id).catch(handleApiError);
  const handleCompleteTodo = (id: number) =>
    updateTodo({ id, completed: true }).catch(handleApiError);

  const handleSelectTodo = (ids: number[]) => {
    // ...handle select
  };

  return (
    <>
      <AddTodoForm onAddTodo={handleAddTodo} />
      <List
        data={data}
        onCompleted={handleCompleteTodo}
        onDeleted={handleDeleteTodo}
        onSelect={handleSelectTodo}
      />
    </>
  );
}
```

```tsx
export function TodoList() {
  const [selectedTodos, setSelectedTodos] = useState([]);

  const {
    data,
    isLoading: areTodosLoading,
    isError: hasTodosError,
  } = useTodosQuery();

  const [addTodo, { isLoading: isAddTodoLoading }] = useAddTodoMutation();
  const [deleteTodo, { isLoading: isDeleteTodoLoading }] =
    useDeleteTodoMutation();
  const [updateTodo, { isLoading: isUpdateTodoLoading }] =
    useUpdateTodoMutation();

  const handleAddTodo = (text: string) => {
    const todo = parseMedia(sanitizeText(text));
    return addTodo(todo).catch(handleApiError);
  };

  const handleDeleteTodo = (id: number) => deleteTodo(id).catch(handleApiError);
  const handleCompleteTodo = (id: number) =>
    updateTodo({ id, completed: true }).catch(handleApiError);

  const handleSelectTodo = (ids: number[]) => {
    // ...handle select
  };

  return (
    <>
      <AddTodoForm onAddTodo={handleAddTodo} />
      <List
        data={data}
        onCompleted={handleCompleteTodo}
        onDeleted={handleDeleteTodo}
        onSelect={handleSelectTodo}
      />
    </>
  );
}
```

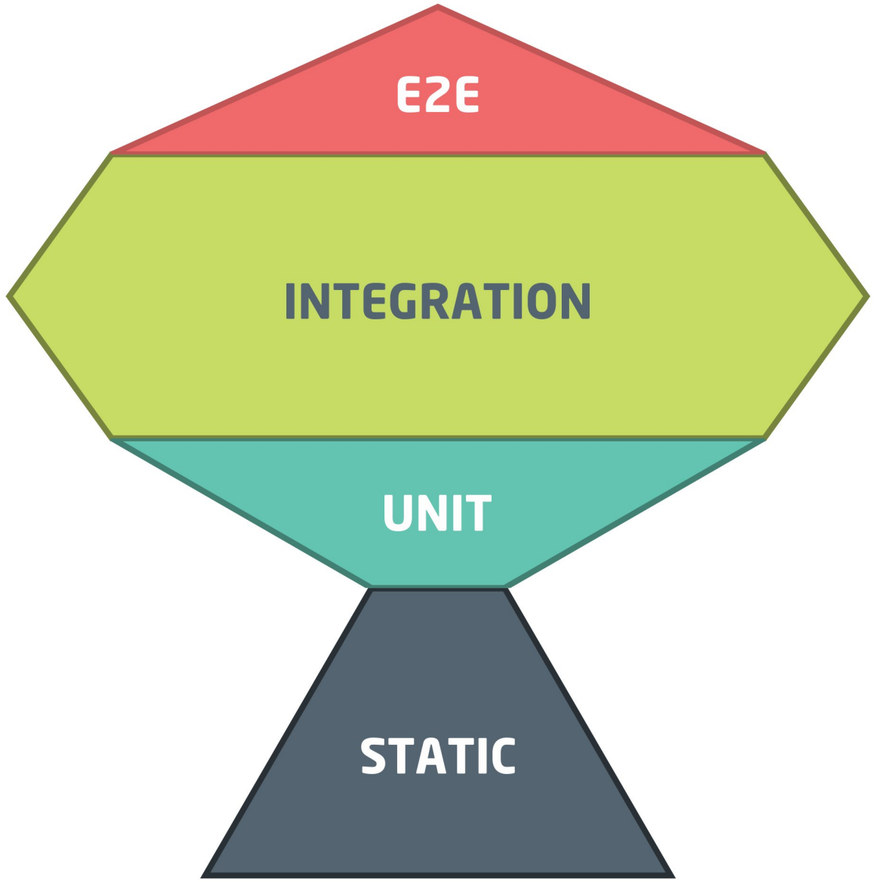
- ✅ Unit tests for parser and sanitizer
- ✅ Component test for AddTodoForm
- ✅ Component test for List


Erin
@erinfranmc
expect(umbrellaOpens).toBe(true)
tests: 1 passed, 1 total
all tests passed
6:06 PM · 10. 7. 2019

```tsx
export function TodoList() {
  const [selectedTodos, setSelectedTodos] = useState([]);

  const {
    data,
    isLoading: areTodosLoading,
    isError: hasTodosError,
  } = useTodosQuery();

  const [addTodo, { isLoading: isAddTodoLoading }] = useAddTodoMutation();
  const [deleteTodo, { isLoading: isDeleteTodoLoading }] =
    useDeleteTodoMutation();
  const [updateTodo, { isLoading: isUpdateTodoLoading }] =
    useUpdateTodoMutation();

  const handleAddTodo = (text: string) => {
    const todo = parseMedia(sanitizeText(text));
    return addTodo(todo).catch(handleApiError);
  };

  const handleDeleteTodo = (id: number) => deleteTodo(id).catch(handleApiError);
  const handleCompleteTodo = (id: number) =>
    updateTodo({ id, completed: true }).catch(handleApiError);

  const handleSelectTodo = (ids: number[]) => {
    // ...handle select
  };

  return (
    <>
      <AddTodoForm onAddTodo={handleAddTodo} />
      <List
        data={data}
        onCompleted={handleCompleteTodo}
        onDeleted={handleDeleteTodo}
        onSelect={handleSelectTodo}
      />
    </>
  );
}
```

- ✅ Unit tests for parser and sanitizer
- ✅ Component test for AddTodoForm
- ✅ Component test for List
- ✅ Integration test for TodoList

The Testing Trophy



e2e

flows/journeys

real implementations

integration

behaviors / interactions

lot of faking (mocking/stubbing)

unit

state

ideally no faking, pure functions


e2e
flows/journeys
real implementations
integration
behaviors / interactions
lot of faking (mocking/stubbing)
unit
state
ideally no faking, pure functions


e2e
flows/journeys
real implementations
integration
behaviors / interactions
lot of faking (mocking/stubbing)
unit
state
ideally no faking, pure functions

```tsx
export function useList() {
  const [selectedTodos, setSelectedTodos] = useState<number[]>([]);

  const {
    data,
    isLoading: areTodosLoading,
    isError: hasTodosError,
  } = useTodosQuery();

  const [addTodo] = useAddTodoMutation();
  const [deleteTodo] = useDeleteTodoMutation();
  const [updateTodo] = useUpdateTodoMutation();

  const addTodo = (text: string) => {
    const todo = parseMedia(sanitizeText(text));
    return addTodo(todo).catch(handleApiError);
  };

  const deleteTodo = (id: number) => deleteTodo(id).catch(handleApiError);

  const completeTodo = (id: number) =>
    updateTodo({ id, completed: true }).catch(handleApiError);

  const toggleSelectedTodo = (id: number) => {
    setSelectedTodos((prevSelectedTodos) => {
      if (prevSelectedTodos.includes(id)) {
        return prevSelectedTodos.filter((selectedTodo) => selectedTodo !== id);
      }
      return [...prevSelectedTodos, id];
    });
  };

  return {
    todos: data,
    selectedTodos,
    isLoading: areTodosLoading,
    hasError: hasTodosError,
    actions: {
      addTodo,
      deleteTodo,
      updateTodo,
      completeTodo,
      toggleSelectedTodo,
    },
  };
}
```

- Internal state becomes testable

```
describe("useList hook", () => {
  it("should toggle selectedTodos correctly", () => {
    const { result } = renderHook(() => useList());

    // Initially, selectedTodos should be empty
    expect(result.current.selectedTodos).toEqual([]);

    // Add an item to selectedTodos
    act(() => {
      result.current.actions.toggleSelectedTodo(1);
    });
    expect(result.current.selectedTodos).toEqual([1]);

    act(() => {
      result.current.actions.toggleSelectedTodo(2);
    });
    expect(result.current.selectedTodos).toEqual([1, 2]);

    // Remove the first item from selectedTodos
    act(() => {
      result.current.actions.toggleSelectedTodo(1);
    });
    expect(result.current.selectedTodos).toEqual([2]);

    act(() => {
      result.current.actions.toggleSelectedTodo(2);
    });
    expect(result.current.selectedTodos).toEqual([]);
  });
});
```

```
export function TodoList() {
  const { todos, selectedTodos, actions, isLoading, hasError } = useList();

  if (isLoading) return <div>Loading...</div>;
  if (hasError) return <div>Error...</div>;

  return (
    <>
      <AddTodoForm onAddTodo={actions.addTodo} />
      <List
        data={todos}
        selectedTodosIds={selectedTodos}
        onCompleted={actions.completeTodo}
        onDeleted={actions.deleteTodo}
        onSelect={actions.toggleSelectedTodo}
      />
    </>
  );
}
```


e2e
flows/journeys
real implementations
integration
behaviors / interactions
lot of faking (mocking/stubbing)
unit
state
ideally no faking, pure functions

WAS IT WORTH IT, THO?

e2e

flows/journeys

real implementations

integration

behaviors / interactions

lot of faking (mocking/stubbing)

unit

state

ideally no faking, pure functions

```ts
export function useVehicleAlbumImageSelection() {
  const [selectedImagesIds, setSelectedImagesIds] = useState<Set<string>>(
    new Set(),
  );

  const getSelectedImages = () => Array.from(selectedImagesIds);

  const isImageSelected = (id: string) => selectedImagesIds.has(id);

  const selectImage = (id: string) => {
    setSelectedImagesIds((prev) => new Set(prev).add(id));
  };

  const selectImagesInBulk = (ids: string[]) => {
    setSelectedImagesIds(new Set(ids));
  };

  const deselectImage = (id: string) => {
    if (selectedImagesIds.has(id)) {
      setSelectedImagesIds((prevSet) => {
        const updatedSet = new Set(prevSet);
        updatedSet.delete(id);
        return updatedSet;
      });
    }
  };

  const clearImageSelection = () => {
    setSelectedImagesIds(new Set());
  };

  const toggleImageSelection = (id: string) => {
    setSelectedImagesIds((prevSet) => {
      const updatedSet = new Set(prevSet);
      if (updatedSet.has(id)) {
        updatedSet.delete(id);
      } else {
        updatedSet.add(id);
      }
      return updatedSet;
    });
  };

  return {
    getSelectedImages,
    selectImage,
    deselectImage,
    isImageSelected,
    selectImagesInBulk,
    toggleImageSelection,
    clearImageSelection,
  };
}
```

```ts
describe("useVehicleAlbumImageSelection", () => {
  //...
  it("should select a single image", () => {
    const { result } = renderHook(() => useVehicleAlbumImageSelection());

    act(() => {
      result.current.selectImagesInBulk(["image1"]);
    });

    expect(result.current.getSelectedImages()).toHaveLength(1);
    expect(result.current.getSelectedImages()).toContain("image1");
  });

  it("should select multiple images in bulk", () => {
    const { result } = renderHook(() => useVehicleAlbumImageSelection());

    act(() => {
      result.current.selectImagesInBulk(["image1", "image2", "image3"]);
    });

    expect(result.current.getSelectedImages()).toHaveLength(3);
    expect(result.current.getSelectedImages()).toEqual(
      expect.arrayContaining(["image1", "image2", "image3"]),
    );
  });

  it("should deselect an image", () => {
    const { result } = renderHook(() => useVehicleAlbumImageSelection());

    act(() => {
      result.current.selectImagesInBulk(["image1"]);
      result.current.toggleImageSelection("image1");
    });

    expect(result.current.getSelectedImages()).toHaveLength(0);
  });

  it("should handle selecting the same image multiple times correctly", () => {
    const { result } = renderHook(() => useVehicleAlbumImageSelection());

    act(() => {
      result.current.selectImagesInBulk(["image1", "image1", "image1"]);
    });

    expect(result.current.getSelectedImages()).toEqual(["image1"]);
  });
  //...
});
```

WELL-DEFINED INTERFACES

- Interface defines contract for interaction (components, functions, endpoints)
  - Good interface makes code easier to test
  - Good interface makes code more readable
  - Good interface makes code less error-prone

BAD INTERFACE

```
interface ListProps {
  data: any[];
  onChange: (data: any) => void;
  onAction: Function;
}
```

- Poorly typed
- Ambigous onChange and onAction function, what is the purpose of it and what im supposed to provide there?

BETTER INTERFACE

```tsx
interface ListProps {
  todos: Todo[];
  onAddTodo: (content: string) => void;
  onUpdateTodo: (id: number, updatedContent: string) => void;
  onToggleTodoCompletion: (id: number) => void;
  onDeleteTodo: (id: number) => void;
}
```

- Strongly typed
- Uses concise naming
- Has specific function signatures
- Developer knows what to provide to component without checking internal implementation
- Testable
- Self-documenting

SELF-DOCUMENTING CODE

- Testable code has high chance of being self-documenting
  - Tests are part of the documentation
  - Leverage concise naming
- If you need to write comment to explain what code does = 
  - Write comments to explain decisions

DISCUSSION