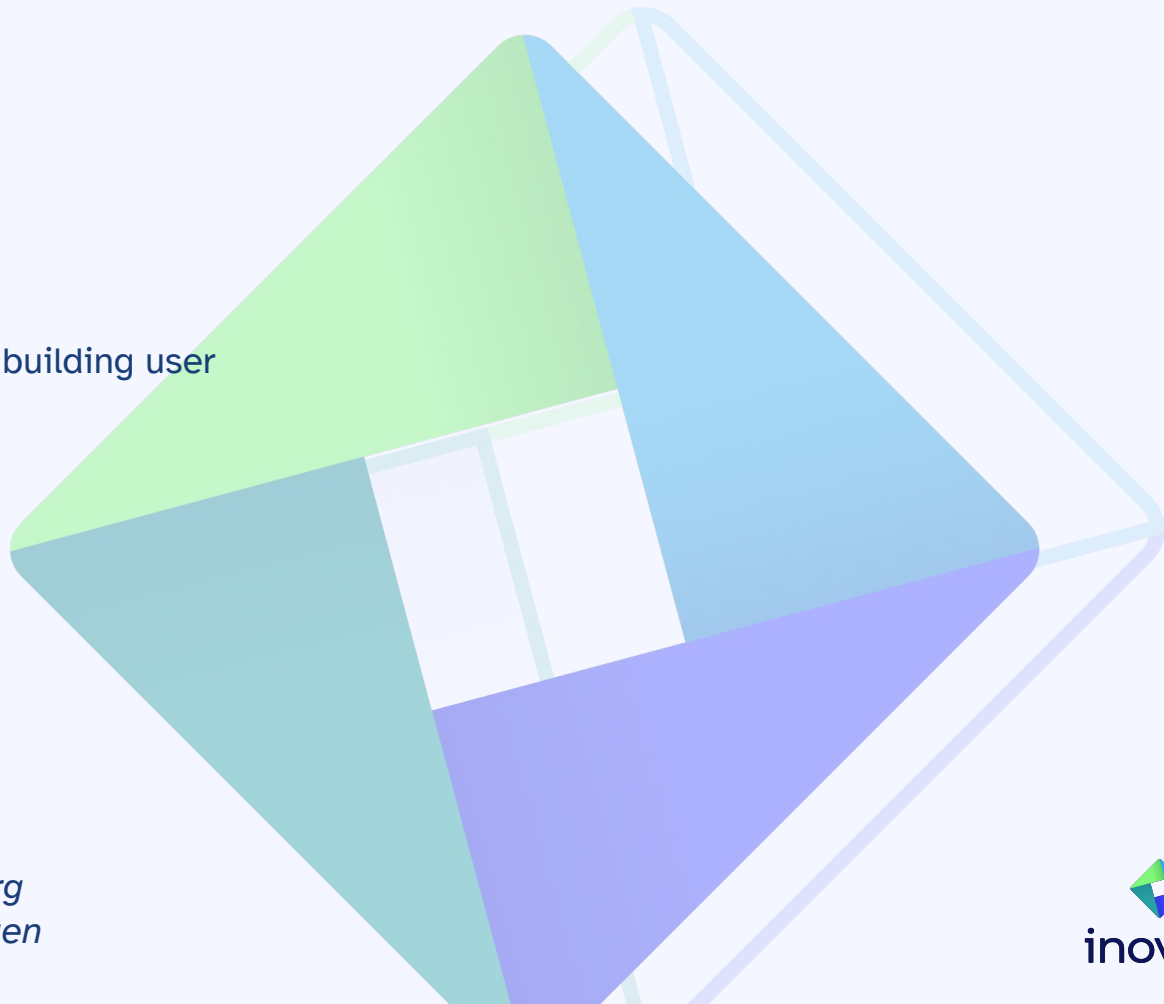


# SolidJS

Simple and performant reactivity for building user interfaces

## Team inovex

*Karlsruhe · Köln · München · Hamburg  
Berlin · Stuttgart · Pforzheim · Erlangen*



# Bernd Kaiser



Software Developer from Erlangen

Focus:

- Web
- Security



[Bernd Kaiser](#)

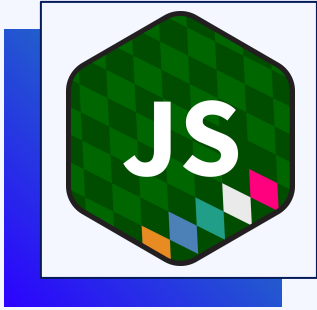


[@meldron](#)



[JSCC23](#) Organizer

# JavaScript Craft Camp 2023



## June 30th & July 1st in Munich

- [BarCamp](#) for JavaScript enthusiasts of all levels
- Free
- Organized by the community, for the community

Register at

<https://iscraftcamp.org/registration>

# Agenda

- SolidJS Reactivity
- JSON Web Token
- Component Props
- Control Flow
- Stores
- Async



# SolidJS Basics

Components, Reactivity & more

## What is SolidJS?

- Performant
- Powerful
- Pragmatic
- Productive



# Performant

Name Duration for...	vanillajs	solid- v1.5.4	vue- v3.2.47	svelte- v3.58.0	angular- v15.0.1	react- v18.2.0
Implementation notes	772					
Implementation link	code	code	code	code	code	code
<a href="#">create rows</a> creating 1,000 rows (5 warmup runs).	37.9 ± 0.4 (1.07)	36.4 ± 0.2 (1.03)	42.0 ± 0.7 (1.18)	48.4 ± 0.1 (1.37)	45.7 ± 0.5 (1.29)	47.5 ± 0.6 (1.34)
<a href="#">replace all rows</a> updating all 1,000 rows (5 warmup runs).	38.4 ± 0.3 (1.00)	40.7 ± 0.3 (1.06)	44.8 ± 0.5 (1.17)	51.4 ± 0.7 (1.34)	48.2 ± 0.6 (1.25)	52.8 ± 0.3 (1.37)
<a href="#">partial update</a> updating every 10th row for 1,000 rows (3 warmup runs). 16 x CPU slowdown.	88.4 ± 2.3 (1.01)	90.4 ± 2.0 (1.03)	103.5 ± 3.4 (1.18)	103.0 ± 3.0 (1.17)	96.7 ± 2.5 (1.10)	128.1 ± 3.0 (1.46)
<a href="#">select row</a> highlighting a selected row. (5 warmup runs). 16 x CPU slowdown.	11.8 ± 1.3 (1.24)	13.1 ± 1.0 (1.37)	22.1 ± 1.3 (2.31)	15.8 ± 1.3 (1.65)	15.6 ± 1.0 (1.64)	39.3 ± 0.6 (4.12)
<a href="#">swap rows</a> swap 2 rows for table with 1,000 rows. (5 warmup runs). 4 x CPU slowdown.	25.0 ± 0.8 (1.01)	28.7 ± 0.5 (1.16)	29.0 ± 0.8 (1.17)	27.6 ± 0.9 (1.11)	166.0 ± 1.0 (6.71)	163.5 ± 0.7 (6.61)
<a href="#">remove row</a> removing one row. (5 warmup runs). 4 x CPU slowdown.	38.8 ± 0.9 (1.01)	39.6 ± 1.1 (1.03)	45.8 ± 1.1 (1.20)	41.1 ± 1.1 (1.07)	42.3 ± 1.2 (1.10)	47.7 ± 1.3 (1.25)
<a href="#">create many rows</a> creating 10,000 rows. (5 warmup runs with 1k rows).	398.2 ± 1.7 (1.00)	420.5 ± 3.5 (1.06)	475.3 ± 1.5 (1.19)	524.0 ± 3.6 (1.32)	474.2 ± 1.9 (1.19)	661.5 ± 2.4 (1.66)
<a href="#">append rows to large table</a> appending 1,000 to a table of 10,000 rows. 2 x CPU slowdown.	82.2 ± 0.6 (1.00)	85.4 ± 0.5 (1.04)	94.7 ± 0.8 (1.15)	109.1 ± 0.4 (1.33)	101.7 ± 0.8 (1.24)	115.0 ± 0.7 (1.40)
<a href="#">clear rows</a> clearing a table with 1,000 rows. 8 x CPU slowdown. (5 warmup runs).	29.2 ± 0.7 (1.02)	34.5 ± 0.6 (1.20)	34.7 ± 1.2 (1.21)	40.5 ± 1.0 (1.41)	61.1 ± 1.7 (2.13)	37.9 ± 1.0 (1.32)
<a href="#">geometric mean</a> of all factors in the table	1.04	1.10	1.27	1.30	1.60	1.87



# Essentials of SolidJS Components

- **JSX Syntax:** good support in all editors & TypeScript
- **Component Composition:** Nested and reusable components
- **Rerender Control:** executed once - only the parts of the UI that are affected by the change will be updated
- **Component Lifecycle:** `onMount` & `onCleanup`

```
function MyButton() {  
  return (<button>I'm a button</button>);  
}  
  
export default function MyApp() {  
  onMount(() =>  
    console.log("app mounted")  
  );  
  return (  
    <div>  
      <h1>Welcome to my app</h1>  
      <MyButton />  
    </div>  
  );  
}
```



# Reactivity: Signals

- `createSignal` is the basic building block
- returns a *getter* and *setter* function
- SolidJS tracks signal *getter* usage in components
- on *setter* usage, all depended components are updated
- updates are fine-grained and do not force a component rerender

```
function Counter() {  
  const [count, setCount] = createSignal(1);  
  const increment = () =>  
    setCount(count() + 1);  
  
  return (  
    <button type="button" onClick={increment}>  
      {count()}  
    </button>  
  );  
}
```

## Derived Signals

- Functions wrapping signals also behave as signals.
- Reactivity Source: Derived signals gain reactivity from accessed signals.

```
function Counter() {  
  const [count, setCount] = createSignal(0);  
  const doubleCount = () => count() * 2;  
  setInterval(() => setCount(count() + 1), 1000);  
  return <div>Count: {doubleCount()}</div>;  
}
```

# createEffect

- Run side effects when dependencies change
- Dependencies: Signals, Memos, Props, Stores
- Effects are meant primarily for side effects that read but don't write to the reactive system

```
function Counter() {  
  const [count, setCount] = createSignal(0);  
  createEffect(() => {  
    console.log("The count is now", count());  
  });  
  return <button onClick={() => setCount(count() + 1)}>Click Me</button>;  
}
```

## createMemo

- Memos are both an observer, like an effect, and a read-only signal
- Run only once for any change
- Cache values in order to reduce duplicated work

```
function Counter() {
  const [count, setCount] = createSignal(1);
  const fib = createMemo(() => {
    console.log('Calculating Fibonacci');
    return fibonacci(count());
  });
  return <button onClick={() => setCount(count() + 1)}>{fib()}{fib()}</button>;
}
```

# JSON Web Token

Short JWT introduction

# JSON Web Tokens (JWT)

- **JWT Definition:** Digitally signed tokens for secure data exchange.
- **Self-contained:** Encodes all relevant information within the token.
- **Usage:** Primarily for authentication and secure information exchange.
- **Scalable Authorization:** Stateless design reduces server load.
- **Data Transmission:** Flexible, supports claims for multiple parties.

# Structure of a JWT

- **Header:** Defines token type (JWT) and signing algorithm used.
- **Payload:** Contains claims or pieces of information about the entity.
- **Signature:** Ensures the sender's identity and data integrity.
- **Encoding:** Each part Base64Url encoded, separated by periods.
- **Format:**  
`encodedHeader.encodedPayload.signature` for a complete JWT.

## Example:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9  
.eyJzdWIiOiI0MiIsIm5hbWUiOiJBcnRodX  
iLCJpYXQiOiE1MTYyMzkwMjJ9.kxQ5yjGcYt  
_H_uDihzbrx9G7sAQP9rdh1_sc0VgtUrM
```

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}  
  
{  
  "sub": "42",  
  "name": "Arthur",  
  "iat": 1516239022  
}
```

## JWT Usage

JWT are typically send with every request in the *Authorization* HTTP header as *Bearer Token*.

Example:

```
curl -X GET \  
      -H "Authorization: Bearer eyJhbG...VgtUrM" \  
      https://example.com/api/resource
```



# JWT Disclaimers

- 🤔 Treat JWT as secrets and never paste them into websites
- 👁️ Use local tools (you trust) to explore JWTs
- 🚫 Do not implement verification yourself
- 🤔 Token revocation is a hard problem
- 🚚 Big JWT increase the size of every request

## Exercise 1: Signals & Effects

1. `git clone https://github.com/meldron/solid-jwt-workshop.git`
2. Follow setup and run instructions
3. Open `src/App.tsx` and add reactivity to **Token** text area so that new Tokens update Header, Payload & co
4. Bonus: verify Token



# Component Props & Control FLOW

# Component Properties

- Props: readonly reactive properties passed to components
- Consistent Form: unchanged by signals, expressions, or static values
- Access Method: utilized via `props.propName`
- Avoid Destructuring **!** : Prevents reactivity loss outside tracking scope
- Utility Functions: merge/split reactive objects, preserving reactivity

# Component Properties Example

```
interface GreetingProps {  
  name: string;  
  age?: number;  
}  
  
const Greeting: Component<GreetingProps> = (props) => {  
  const merged = mergeProps({ age: 42 }, props);  
  return (  
    <div>  
      <p>Hello, my name is {merged.name}.</p>  
      <p>I am {merged.age} years old.</p>  
    </div>  
  );  
};
```

```
<Greeting name="Bernd"/>
```

renders as

Hello, my name is Bernd.  
I am 42 years old.

# Control Flow

- Solid provides several control flow components:  
**Show**, **For**, **ErrorBoundary**, **Switch / Match**, **Index**,  
**Suspense**, **Dynamic**, **Portal**
- Should be preferred over JS control flow (e.g., **Array.map**)
  - **Fine-grained Reactivity**: Only affected components rerender.
  - **Memory Efficiency**: Automatic cleanup of signal listeners.
  - **Syntax Consistency**: Embed control logic directly in JSX.
  - **Keyed Updates**: Better tracking and handling of lists

## Show Component

- Conditionally renders children, if the when condition is true
- When used with a callback, the callback is only executed, if the condition is null asserted
- Optionally can be keyed to a specific data model => the function is re-executed whenever the model is replaced

```
<Show when={merged.age > 17}
      fallback={<div>Loading...</div>}>
  <div>My Mature Content</div>
</Show>
```

```
<Show when={merged.user}
      fallback={<div>Loading...</div>}>
  {{user) => <div>{user.name}</div>}
</Show>
```

```
<Show when={merged.user}
      keyed>
  {{user) => <div>{user.name}</div>}
</Show>
```

# For Component

- Iterates over lists for rendering
- Accepts an `each` and `fallback` prop
- The callback takes the current item as the first argument
- The optional second argument is an *index* signal
- On a list change, updates or moves items in the DOM

```
<For each={merged.users}
  fallback={<div>Loading...</div>}>
  {(item, index) => (
    <div>
      #{index()} {item.name} - {item.age}
    </div>
  )}
</For>
```



# ErrorBoundary Component

- Catches uncaught errors and renders fallback content
- Also supports callback form which passes in error and a reset function.

```
<ErrorBoundary
  fallback={<div>My Bad</div>}>
  <Greeting name="Test"/>
</ErrorBoundary>
```

```
<ErrorBoundary
  fallback={
    (err, reset) =>
      <div onClick={reset}>
        Error: {err.toString()}
      </div>
  }>
  <Greeting name="Test"/>
</ErrorBoundary>
```

## Exercise 2: Refactor into Components

- Refactor the app into multiple components with props
  - TokenInput
  - Header
  - Payload
  - ...
- Use Control Flow Components to enhance the user experience
  - For loop over header properties / values
  - Show an error message if the JWT is invalid (not set)
  - ...
- Use `.tsx` as file extension for components
- Bonus: verify token only for HS256 JWTs

# Stores

How to make your life easier

# Stores

- Stores are proxy objects supporting nested reactivity
- Signals are created as needed under tracking scopes
- `createStore` returns a readonly store proxy and a setter function
- Merges new properties with state and supports nested updates
- **Path Syntax:** Allows powerful iteration, range capabilities and granular reactivity

```
const [todos, setTodos] =  
  createStore([]);
```

```
const addTodo = (text) => {  
  setTodos(  
    [...todos,  
    { id: ++todoId,  
      text,  
      completed: false  
    }])  
};
```

```
const toggleTodo = (id) => {  
  setTodos(  
    todo => todo.id === id,  
    "completed",  
    completed => !completed  
  );  
};
```

# Store Mutations

- Solid strongly recommends the use of shallow immutable patterns for updating state
- **produce** is and an Immer inspired store modifier
- **produce** mutates writable proxy version of the Store

```
const [todos, setTodos] =
  createStore([]);
const addTodo = (text) => {
  setTodos(
    produce((todos) => {
      todos.push(
        { id: ++todoId,
          text,
          completed: false
        }
      );
    }));
};
const toggleTodo = (id) => {
  setTodos(
    todo => todo.id === id,
    produce((todo) =>
      (todo.completed =
        !todo.completed)),
  );
};
```

## Exercise 3: add Stores

- Add store(s) in separate file(s) (use `.ts` file extension)
- Use store in your components to pass state around

# Async

Promises & other lazy Stuff

# createResource

- **createResource** provides simple and efficient data-fetching.
- Streamlines handling of async operations in UI.
- Automatically handles component re-render on data change.
- Built-in suspense and error boundary support.

```
const [data, { refetch }] =
  createResource(requestToken);

return (
  <>
    <Show when={data()}>{data()}</Show>
    <Show when={data.loading}>⌚</Show>
    <Show when={data.error}>
      ⚠ {data.error.message}
    </Show>
  </>
);
```



## More Async Helper

- **lazy**: Allows dynamic import of components to supports code splitting
- **Suspense**: coordinating multiple async events, eliminating partial loading states, and offering fallback during resolvment
- **SuspenseList**: allows grouping and ordering the reveal of loaded **Suspense** components
- **useTransition**: maintaining current view until all asynchronous events are complete

## Exercise 4: Async Calls

- `cd` into `token-server/` and run `npm run start`
- Explore `src/facts.ts`
- Use a `createResource` to load a JWT with `requestToken`
- Use `getFact` with the loaded JWT to receive a animal fact
- Display animal fact

# Thank you!



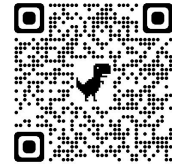
Bernd Kaiser

Software Developer

[bernd.kaiser@inovex.de](mailto:bernd.kaiser@inovex.de)

inovex is an IT project center driven by innovation and quality, focusing its services on 'Digital Transformation'.

- founded in 1999
- 500+ employees
- 8 offices across Germany



[www.inovex.de](http://www.inovex.de)