PRACTICAL REACT WITH TYPESCRIPT

bouvet

https://tinyurl.com/practical-react

Setup

https://tinyurl.com/practical-react

https://github.com/rudfoss/practical-react-with-typescript

Our development environment

- Nx.dev powered repository configured as integrated monorepo
  - Code generators
  - Project dependency tracking
  - Task orchestration
- Visual Studio Code some custom extensions and configurations
- Linting using ESlint with custom rules from Nx and unicorn, auto-fix on save where possible
- Formatting using prettier with “some” EditorConfig, auto-format on save
- React front-end bundled with Vite (generated using @nx/react)
  - Vite dev server with hot-reloading
  - Testing using Jest and testing-library/react
  - End-to-end testing using Playwright
- NestJs backend API (generated using @nx/nest)
  - Auto-generated OpenAPI specification
  - Auto client-generation using NSwag (npm package + .NET 6+)

Agenda

1. What is React
2. Props, events and state
3. Lists and loops
4. Organizing code
5. Styling
6. Routing
7. Services / Contexts
8. Optimize rendering
9. Server communication
10. Building the application (a bunch of tasks)

Jump to references 🚀

bouvet

What is React?

> A JavaScript library for building user interfaces
>
> -reactjs.org

bouvet

Anatomy of React

The component

bouvet

Anatomy of React

```
function GreetPerson(props) {
  const userData = useContext(User)
  const [greeting, setGreeting] = useState("Hello there,")

  return (
    <div>
      <h1>{greeting} {userData.name}</h1>
      <input
        type="text"
        value={greeting}
        onChange={(event) => setGreeting(event.currentTarget.value)}
        disabled={props.disabled}
        />
    </div>
  )
}
```

Props

State

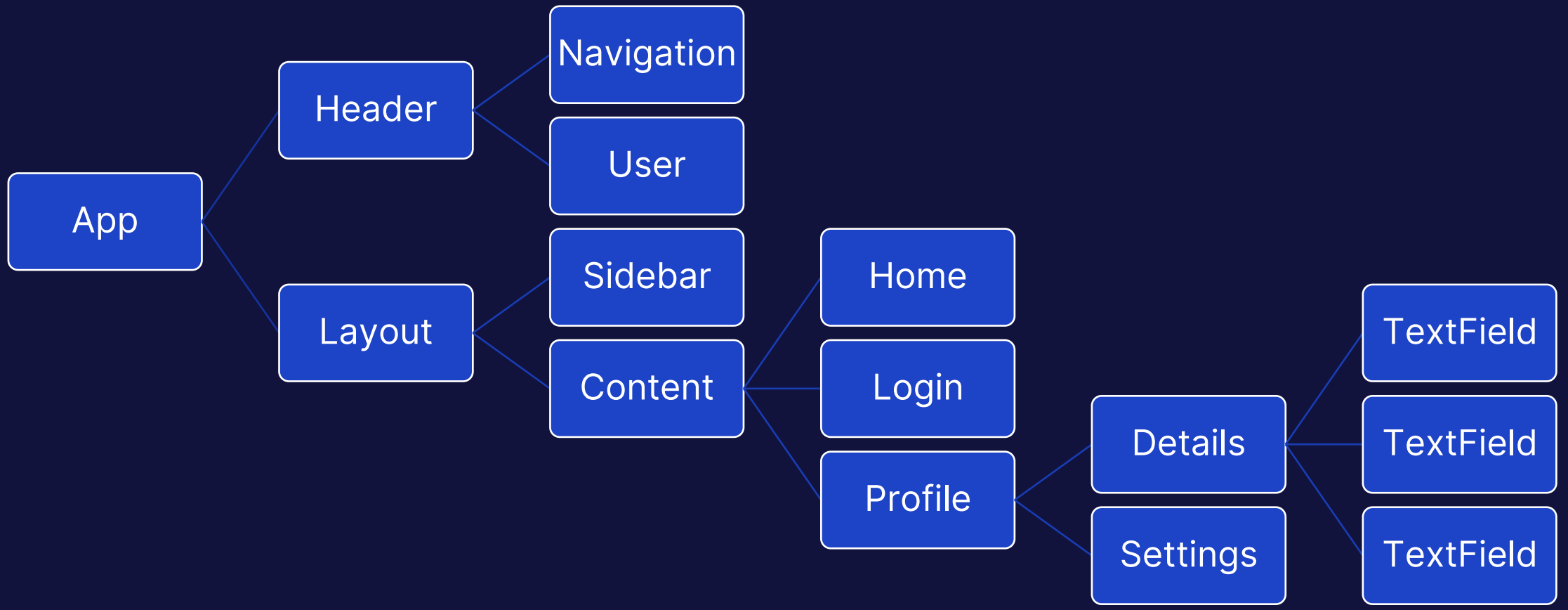
Context

JSX

bouvet

Anatomy of React

The component tree



bouvet

Props

- “Arguments” to a component.
- The component re-renders if the props change.
- Props are «packed» into an object and passed as the first argument to the component function.

```
<MyComponent foo="hey" bar={42} baz>Hello</MyComponent>
```

```
props = {
  foo: "Hey",
  bar: 42,
  baz: true,
  children: "Hello"
}
```

bouvet

Header

- Create a basic header with a <h1> title.
- Allow setting the header title from the “outside” using a “prop”.

bouvet

Events

- Events are actions triggered by the user (or browser)
- Register a function to act on the event.

```
const onButtonClicked = (event: MouseEvent<HTMLButtonElement>) => {
  const buttonText = event.currentTarget.textContent
  console.log(`${buttonText} clicked`)
}

return <button onClick={onButtonClicked}>Click me</button>
```

TextField

- Create a component where the user can enter a single line of text.
- Add a “prop” to define the label of the field.
- Clicking the label should put focus in the text field (accessibility).
- Add “prop” to disable the field.

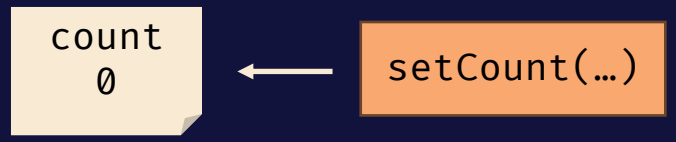
bouvet

State

- A “live” value the component can keep track of and update.
- “Setting” the value re-renders the component.

```
const [count, setCount] = useState(0)

return <button onClick={() => setCount(count + 1)}>{count}</button>
```



bouvet

Control header using text field

- Use the text entered in the text field to control the header text.
- How do we get the text of the text field over to the header?

bouvet

Hoisting state

- Lift the state “up” from a component to a parent.
- Allows sharing state between components.

<> PasswordField

- Create a component where the user can enter a password.
- Add a configurable label that focuses on the input when clicked.
- Add an optional “prop” for controlling the maximum length of the password.
- Add “prop” to disable the field.

<> CheckboxField

- Create a component where the use can check a box.
- Add a configurable label that focuses on the input when clicked.
- Add “prop” to disable the field.
- Ensure the field is reusable like the TextField.

<> LoginForm

- Create a form where the user must enter a username and password.
- Add a checkbox to show the password value.
- Add a button to log in. It should be disabled until there is any text in both the username and password fields.
- Add “onLogin” prop that takes a function which receives the username and password when the clicks the “Log in” button.
- Add prop to disable the entire form.

Lists

- Render lists of items using the same components.
- Must keep track of which components corresponds to which item.
- Rules of keys.

bouvet

<> StaticGroupTable

- Copy groups from the API and list them in a table with at least the id and display name.
- Allow moving groups up or down.
- Allow deleting groups.
- Bonus: Disable buttons that don’t make sense.

Tests

- Unit tests using Jest (or Vitest)+ testing-library/react
- End-to-end tests using Playwright

bouvet

Test

- Write basic unit tests for the <Header/> that verifies the provided text is rendered in an <h1> tag.
- Write an end-to-end test that verifies that the page loads and the app starts.

Organizing our code

- Apps vs libraries
  - An app can be built and deployed.
  - A library is used by one or more apps.
- Group libraries by domain
  - User components
  - Group components
  - Fields
  - UI
  - ++
- Utilize index files (barrel files) to control public vs private APIs.

<> ChoiceField

- Create a component that allows the user to select one item from a list of items.
- Place the component in the “fields” library.
- Provide a way to choose between radio buttons and a drop-down for selection.
- Provide a configurable label.
- Add prop to disable the field.

Styling

- Style-prop on elements
- CSS files
- CSS modules
- CSS preprocessors
  - SASS, LESS, PostCSS
- CSS-in-js
  - styled-components, emotion, linaria, griffel

Style header

- Add some spacing and a border to the header.

bouvet

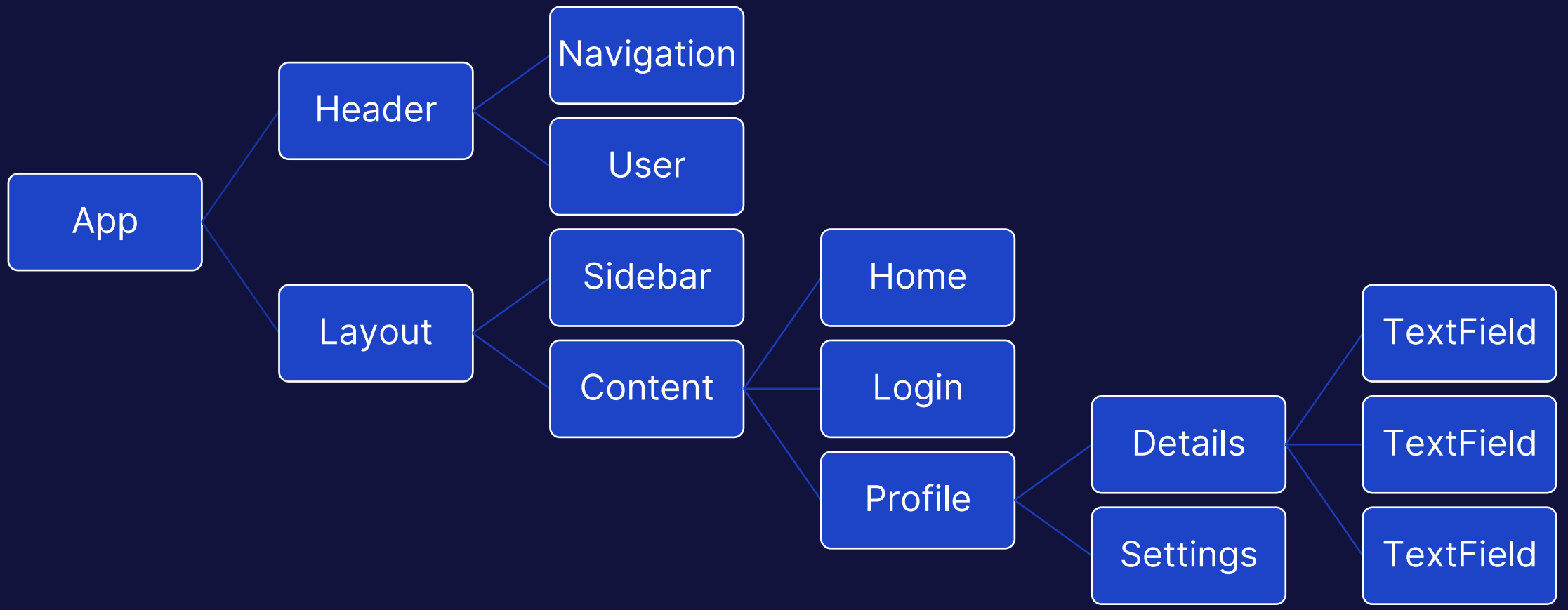
Style fields

- TextField/PasswordField
  - Move label above the input.
  - Add 16px top/bottom, 8px left/right margin around the entire field (label and input).
  - Use all available width for the input.
- ChoiceField
  - Move label above select.
  - Add the same padding as the TextField.
  - Add some spacing around each radio button choice.
  - Vertically center the radio button and the label next to each other.
- CheckboxField
  - Add the same padding as the TextField.
  - Vertically center the checkbox and the label next to each other.
  - Add some spacing between the checkbox and the label.

Routing

- Selectively render pieces of our application tree based on the users current path.
- Extract parameters from the path and use them to select what content to render.
- Utilize History API in the browser.

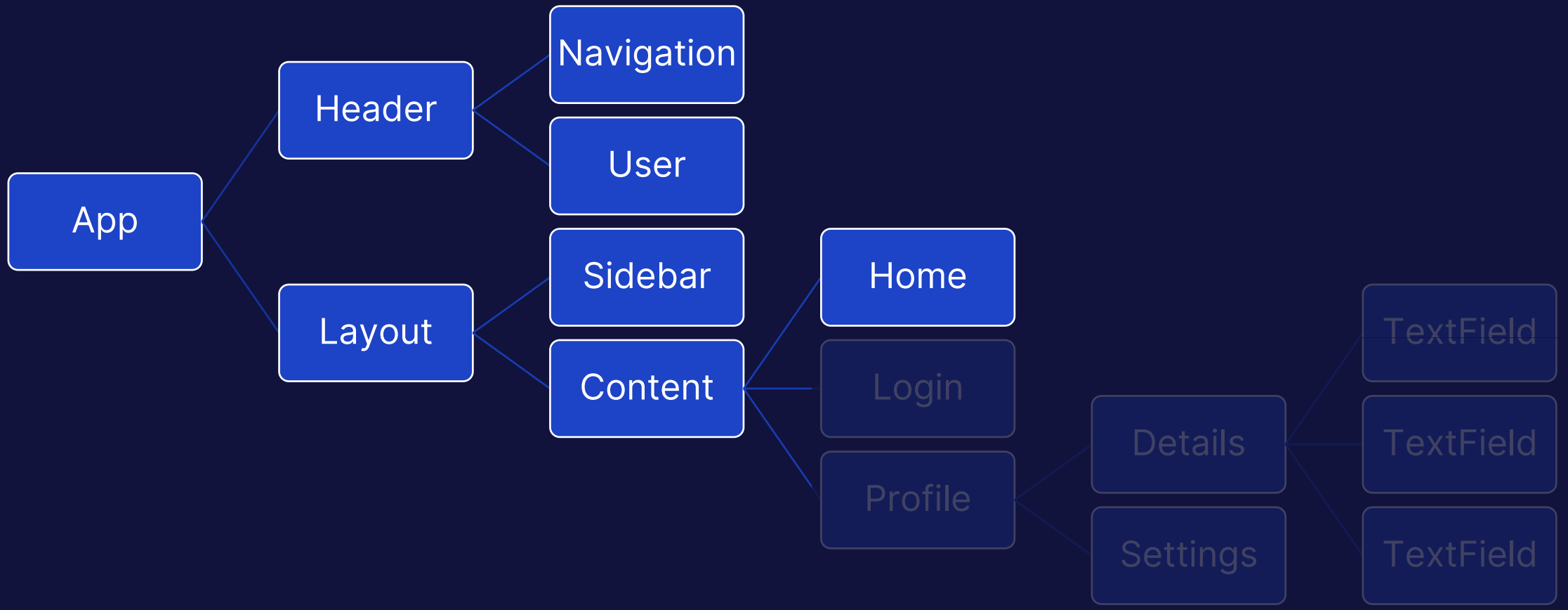
Routing



bouvet

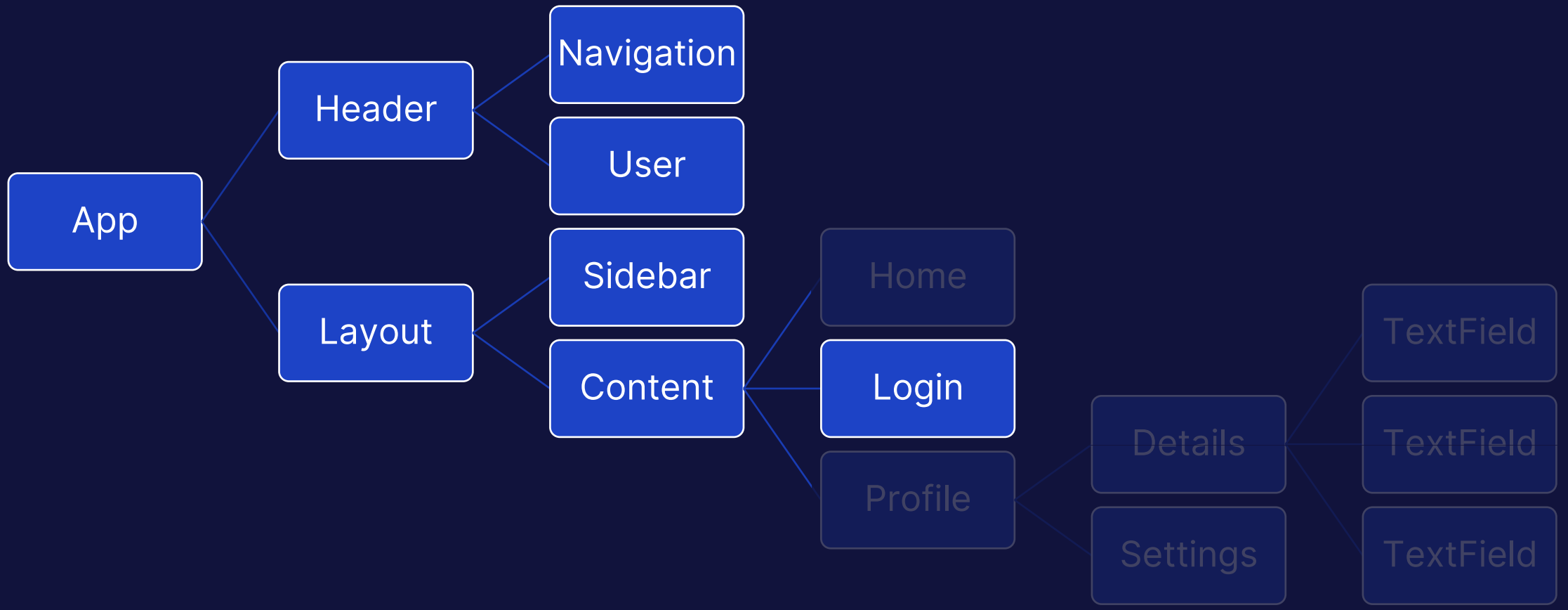
Routing

/home



Routing

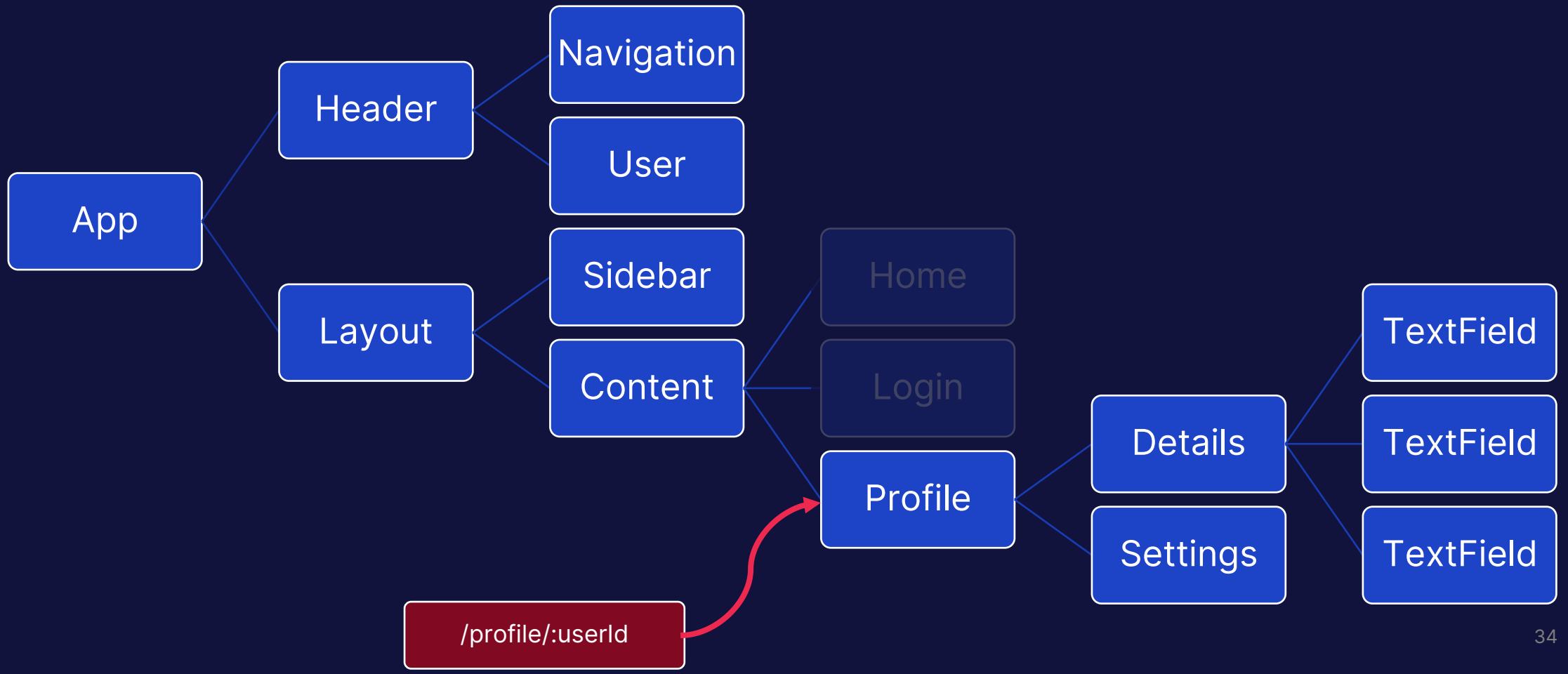
/login



bouvet

Routing with parameter

/profile/0e768eea-1c35-4024-8769-ef2dd62915e2



bouvet

Setting up routing

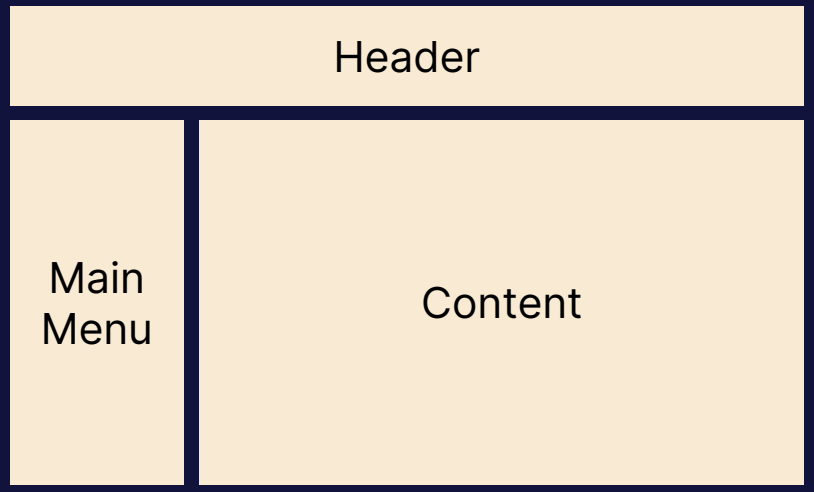
- `pnpm add react-router-dom`
- Define routes with pages
  - / -> Home Page
  - /users -> Users Page
- Provide routes
- Create error pages
- Navigation with links

Layouts

- Control the view surrounding the “route” component.

MainLayout

- Create a classic two-column layout with the header on top, navigation to the left and main content to the right.
- Use the layout for all pages in the application.



<> GroupsPage

- Create a page that displays the static groups list.
- Add a route for it at /groups
- Add a home link to the page.
- Add a link to the groups page on the home page.
- Add a parameter that takes a group id.
- Highlight the group in the list if the parameter is present and the group exists.

<> GroupDetailsPage

- Create a page that displays details about a single group.
- The id of the group to display should come from a URL parameter.
  - E.g.: /groups/abc123
- Add links to the id column of the static groups table that link to the details page for that specific group.

<> MainMenu

- Create a component that displays a list of main menu links for the application.
- Include links to pages and a Home link to the home page.
- Add the menu to the Home Page and each sub page.
- Bonus: Highlight the link that is currently active.

Optimize rendering

- Make sure to only run code that needs to run.
- Mark heavy code so that it only runs if values it depends on have changed (useMemo())
- Mark an entire component as “pure” so that it is only called if any of its props have changed (React.memo())
- Write immutable state changes.

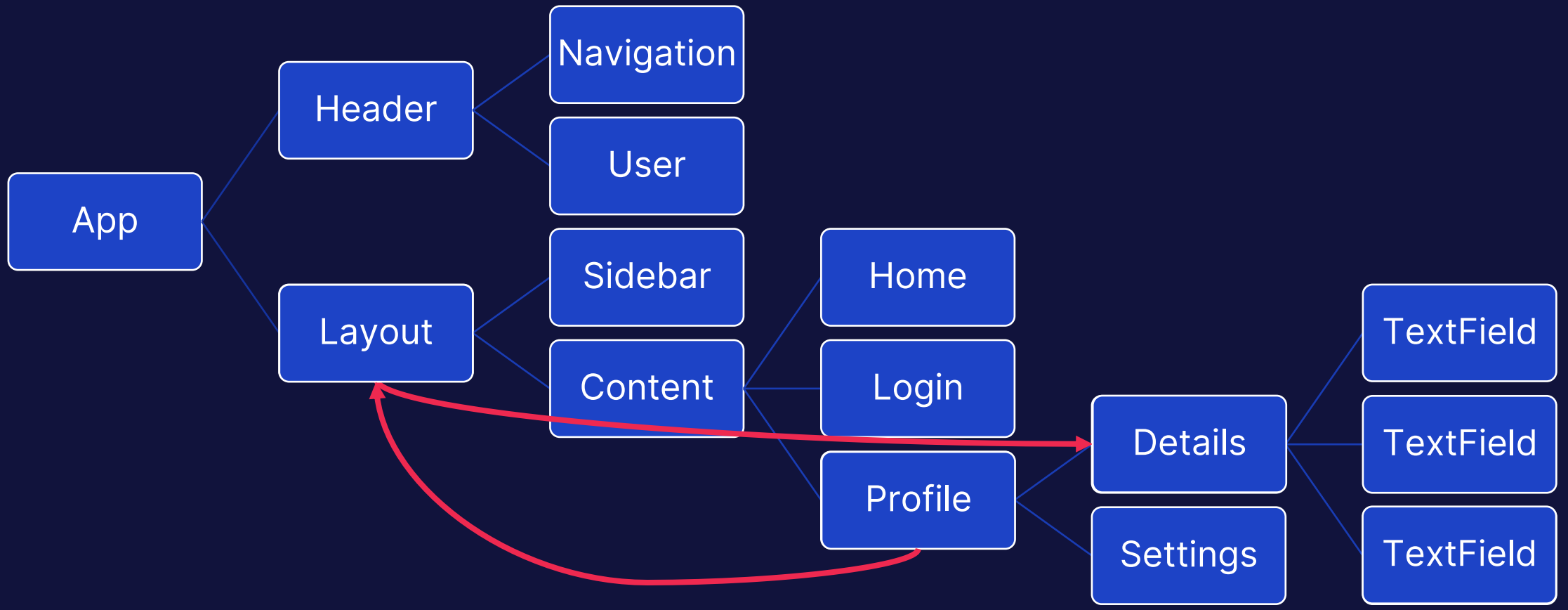
Immutability

- React works on the assumption that objects are immutable.
- An immutable object cannot change it can only be replaced.
- Optimizes for performance.

Code-splitting

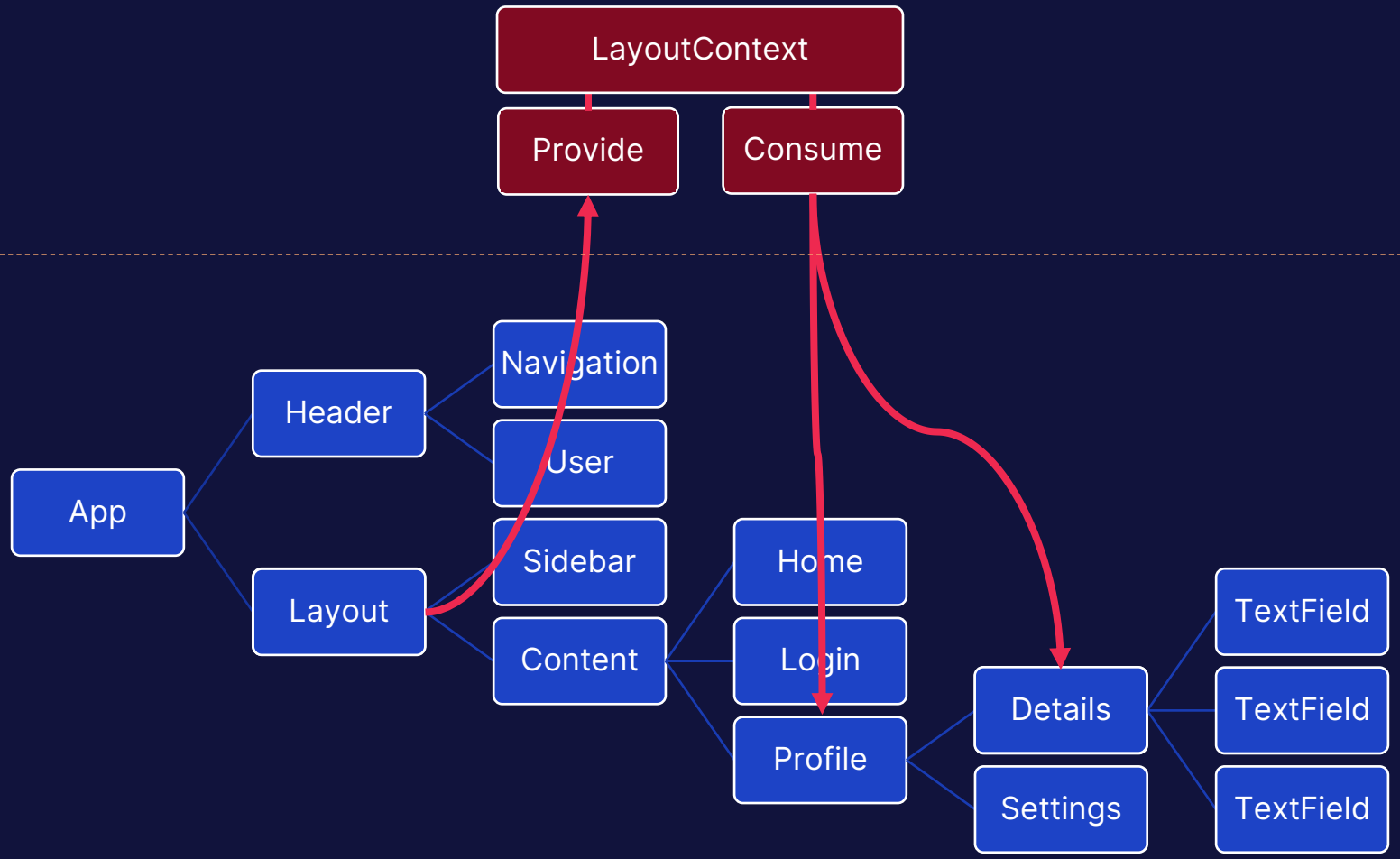
- Reduce the size of the bundle that needs to be downloaded to start your app.
- Divide our bundle into discreet chunks (usually based on routes).

Anatomy of Context



bouvet

Anatomy of Context



bouvet

Services using contexts

- **Provide** (inject) state or functionality to the application tree.
- Manipulate shared state from nonadjacent components.
- Basically, just contexts with extra steps…

HeaderService

- Create a service that allows pages to set the header text in the header component.
- Create a custom hook “useHeading” to easily set the heading text once a page is displayed.

bouvet

<> FieldsService

- Create a service that controls whether any fields below it are enabled or disabled.
- Add a function that allows toggling the disabled state (without providing an explicit true/false flag).
- Add a checkbox to the home page that disables all fields. Make sure it can be deselected as well.
- Bonus: Do not require the service to be provided for individual fields to be used.

Server communication

- Get and set data on a remote server.
- Ensure the date on the client remains up to date.
- Show the user that data is being fetched and handle errors gracefully.
- Use types generate from the system that “owns” them.

Tanstack Query

- Client-side caching and invalidation.
- Query state and error handling.
- Retries and polling.
- Reuse data across components.
- TypeScript support ensures type-safe client code.

Tanstack Query

Two types of requests

Query

- Get data from a server with optional parameters.
- Runs as soon as a useQuery() hook is encountered.
- Read-only.
- Usually corresponds to HTTP verbs: GET.

Mutation

- Change data on the server with optional parameters.
- useMutation() hook runs only when explicitly requested.
- May invalidate locally cached data.
- Usually corresponds to HTTP verbs: POST, PUT, PATCH, DELETE.

ApiStats

- Create a component that displays statistics from the API.

bouvet

<> ApiHealth

- Create a component that shows health information from the API.

Basic login

- Use the login form to log a user in and display the result.
- Disable the form while the user is being logged in.
- Implement a logout button that logs the user out. It should only be visible IF the user is previously logged in.
- Add a loading indicator that can be displayed while the login is in-flight.

Organizing queries and mutations

- Goal: Make it easy to “query” and “mutate” from our pages and components.
- Generalize and isolate queries and mutations.
- Downside: Opinionated and takes some time to get used to.

Data Service

One approach

- A combination of a service and custom hooks for working with queries and mutations.
- Use TkDodos recommeded pattern for holding queries.
- Use custom hooks to encapsulate mutations.
- Use a context to hold and provide queries.

bouvet

Handling session tokens

- Session tokens usually come from an OAuth flow or equivalent.
- They must be passed as a header to the back-end.
- Many ways to do this. See NSwag documentation for more possibilities.
- We will be using the “base-class-method” with a static field.

authDataService

- Create a data service for the Auth and Authenticated User endpoints.
- Provide relevant queries from it.
- Create custom mutation-hooks where applicable.

<> Login using data service hooks

- Create a login page that uses the data service provided useLogin() and useLogout() hooks.

authService

- Create a service to manage authentication information.
- Provide helper functions for logging in, logging out and refreshing the session.
- If the user is authenticated provide all relevant user information:
  - Session
  - User
  - Roles
  - Groups

<> Login with services

- Create a new page where the user can log in.
- Use the data and client services to facilitate login.
- Display all user information using a query.

useEnsureFreshSession

- Create a custom hook that ensures the session is fresh when navigating between pages.
- Do nothing if the user is not authenticated.

bouvet

<> RestrictByRoles

- Create a component that only renders its children if the user is logged in and has one of a set of specified roles.
- If no roles are provided it should only require the user to be authenticated.
- Admins should have access to everything.

Building the application

- What remains now is using what we have learned to build out the application.
- We will most likely not be able to complete all these, but you can find example solutions in the solution/* branches.

<> ActiveSessionsPage

- Create a page where admins can view and cancel all active sessions.

<> usersDataService

- Create a data service for the Users endpoints.
- Provide relevant queries from it.
- Create custom mutation-hooks where applicable.

<> UserDetailsPage

- Create a page that displays details about a specific user from the server.
- It should receive the user id from a URL parameter to allow deep linking.
- It should show a list of all groups the user is a member of.
- It should show a list of all distinct roles the user has.

<> UsersPage

- Create a page that displays all users in the system in a table.
- Clicking the user id or display name should take you to the UserDetailsPage for that user.
- If the current user is a user admin add a delete button for each row.

<> EditUserPage

- Create a page with a form that allows you to edit an existing user.
- Allow admins and user admins to edit any user while regular users can only edit themselves. Guests should not be able to edit themselves.
- Validate the form client-side according to the rules described in the OpenAPI specification.

<> CreateUserPage

- Create a page with a form that allows you to create a user.
- Validate the form client-side according to the rules described in the OpenAPI specification.

<> groupsDataService

- Create a data service for the Groups endpoints.
- Provide relevant queries from it.
- Create custom mutation-hooks where applicable.

<> GroupDetailsPage

- Create a page that displays details about a specific group.
- It should receive the group id from a URL parameter to allow deep linking.

<> GroupsPage

- Create a page that displays all groups in the system in a table.
- Clicking the group id or name should take you to the GroupDetailsPage for that group.
- If the current user is a user admin add a delete button for each row.

<> EditGroupPage

- Create a page with a form that allows you to edit an existing group.
- Allow admins and user admins to edit any group. Others should not be able to edit at all.
- Validate the form client-side according to the rules described in the OpenAPI specification.

<> CreateGroupPage

- Create a page with a form that allows you to create a group.
- Validate the form client-side according to the rules described in the OpenAPI specification.

References

https://react.dev
React documentation

https://nx.dev/
The tool used to generate code and manage the monorepo we are working in

https://vitejs.dev/
Compiler and bundler for our front-end code.

https://reactrouter.com/
A robust router for react

https://griffel.js.org
Atomic-CSS-in-JS library with ahead-of-time compilation

https://emotion.sh
CSS-in-JS styling library

https://tanstack.com/query
Remote-state cache and orchestration library

- https://tkdodo.eu/blog/practical-react-query
  A comprehensive blog post about most of the features of Tanstack Query
- https://query.gg
  The official Tanstack Query course

https://github.com/pmndrs/zustand
Simple global state manager

https://prettier.io
An opinionated code formatter

https://eslint.org
Linting tool for enforcing coding standards

- https://github.com/sindresorhus/eslint-plugin-unicorn
  Extra linting rules

https://playwright.dev/
End-to-end testing using multiple browsers

https://github.com/RicoSuter/NSwag
Tool for generating TypeScript (and other) clients from OpenApi

https://www.mockaroo.com
Tool for generating test data

https://editorconfig.org/
IDE-agnostic formatting and style configurations

https://monorepo.tools/
An overview of what a monorepo is

MDN Web Docs (mozilla.org)
Reference documentation and tutorials for HTML, CSS, JavaScript and more