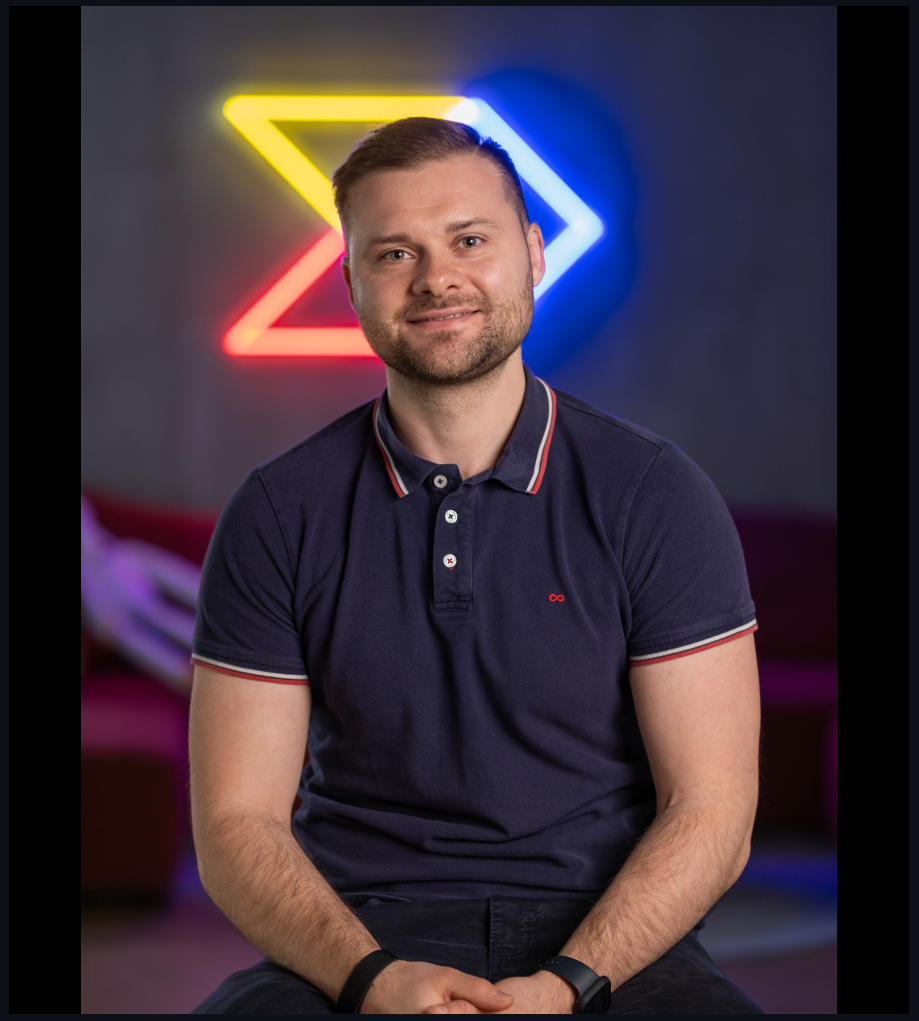
# React Training - Day 1 - Part I

## React Ecosystem and Tools

# About Me

- **Productboard** (since March 2025)
  - Product Staff Engineer
  - Tech Lead Nucleus Guild, member of FE guild
- **React Experience**
  - React Lover (10+ years)
  - Consultant
  - Courses & Workshops
    - React, Next.js, QA
  - Video courses for Skillmea



# Agenda

1. React Router 7
    - Declarative-way
    - Data-way
2. State Management in Pure React
3. Styling Approaches
    - CSS Modules
    - Tailwind CSS
    - Styled Components
    - CSS-in-JS Comparison
4. Forms & Validation

# React Ecosystem & Tools

## Project Structure

- Monorepo with Yarn Workspaces
- Two apps demonstrating different approaches:
  - `data-way` : Traditional React Router
  - `declarative-way` : Modern patterns

## Tech Stack

- React 19
- React Router
- Libs - React-hook-form, Zod

# Project Structure Demo

# React Router 7

## React Router Approaches

## 1. Declarative-way (Old)

```tsx
// DeclarativeRoutes.tsx
export const DeclarativeRoutes = () => (
  <Routes>
    <Route element={<Layout />}>
      <Route path="users">
        <Route index element={<Users />} />
        <Route path=":userId" element={<UserDetails />} />
      </Route>
    </Route>
  </Routes>
);
```

## 2. Data-way (Modern)

```tsx
// Routes.tsx
export const router = createBrowserRouter([
  {
    path: '/',
    element: <Layout />,
    children: [
     {
        index: true,
        element: <Home />,
      },
    ]
  }
]);
```

## 3. Framework

We will not cover

## Client-side Routing

```
// Programmatic navigation
const navigate = useNavigate();
navigate('/users/new');

// Link component
<Link to={`/users/${user.id}`}>View Details</Link>

// Active link detection
<NavLink
  to="/users"
  className={({ isActive }) =>
    isActive ? 'text-blue-600' : 'text-gray-600'
  }
>
  Users
</NavLink>
```

## Route Parameters

```
// Accessing route params
const { userId } = useParams();

// Search params
const [searchParams, setSearchParams] = useSearchParams();
const query = searchParams.get('search');
```

# React Router Demo

# React Router

## API + Availability Table

# State Management

## Local State - useState

```
// useState for simple state
const [isLoading, setIsLoading] = useState(false);
const [error, setError] = useState<Error | null>(null);
```

# Local State - useReducer

## useReducer docs

```
// useReducer for complex state
const [state, dispatch] = useReducer(appReducer, initialState);
```

# Global State Manegement with Context API

```
// Context definition
export const AppStateContext = createContext<AppState | null>(null);
export const AppDispatchContext = createContext<AppDispatch | null>(null);

// Context Provider
export const AppProvider = ({ children }) => {
  const [state, dispatch] = useReducer(appReducer, initialState);
  return (
    <AppStateContext.Provider value={state}>
      <AppDispatchContext.Provider value={dispatch}>
        {children}
      </AppDispatchContext.Provider>
    </AppStateContext.Provider>
  );
};
```

# Scaling Up with Reducer and Context

Docs

# State Management Demo

## Declarative-way App (AppContext)

# Styling Approaches

# CSS Modules

```
// UserCard.module.css
.card {
  padding: 1rem;
  border-radius: 0.5rem;
  background: white;
  box-shadow: 0 1px 3px rgba(0,0,0,0.1);
}

.title {
  font-size: 1.25rem;
  color: #1e40af;
  margin-bottom: 0.5rem;
}

// UserCard.tsx
import styles from './UserCard.module.css';

export const UserCard = ({ user }) => (
  <div className={styles.card}>
    <h2 className={styles.title}>{user.name}</h2>
  </div>
);
```

## Tailwind CSS

```tsx
// UserCard.tsx
export const UserCard = ({ user }) => (
  <div className="p-4 rounded-lg bg-white shadow-sm">
    <h2 className="text-xl text-blue-800 mb-2">{user.name}</h2>
    <p className="text-gray-600">{user.email}</p>
  </div>
);
```

# Styled Components

```tsx
// UserCard.tsx
import styled from 'styled-components';

const Card = styled.div`
  padding: 1rem;
  border-radius: 0.5rem;
  background: white;
  box-shadow: 0 1px 3px rgba(0,0,0,0.1);
`;

const Title = styled.h2`
  font-size: 1.25rem;
  color: #1e40af;
  margin-bottom: 0.5rem;
`;

export const UserCard = ({ user }) => (
  <Card>
    <Title>{user.name}</Title>
  </Card>
);
```

# Styling Demo

## Data-way App (styling)

# CSS-in-JS Comparison

## Pros and Cons

| Approach | Pros | Cons |
| --- | --- | --- |
| CSS Modules | - Scoped styles<br>- No runtime overhead<br>- Familiar CSS syntax | - No dynamic styles<br>- Limited composition |
| Tailwind | - Utility-first<br>- Highly customizable<br>- Great DX | - Large CSS bundle<br>- Learning curve |
| Styled Components | - Dynamic styles<br>- Component-based<br>- Theme support | - Runtime overhead<br>- Bundle size |

# Forms and Validation

# Controlled Components

```tsx
// CreateUser.tsx
export const CreateUser = () => {
  const [formData, setFormData] = useState({
    name: '',
    email: ''
  });

  const handleSubmit = (e: FormEvent) => {
    e.preventDefault();
    // Validation and submission
  };

  return (
    <form onSubmit={handleSubmit}>
      <Input
        value={formData.name}
        onChange={e => setFormData(prev => ({
          ...prev,
          name: e.target.value
        }))}
      />
    </form>
  );
};
```

# Form Validation with useActionState

```tsx
// CreateUser.tsx
export const CreateUser = () => {
  const [state, submitForm, isPending] = useActionState<ActionState, FormData>(
    async (_, formData) => {
      ...
    },
    { success: null, errors: {} }
  );

  return (
    <form action={submitForm}>
      ...

      <button type="submit" disabled={isPending}>
        {isPending ? 'Creating...' : 'Create User'}
      </button>
    </form>
  );
};
```

# Form Validation with Zod

```
// userSchema.ts
const userSchema = z.object({
  name: z.string().min(2, 'Name is too short'),
  email: z.string().email('Invalid email'),
});

// Usage in component
const validateForm = (data: FormData) => {
  try {
    return userSchema.parse(data);
  } catch (error) {
    if (error instanceof z.ZodError) {
      return { errors: error.errors };
    }
    throw error;
  }
};
```

# Form Validation with React Hook Form

```tsx
// CreateUserForm.tsx
import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';

export const CreateUserForm = () => {
  const {
    register,
    handleSubmit,
    formState: { errors, isSubmitting }
  } = useForm({
    resolver: zodResolver(userSchema),
    defaultValues: {
      name: '',
      email: ''
    }
  });

  const onSubmit = async (data: UserFormData) => {
    ...
  };

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <div>
        <input {...register('name')} />
        ...
      </div>

      <button type="submit" disabled={isSubmitting}>
        {isSubmitting ? 'Creating...' : 'Create User'}
      </button>
    </form>
  );
};
```

# Form Validation with React Router action

```tsx
// CreateUser.tsx
export const action = async ({ request }: { request: Request }) => {
  const formData = await request.formData();
  const name = formData.get('name');
  const email = formData.get('email');

  try {
    const user = await createUser({ name, email });
    return redirect(`/users/${user.id}`);
  } catch (error) {
    return json({ errors: { form: 'Failed to create user' } }, { status: 400 });
  }
};
```

# Form and Validation Demo

Declarative-way App - useActionState

Data-way App - Action, Zod, RHF

# Form Handling Best Practices

1. **Validation**
   - Client-side validation for immediate feedback
   - Server-side validation for security
   - Clear error messages
   - Disable form during submission
2. **State Management**
   - Use `useActionState` for form submission
   - Handle loading and error states
   - Provide feedback on success/failure
   - Maintain form state during validation

3. **User Experience**
    - Disable submit button during submission
    - Show loading indicators
    - Clear error messages on new input
    - Provide clear success/error feedback
4. **Accessibility**
    - Proper label associations
    - Error message announcements
    - Keyboard navigation
    - Focus management

# Resources

- React Documentation
- React Router Documentation
- TypeScript Documentation
- Tailwind CSS Documentation
- Zod Documentation

# Thank You!

## Questions?

kristofmartin.eu