ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Professional NgRx

4 - NgRx & Friends

Professional NgRx

5 - NgRx & Friends

Agenda

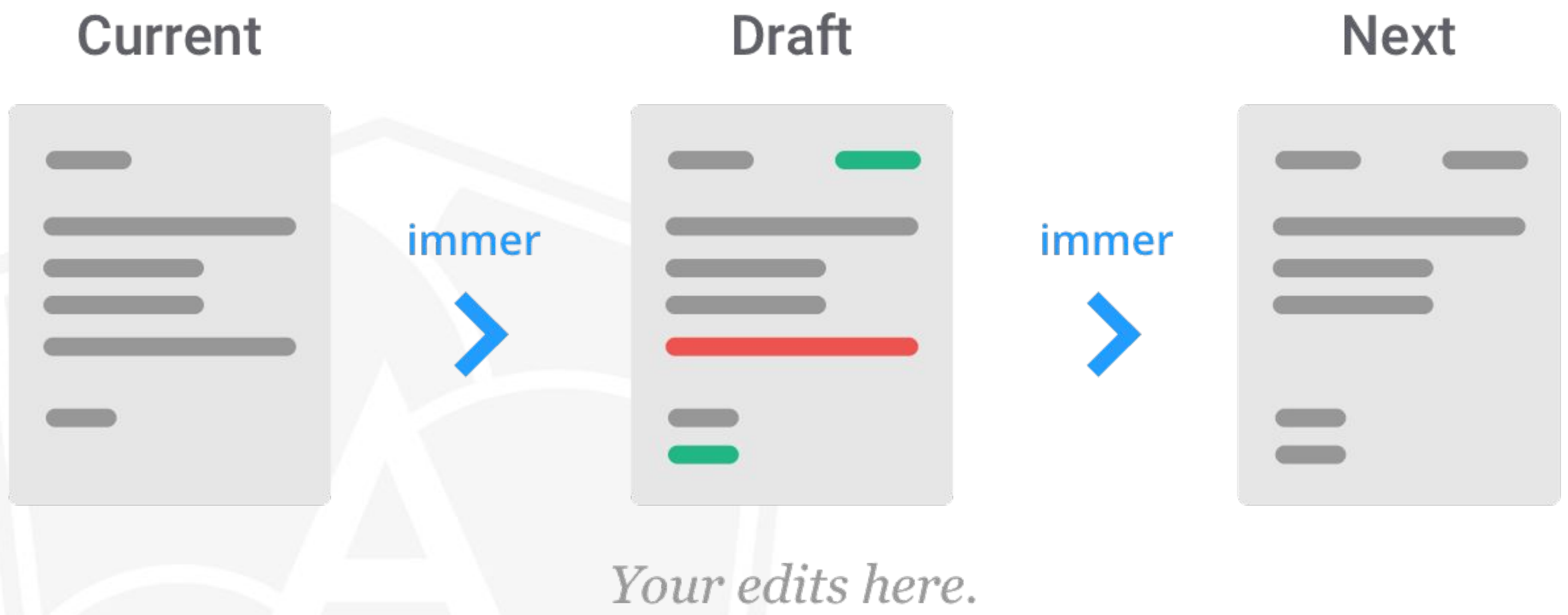
- ngrx-immer: guaranteed immutability
- ngrx-store-localstorage: local storage for persistence
- ngrx-wieder: undo, redo
-

ngrx-immer

Immer

- Transforms mutable into immutable function
- https://github.com/immerjs/immer
- ngrx-immer as bridging library
- more readable code
- guaranteed immutability
    - inside the reducer only

Immer

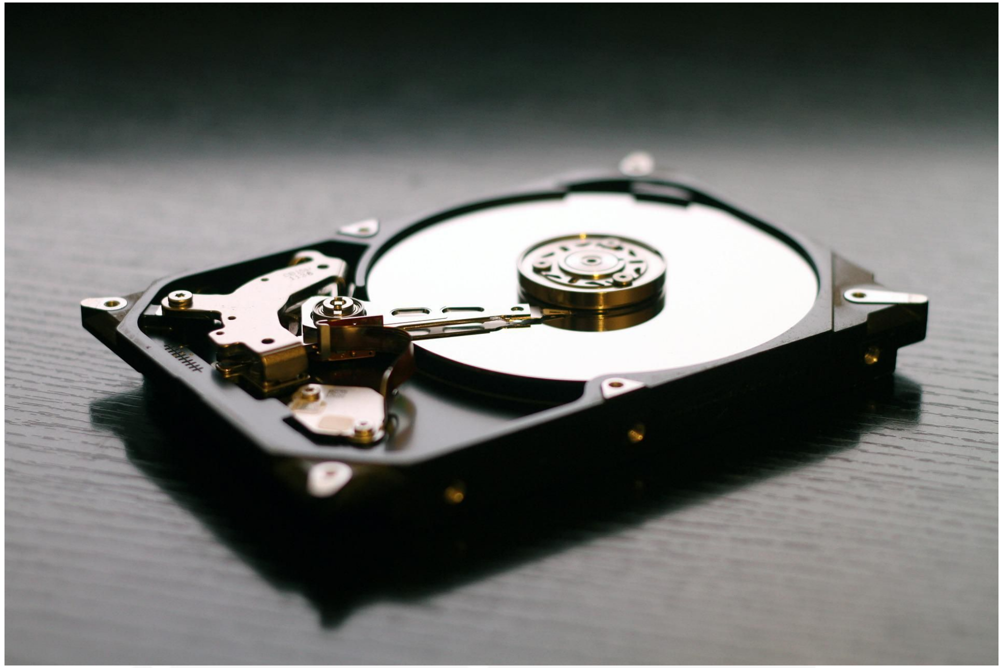


ngrx-immer (formerly ngrx-etc)

```
createReducer(
  initialState,
  on(load, (state) => ({
    ...state,
    loadStatus: "loading",
  })),
  on(loaded, (state, { holidays }) => ({
    ...state,
    loadStatus: "loaded",
    holidays,
  }))
);
```

```
createReducer(
  initialState,
  immerOn(load, (state) => {
    state.loadStatus = "loading";
  }),
  immerOn(loaded, (state, { holidays }) => {
    state.loadStatus = "loaded";
    state.holidays = holidays;
  })
);
```

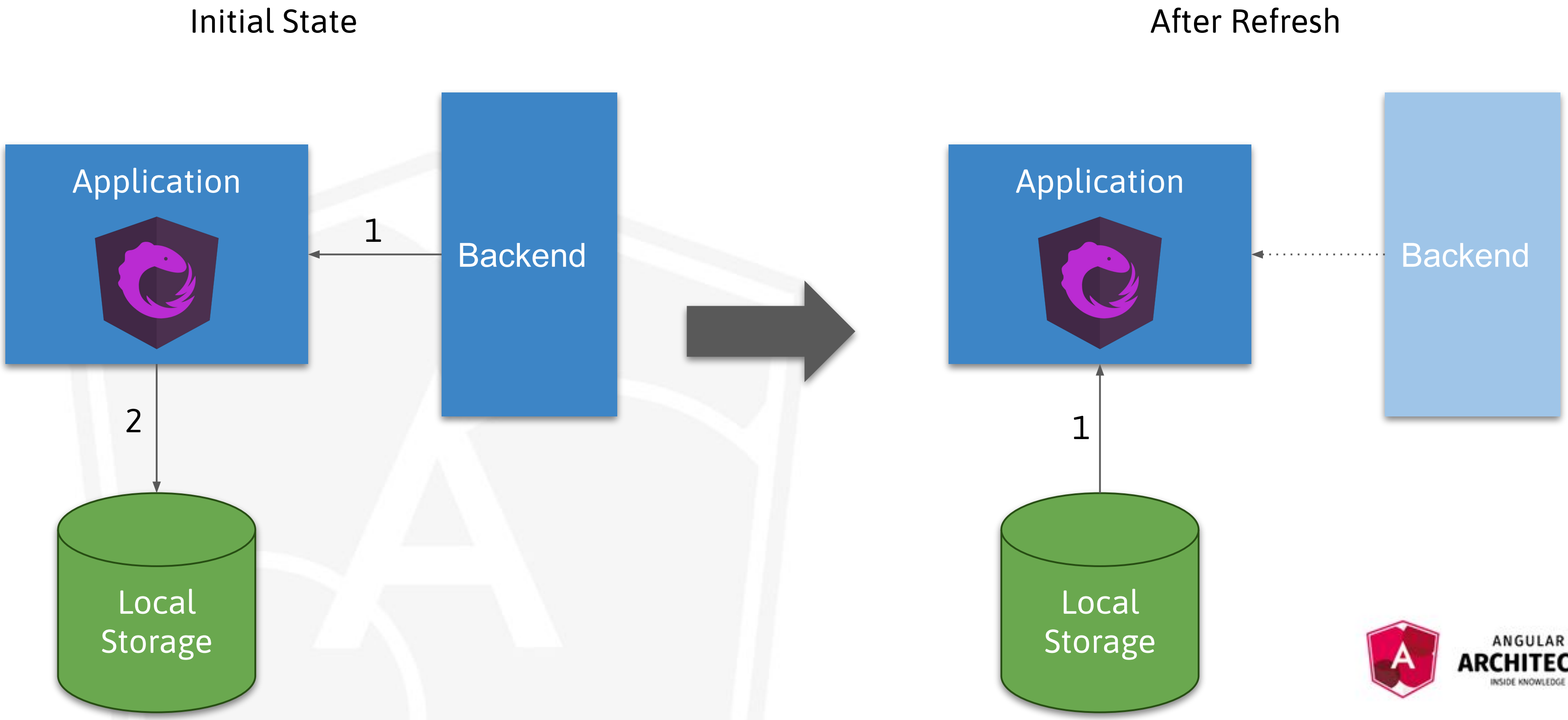
ngrx-store-localstorage



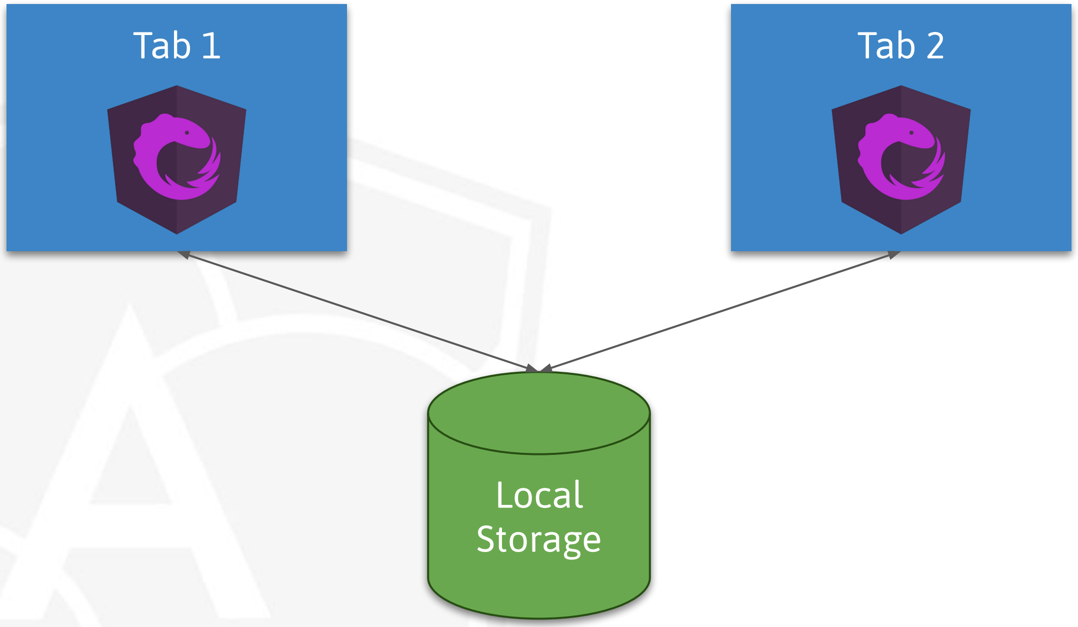
Local Storage

- Caching between browser sessions
- Multi-tab synchronisation
- (limited) offline capabilities
- Extremely powerful but high-risk of overdoing

Caching


Initial State
After Refresh
Application
Backend
1
2
Local Storage
Application
Backend
1
Local Storage

Tab synchronization



Code: configuration

```
const localStorageSyncer = localStorageSync({ keys: ["customers"] });
```

Code: meta reducer

```
const localStorageSyncer = localStorageSync({ keys: ["customers"] });

function localStorageReducer<S>(reducer: ActionReducer<S>): ActionReducer<S> {
  return localStorageSyncer(reducer);
}
```

Code: integrate into StoreModule

```
const localStorageSyncer = localStorageSync({ keys: ["customers"] });

function localStorageReducer<S>(reducer: ActionReducer<S>): ActionReducer<S> {
  return localStorageSyncer(reducer);
}

@NgModule({
  imports: [
    BrowserModule,
    StoreModule.forRoot(reducers, { metaReducers: [localStorageReducer] }),
  ],
})
export class AppModule {}
```

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Local storage: Further information

- Sync state across multiple tabs · Issue #40 · btroncone/ngrx-store-localstorage · GitHub
- Keeping browser tabs in sync using localStorage, NgRx, and RxJS - Tim Deschryver
- GitHub - tomastrajan/angular-ngrx-material-starter

ngrx-wieder

General considerations

- Make
    - Undo Und Redo Mit Redux, @Ngrx/Store Und Angular 2 - ANGULARarchitects
    - ~~Steal~~Copy from outdated https://github.com/Hityy/ngrx-undo-redo
- Buy (it's OS 😉)
    - ngrx-wieder
- Use Cases
    - Difficult with side effects
    - Focus on UI-managed state

ngrx-wieder

- Uses immer internally
- Limiting amount of undoable actions
- Extends a featureState with a history property
- Dedicated selectors
- Support for special use cases:
  - Consecutive: Grouping multiple actions of same type
  - Segmentation: Awareness for selected entities
- Significantly increases state's size

Code: define state

```
export interface HolidaysState extends UndoRedoState {
  holidays: Holiday[];
  loadStatus: LoadStatus;
  favouriteIds: number[];
}
```

Code: initialize state

```
export interface HolidaysState extends UndoRedoState {
  holidays: Holiday[];
  loadStatus: LoadStatus;
  favouriteIds: number[];
}

const initialState: HolidaysState = {
  holidays: [],
  favouriteIds: [],
  loadStatus: "not loaded",
  ...initialUndoRedoState,
};
```

Code: create reducer factory

```
export interface HolidaysState extends UndoRedoState {
  // ...
}

const initialState: HolidaysState = {
  // ...
};

const { createUndoRedoReducer } = undoRedo();
```

Code: create reducer

```
export interface HolidaysState extends UndoRedoState {
  // ...
}

const initialState: HolidaysState = {
  // ...
};

const { createUndoRedoReducer } = undoRedo();

export const holidaysFeature = createFeature({
  name: "holidays",
  reducer: createUndoRedoReducer<HolidaysState>(initialState,
    // ...,
  )
});
```

Code: selectors for history

```
const { selectCanRedo, selectCanUndo } = createHistorySelectors(

  customersFeature.selectHolidaysState

)
```

Usage

```
// Version 1: Native way

this.store.dispatch({ type: "UNDO" });

this.store.dispatch({ type: "REDO" });




// Version 2: Requires additional setup and new actions

this.store.dispatch(holidaysActions.undo());

this.store.dispatch(holidaysActions.redo());
```

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

Optimistic Updates

Optimistic Updates

- Responsive UI
- Built-in feature of nx
- Fallback operation required
- Alternative to customized error-handling operators
  - e.g. safeConcatMap
  - less control

ANGULAR ARCHITECTS INSIDE KNOWLEDGE

Optimistic Updates

- Responsive UI
- Built-in feature of nx
- Fallback operation required
- Alternative to customized error-handling operators
    - e.g. safeConcatMap
    - less control

Code

```
addFavourite$ = createEffect(() =>
  this.actions$.pipe(
    ofType(actions.addFavourite),
    optimisticUpdate({
    })
  )
);
```

Code

default operation →

```
addFavourite$ = createEffect(() =>
  this.actions$.pipe(
    ofType(actions.addFavourite),
    optimisticUpdate({
      run: ({ id }) =>
        this.httpClient
          .post<void>(`${this.#baseUrl}/favourite/${id}`, {})
          .pipe(map(() => actions.favouriteAdded({ id })))
    })
  )
);
```

Code

default operation →

fallback operation →

```
addFavourite$ = createEffect(() =>
  this.actions$.pipe(
    ofType(actions.addFavourite),
    optimisticUpdate({
      run: ({ id }) =>
        this.httpClient
          .post<void>(`${this.#baseUrl}/favourite/${id}`, {})
          .pipe(map(() => actions.favouriteAdded({ id }))),
      undoAction: ({ id }) => actions.addFavouriteUndo({ id }),
    })
  )
);
```

Code

default operation

fallback operation

uses `combineLatest` internally

```
addFavourite$ = createEffect(() =>
  this.actions$.pipe(
    ofType(actions.addFavourite),
    optimisticUpdate({
      run: ({ id }) =>
        this.httpClient
          .post<void>(`${this.#baseUrl}/favourite/${id}`, {})
          .pipe(map(() => actions.favouriteAdded({ id }))),
      undoAction: ({ id }) => actions.addFavouriteUndo({ id }),
    })
  )
);
```

Different actions in reducer

```
createReducer<HolidaysState>(
  // ...
  immerOn(addFavourite, removeFavouriteUndo, (state, { id }) => {
    if (state.favouriteIds.includes(id)) {
      return;
    }

    state.favouriteIds.push(id);
  }),
  immerOn(removeFavourite, addFavouriteUndo, (state, { id }) => {
    state.favouriteIds = state.favouriteIds.filter(
      (favouriteId) => favouriteId !== id
    );
  })
);
```

Forms

GraphQL

- Fetch Holidays along Dates and per Date the Guide information. Display all in one holiday card