Angular & NgRx

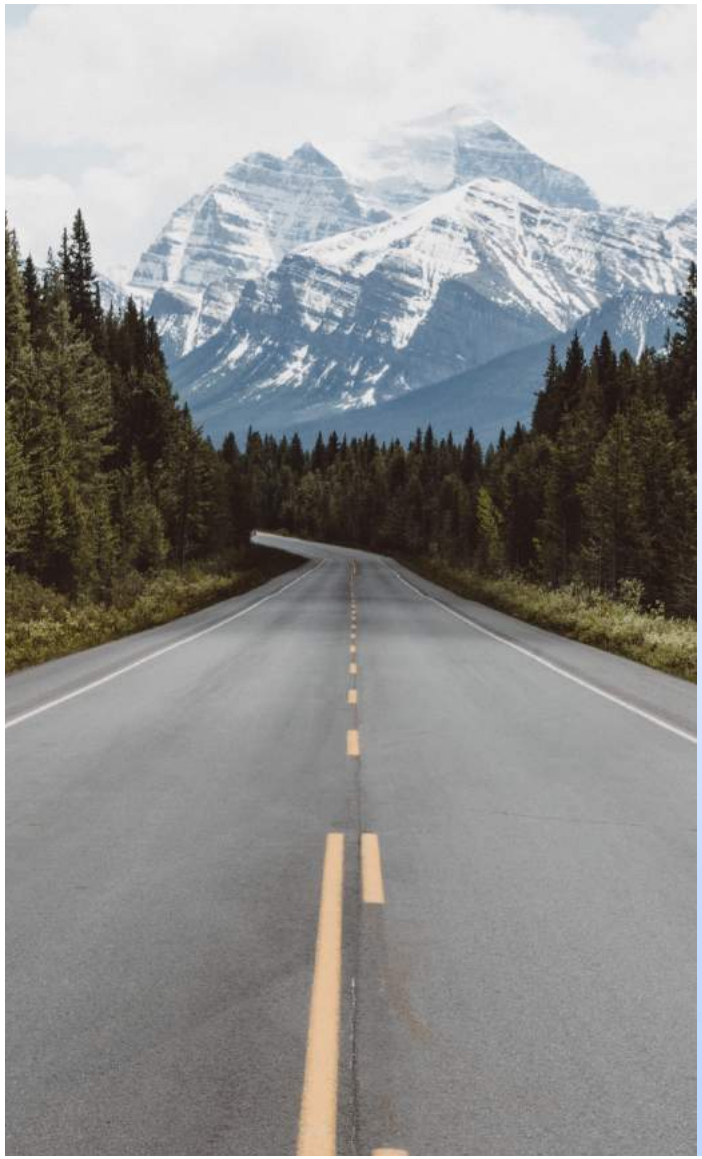
- Slides here:

- Starter project here:
  - http://stackblitz.com/github/apaytonmn/bananaapp
- Checkpoint repositories
- Chrome extension: Redux DevTools
  - Install if you want to participate with debug
- Questions?
  - Please hold questions so we can stay on track

MAYO CLINIC

Angular & NgRx

Aspen Payton
Research Applications Unit
Mayo Clinic



Journey to NgRx

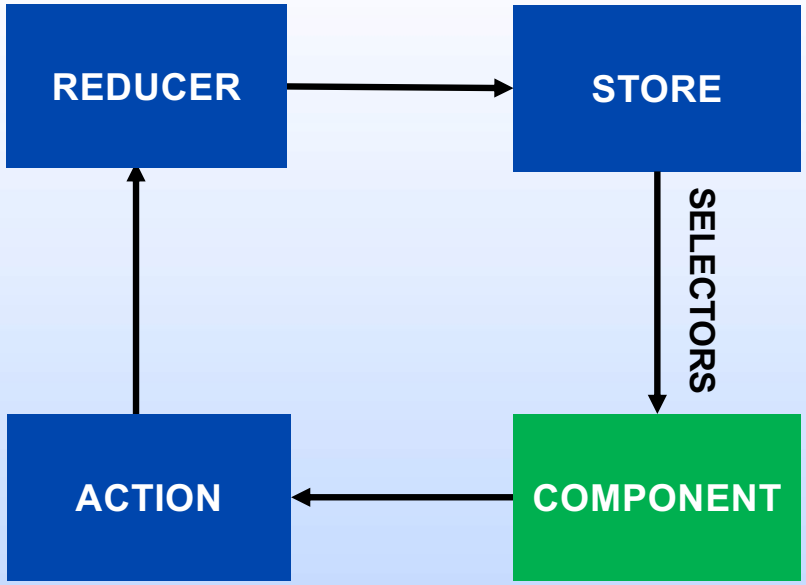
- What is NgRx?
- I had never heard of NgRx until Feb 2018
- Initial misunderstanding of intent
- The AH-HA moment!
- Education challenges
  - Options limited & often too basic
  - Real-world code examples often unrelatable

Why NgRx?

- Avoid spaghetti code!
- Consistent behavior
- All state stored in single object
- Great perks for testing
- Framework agnostic

- Maybe not for everyone

NgRx Basic Pattern

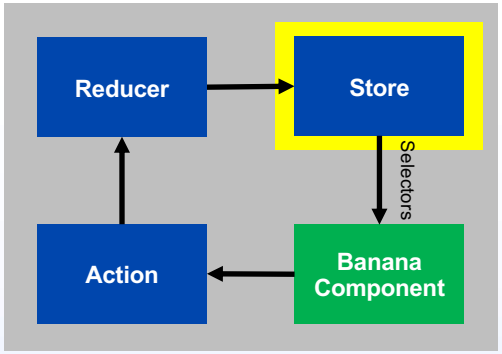


STARTING POINT

- Banana App!
  - https://bit.ly/2GV8XY9
  - http://stackblitz.com/github/apaytonmn/bananaapp
  - Fork the repository
  - Skeleton code structure
  - HTML already written
    - We will not touch the HTML in this workshop
  - Chrome extension: Redux DevTools

Banana State

- In banana directory
  - Create new state directory
  - In state directory, create new state file

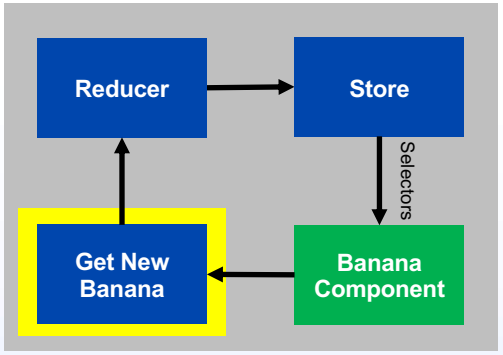


banana.state.ts

```
export interface State {
    isPeeled: boolean;
    bitesRemaining: number;
    color: string;
}

export const initialState: State = {} as State;
```

Action: Get New Banana

- In banana/state directory, create new actions file

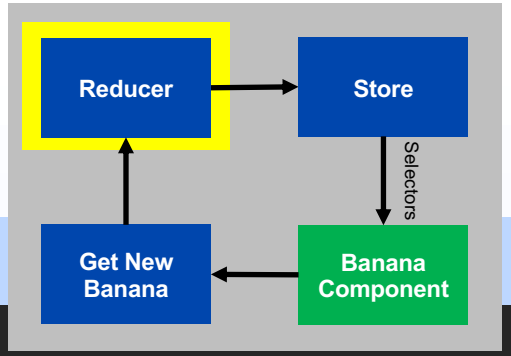


banana.actions.ts

```
import { Action } from '@ngrx/store';

export const GET_NEW_BANANA = 'Get New Banana';

export class GetNewBanana implements Action {
  readonly type: string = GET_NEW_BANANA;

  constructor(public payload: any) {
    console.log('ACTION ' + GET_NEW_BANANA);
  }
}

export type BananaAction = GetNewBanana;
```

Reducer: Get New Banana

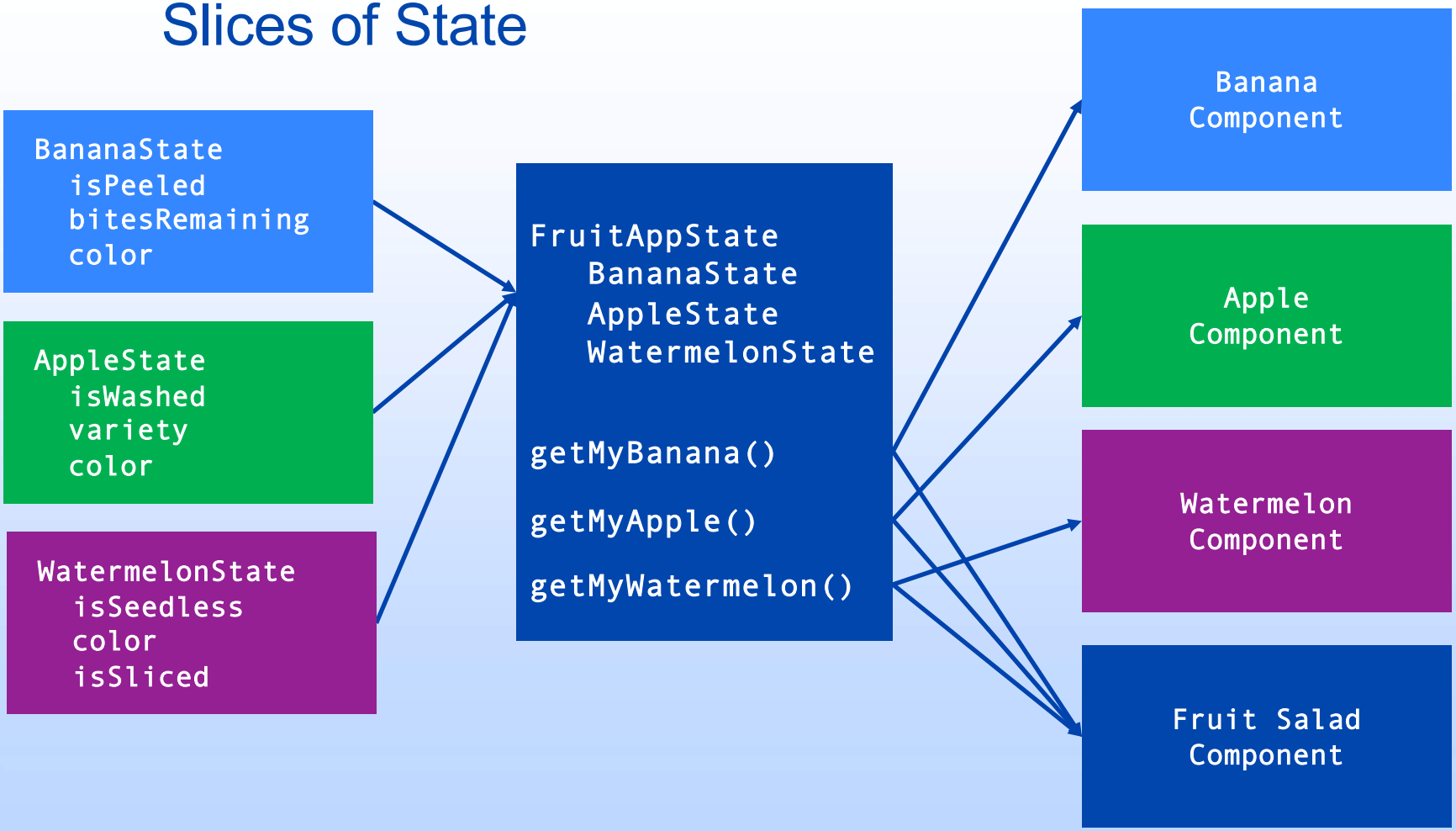
- In banana/state directory, create new reducer file



banana.reducer.ts

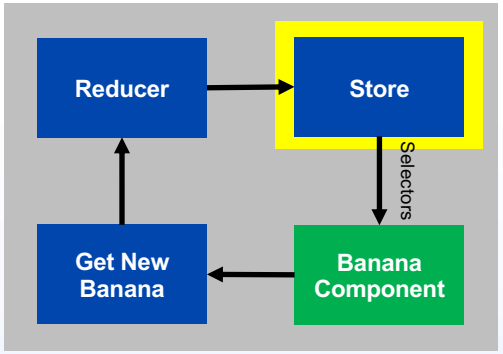
```
import { GET_NEW_BANANA } from './banana.actions';
import * as programActions from './banana.actions';

export function reducer(state: any, action: programActions.BananaAction): any {
  switch (action.type) {
    case GET_NEW_BANANA: {
      console.log('REDUCER ' + GET_NEW_BANANA);
      return {
        isPeeled: false,
        bitesRemaining: 9,
        color: 'yellow'
      };
    }
    default: {
      return {
        ...state
      };
    }
  }
}
```

Slices of State



Wire It Up!

- In banana/state directory, create new file for exports

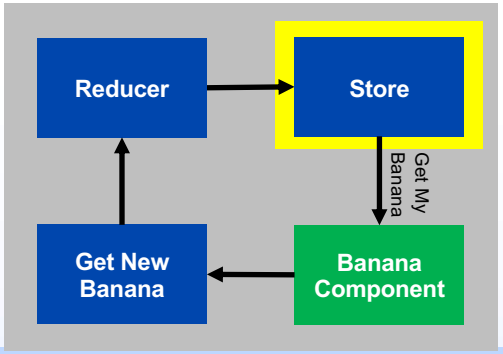


index.ts

```
export { reducer } from './banana.reducer';
export * from './banana.actions';
export { State, initialState } from './banana.state';
```

Wire It Up!

- At the app level create new file to define application level state

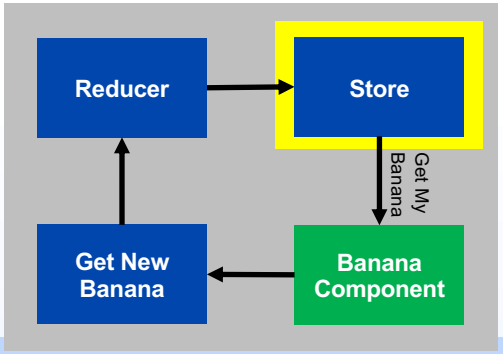


app.state.ts

```
import { ActionReducerMap } from '@ngrx/store';
import * as bananaStore from './banana/state';

export interface AppState {
  banana: bananaStore.State;
}

export const initialState: AppState = {
  banana: bananaStore.initialState
}

export const reducers: ActionReducerMap<AppState> = {
  banana: bananaStore.reducer
}

export const getMyBanana = (s: AppState) => s.banana;
```

Wire It Up!

- Update app module to bring in NgRx support



app.module.ts

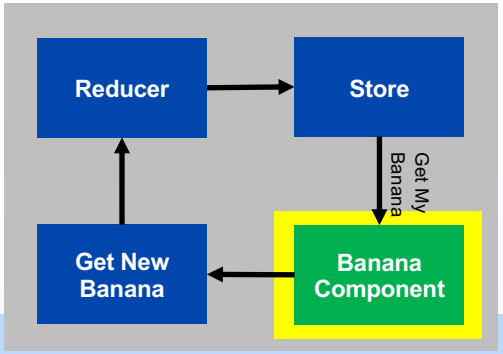
```
import { StoreModule } from '@ngrx/store';
import { StoreDevtoolsModule } from '@ngrx/store-devtools';
import { initialState, reducers } from './app.state';
```

```
StoreModule.forRoot(reducers, {initialState}),
StoreDevtoolsModule.instrument({
  maxAge: 25
})
```

For debug!
We will follow up on this later

Let's Get a Banana!

- Bring all the work we just did into the component

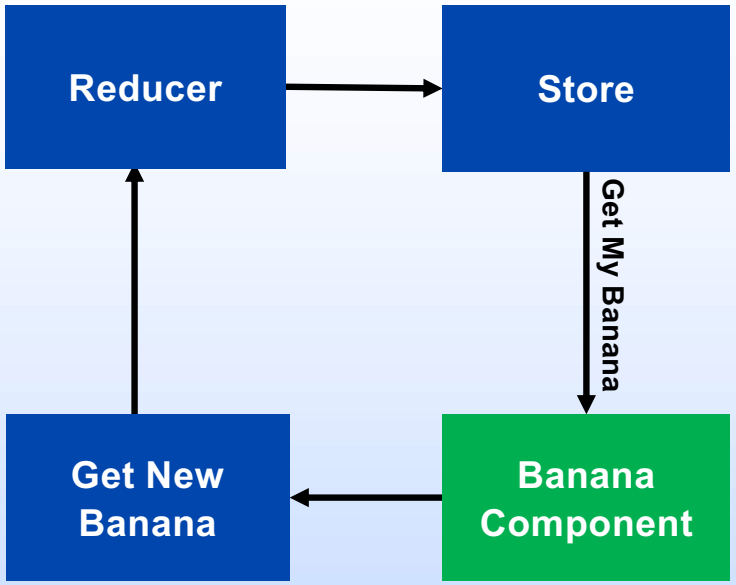


banana.component.ts

```
import { Store, select } from '@ngrx/store';
import { Observable } from 'rxjs';
import { AppState, getMyBanana } from '../app.state';
import { GetNewBanana } from './state';
```

```
banana$: Observable<any>;

constructor(private store: Store<AppState>) {}

ngOnInit() {
  this.newBanana();
  this.banana$ = this.store.pipe(select(getMyBanana));
}

newBanana() {
  this.store.dispatch(new GetNewBanana(null));
}
```

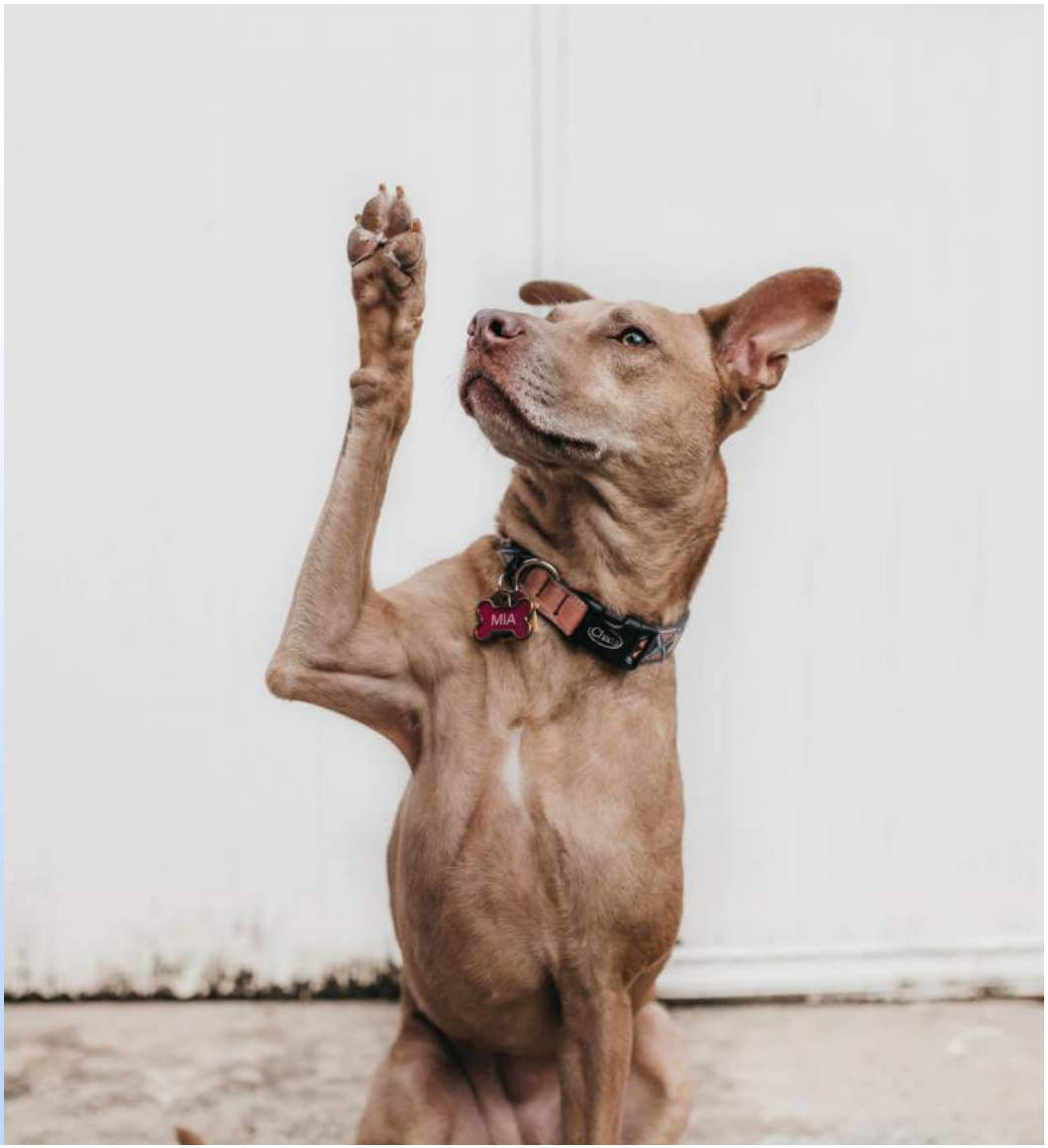
CHECKPOINT #1



- https://bit.ly/2GV8XY9
- http://stackblitz.com/github/apaytonmn/bananaapp-checkpoint-one

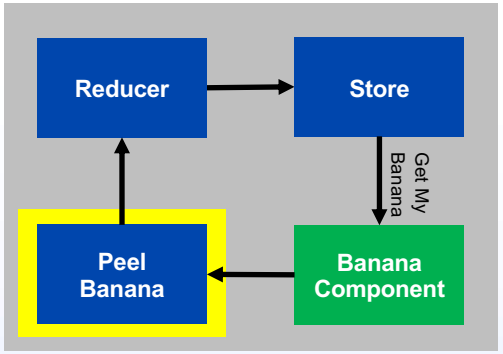
More actions!

- That was a lot of work.
  Do I have to do all that work EVERY time I want to add a new action??
- NO!
  Now that we have the infrastructure in place, adding a new action is easy!



Action: Peel Banana

- Add the Peel Banana action to your actions file



banana.actions.ts

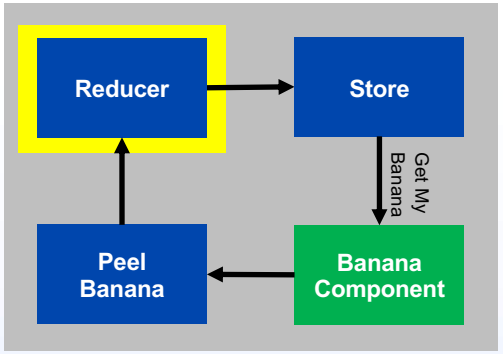
```
export const PEEL_BANANA = 'Peel Banana';
```

```
export class PeelBanana implements Action {
  readonly type: string = PEEL_BANANA;

  constructor(public payload: any) {
    console.log('ACTION ' + PEEL_BANANA);
  }
}
```

```
export type BananaAction = GetNewBanana | PeelBanana;
```

Reducer: Peel Banana

- Handle the Peel Banana action in your reducer



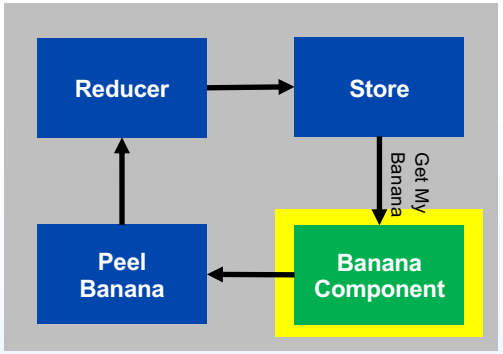
banana.reducer.ts

```
import { GET_NEW_BANANA, PEEL_BANANA } from './banana.actions';
```

```
case PEEL_BANANA: {
  console.log('REDUCER ' + PEEL_BANANA);
  return {
    ...state,
    isPeeled: true
  };
}
```

Component: Peel Banana

- Dispatch the Peel Banana action from your component

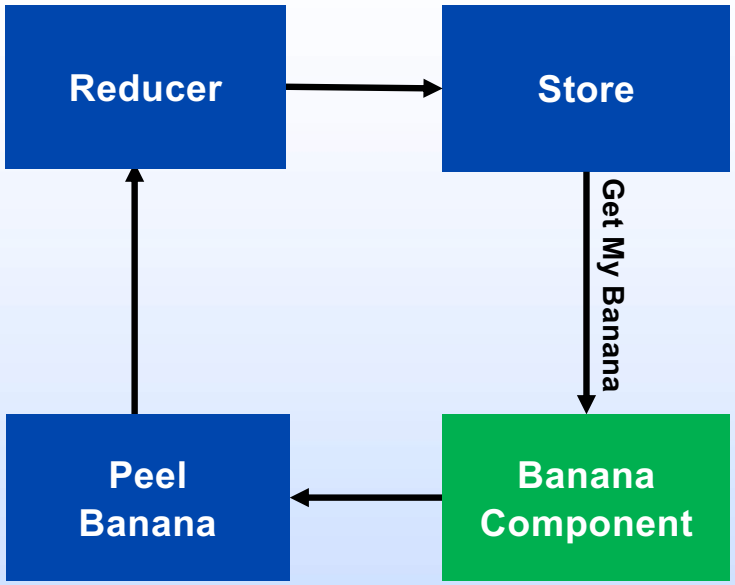


banana.component.ts

```
import { GetNewBanana, PeelBanana } from './state';
```

```
peelBanana() {
  this.store.dispatch(new PeelBanana(null));
}
```

CHECKPOINT #2



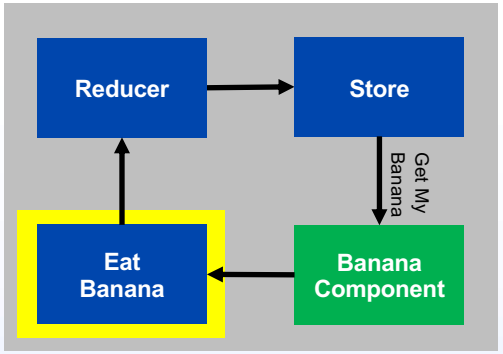
- https://bit.ly/2GV8XY9
- http://stackblitz.com/github/apaytonmn/bananaapp-checkpoint-two

Action with payload

- What if I need additional information to figure out how my state needs to change?
  - Ex. Retrieve info by record ID
- Action follows same pattern
  - Pass data as payload on dispatch
  - Handle payload in the reducer

Action: Eat Banana

- Add the Eat Banana action to your actions file



banana.actions.ts

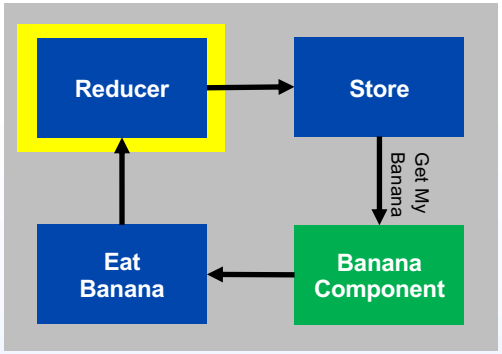
```
export const EAT_BANANA = 'Eat Banana';
```

```
export class EatBanana implements Action {
    readonly type: string = EAT_BANANA;

    constructor(public payload: number) {
        console.log('ACTION ' + EAT_BANANA);
    }
}
```

```
export type BananaAction = GetNewBanana | PeelBanana | EatBanana;
```

Reducer: Eat Banana

- Handle the Eat Banana action in your reducer



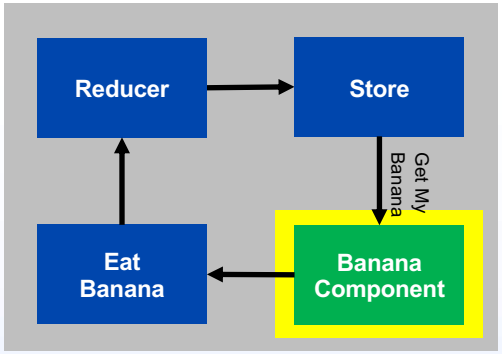
banana.reducer.ts

```
import { GET_NEW_BANANA, PEEL_BANANA, EAT_BANANA } from './banana.actions';
```

```
case EAT_BANANA: {
  console.log('REDUCER: Taking ' + action.payload + ' bites of the banana')
  return {
    ...state,
    bitesRemaining: (state.bitesRemaining - action.payload)
  };
}
```

Component: Eat Banana

- Dispatch the Eat Banana action from your component

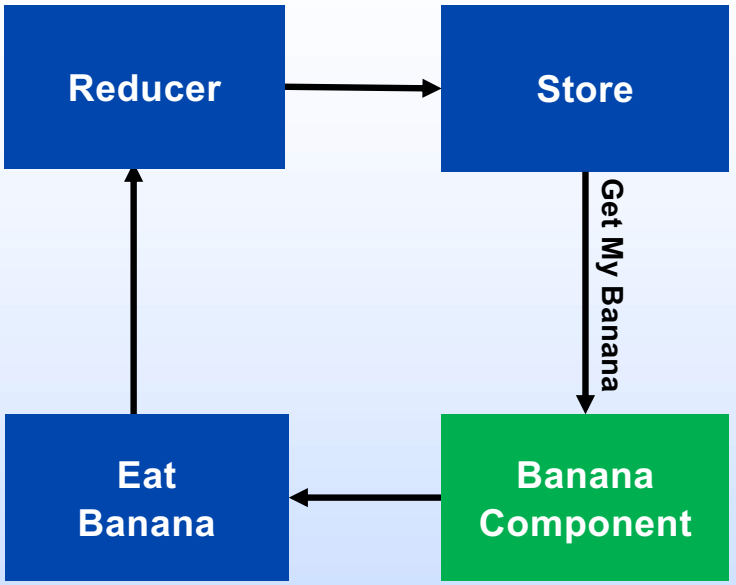


banana.component.ts

```
import { GetNewBanana, PeelBanana, EatBanana } from './state';
```

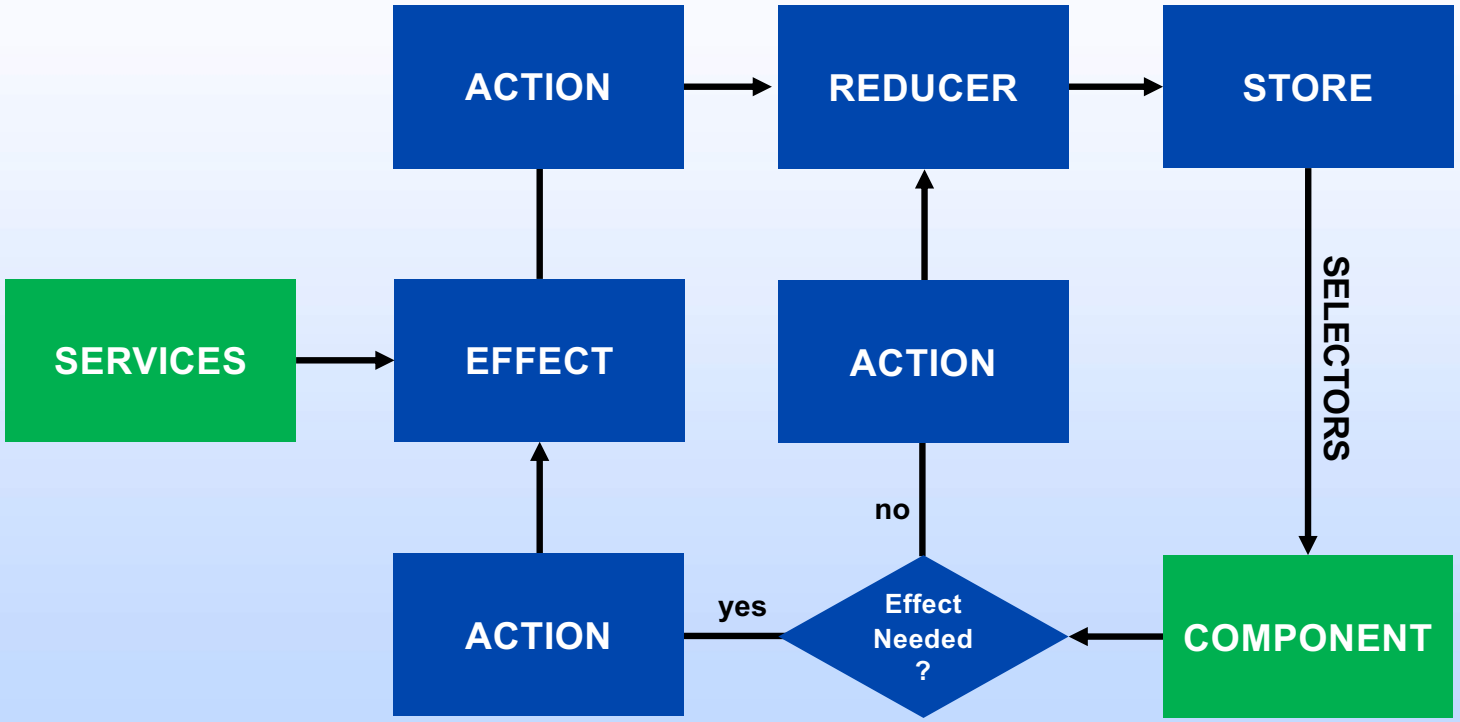
```
eatBanana() {
  this.store.dispatch(new EatBanana(3));
}
```

CHECKPOINT #3

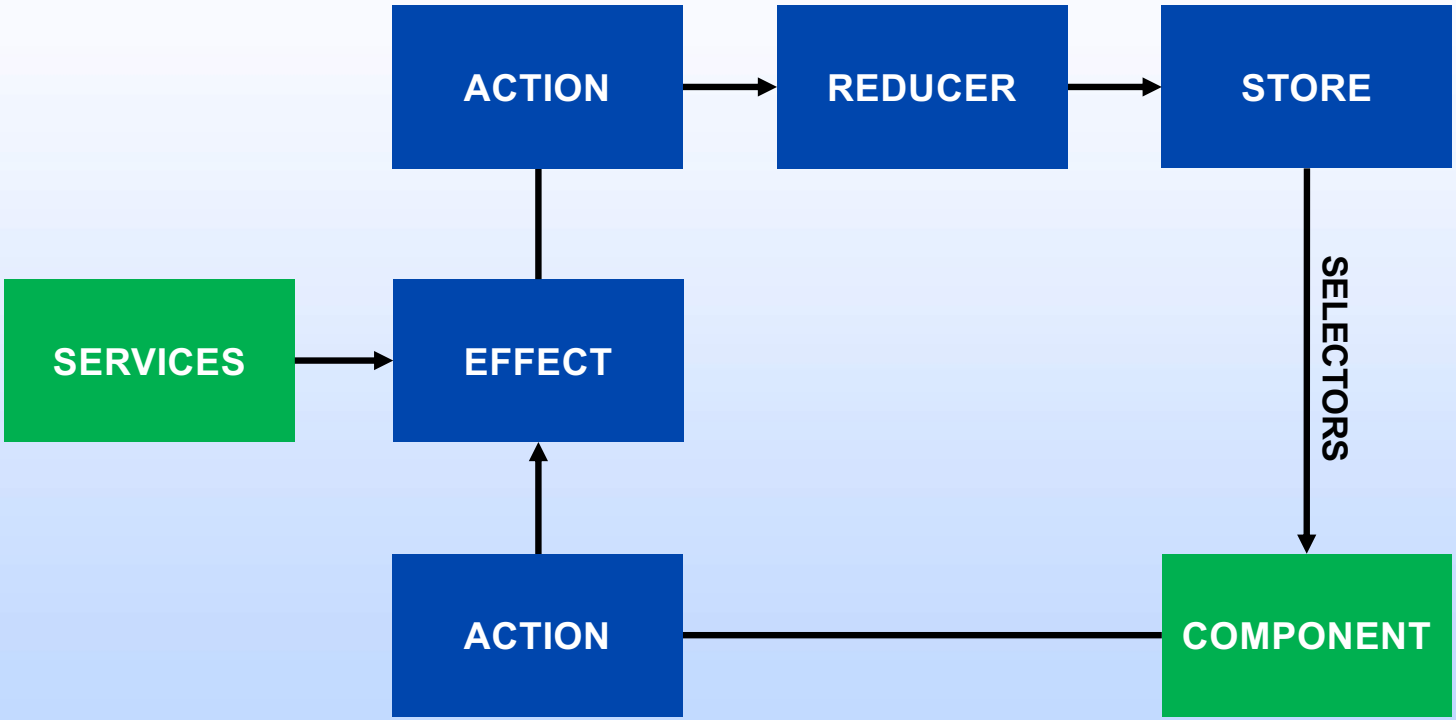


- https://bit.ly/2GV8XY9
- http://stackblitz.com/github/apaytonmn/bananaapp-checkpoint-three

ngRx Extended Pattern

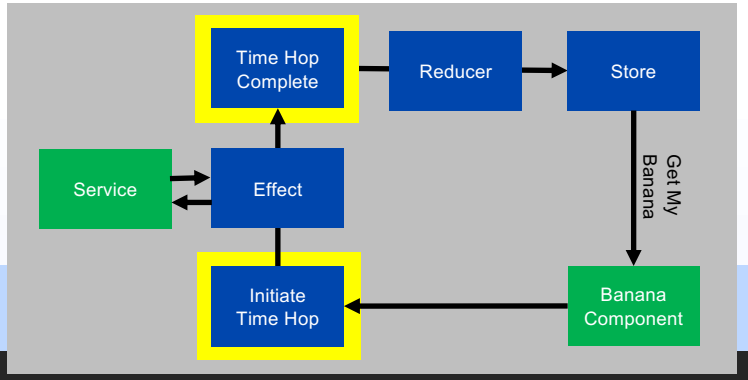


ngRx Extended Pattern



Actions: Initiate Time Hop & Time Hop Complete

- Add new actions to your actions file



banana.actions.ts

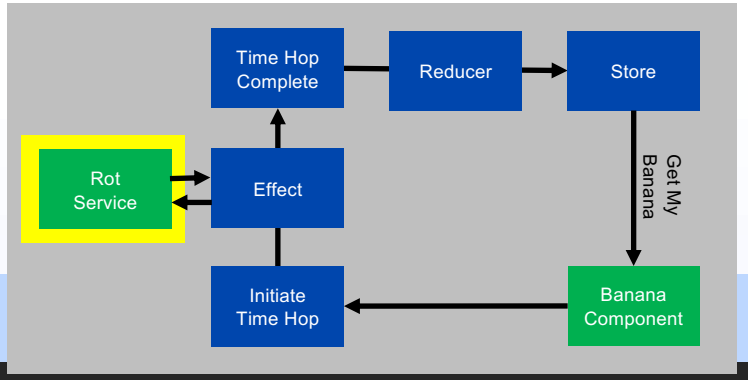
```
export const INITIATE_TIME_HOP = 'Initiate Time Hop';
export const TIME_HOP_COMPLETE = 'Time Hop Complete';
```

```
export class InitiateTimeHop implements Action {
  readonly type: string = INITIATE_TIME_HOP;

  constructor(public payload: any) {
    console.log('ACTION ' + INITIATE_TIME_HOP);
  }
}
export class TimeHopComplete implements Action {
  readonly type: string = TIME_HOP_COMPLETE;

  constructor(public payload: any) {
    console.log('ACTION ' + TIME_HOP_COMPLETE);
  }
}
```

```
export type BananaAction = GetNewBanana | PeelBanana | EatBanana | InitiateTimeHop | TimeHopComplete;
```

Service: Rot Service

- At app level, create new file for service

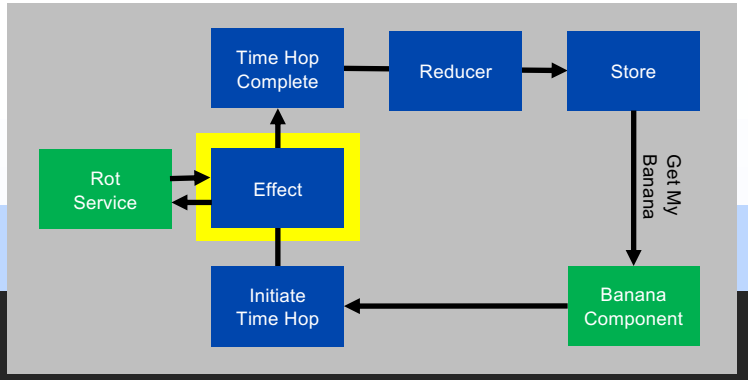


rot.service.ts

```
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs/Observable';

@Injectable({providedIn: 'root'})
export class RotService {

  rotBanana(): Observable<any> {
    console.log('ROT BANANA');
    const milliseconds = 10000; // 10 seconds
    return Observable.create(observer => {
      setTimeout(() => {
        console.log('Done waiting');
        observer.next('brown');
        observer.complete();
      }, milliseconds);
    });
  }
}
```

Effects

- In banana/state directory, create effects file

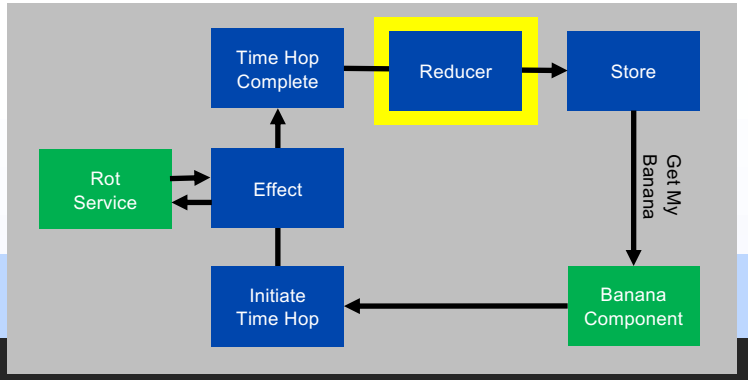


banana.effects.ts

```
import { Injectable } from '@angular/core';
import { Actions, Effect, ofType } from '@ngrx/effects';
import { switchMap, map } from 'rxjs/operators';
import { INITIATE_TIME_HOP, InitiateTimeHop, TimeHopComplete} from './banana.actions';
import { RotService } from '../../rot.service';

@Injectable()
export class BananaEffects {
  constructor(private actions$: Actions, private rot: RotService) { }

  @Effect()
  public initiateTimeHop$ = this.actions$.pipe(
    ofType(INITIATE_TIME_HOP),
    switchMap((action: InitiateTimeHop) =>
      this.rot.rotBanana().pipe(
        map(color => new TimeHopComplete(color))
      ),
    ),
  );
}
```

Reducer: Time Hop Complete

- Handle the Time Hop Complete action in your reducer



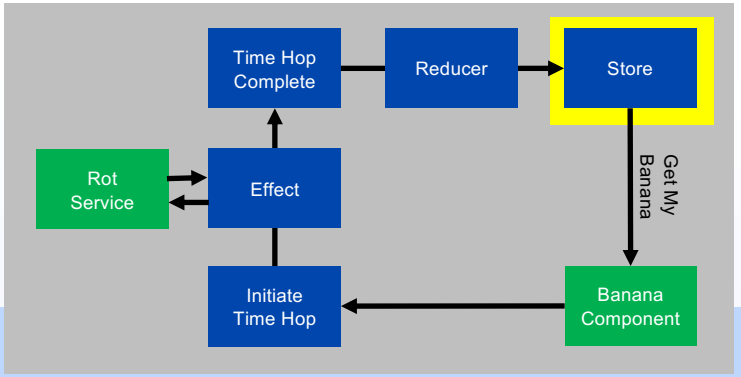
banana.reducer.ts

```
import { GET_NEW_BANANA, PEEL_BANANA, EAT_BANANA, TIME_HOP_COMPLETE } from './banana.actions';
```

```
  case TIME_HOP_COMPLETE: {
    console.log('REDUCER: Time hop complete')
    return {
      ...state,
      color: action.payload
    }
  }
```

Wire It Up!

- Tie the work we did in at the app level



index.ts

```
export { BananaEffects } from './banana.effects';
```

app.state.ts

```
export const effects: Array<any> = [
   bananaStore.BananaEffects
 ];
```

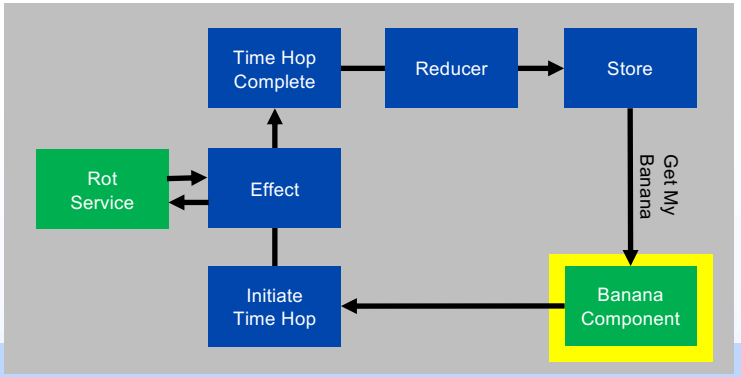
app.module.ts

```
import { initialState, reducers, effects } from './app.state';
import { RotService } from './rot.service';
import { EffectsModule } from '@ngrx/effects';

EffectsModule.forRoot(effects),
```

In imports

Component: Initiate Time Hop

- Dispatch the Initiate Time Hop action from your component

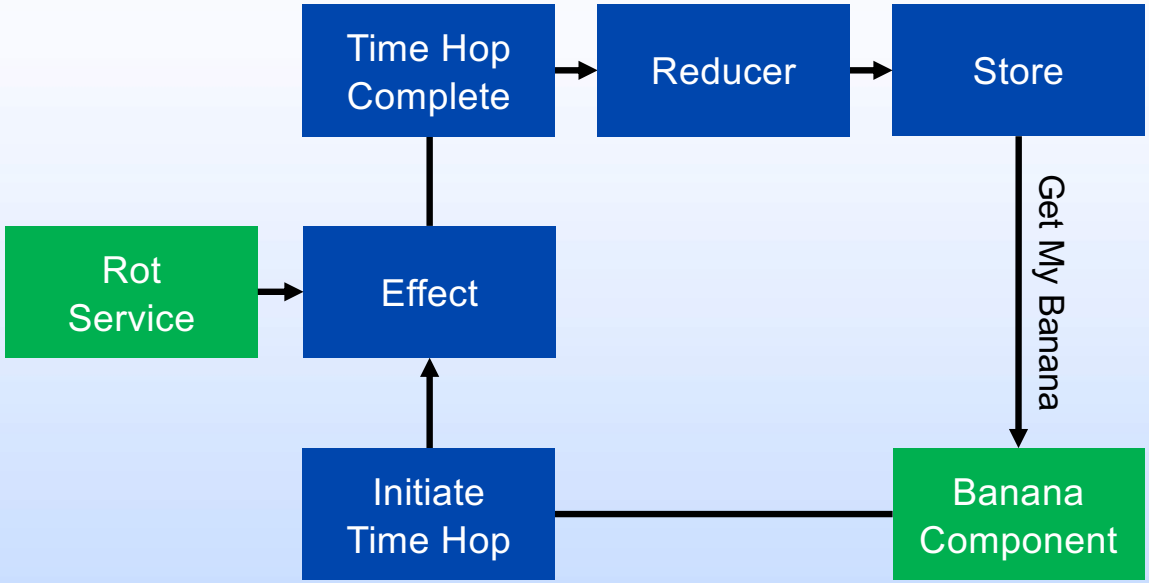


banana.component.ts

```
import { GetNewBanana, PeelBanana, EatBanana, InitiateTimeHop } from './state';
```

```
timeHop() {
  this.store.dispatch(new InitiateTimeHop(null));
}
```

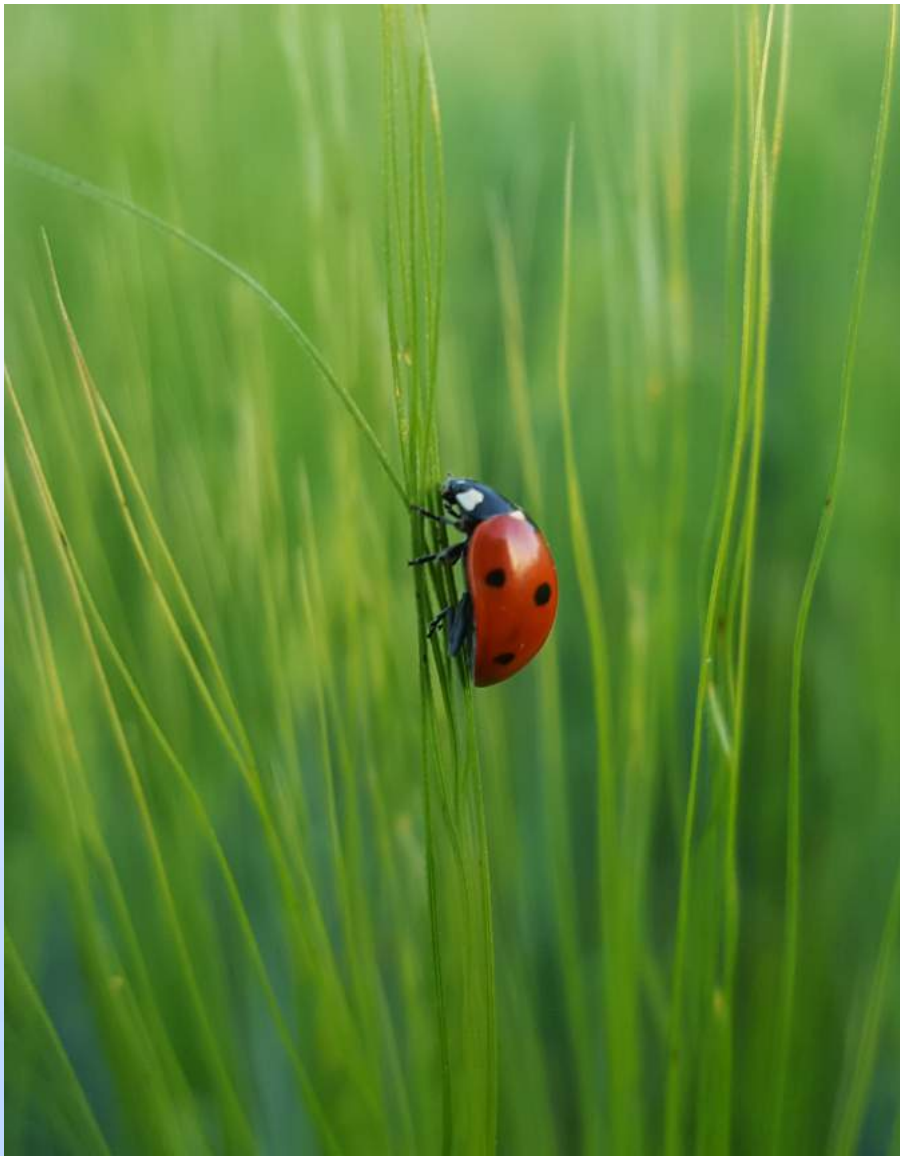
FINAL CHECKPOINT



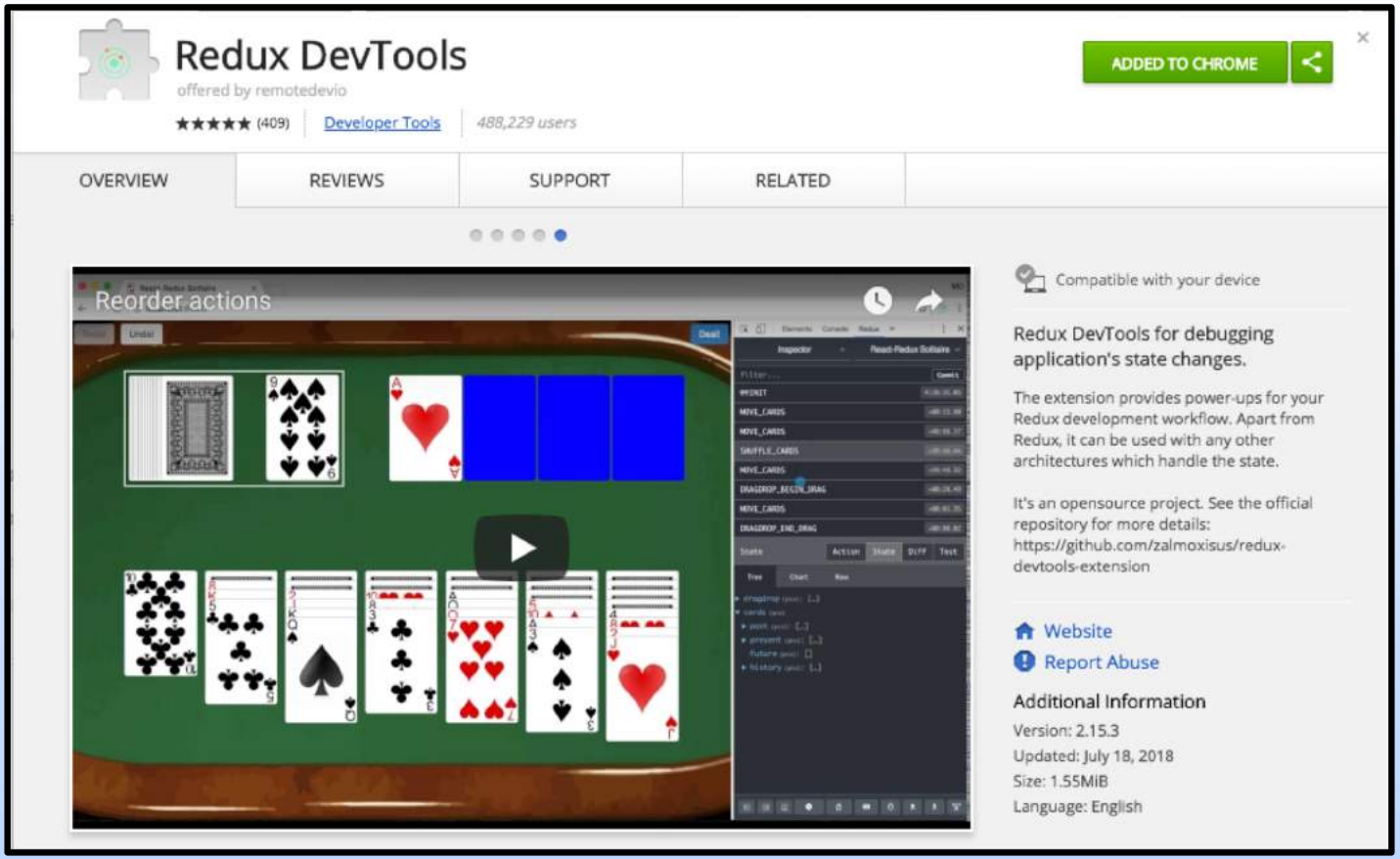
- http://stackblitz.com/github/apaytonmn/bananaapp-final

Debug with Redux DevTools

- HUGE benefit of NgRx
- Maintains history of actions
- Look “back in time” at state of application
- Easy to set up and use
- Feed state object for test



Debug with Redux DevTools

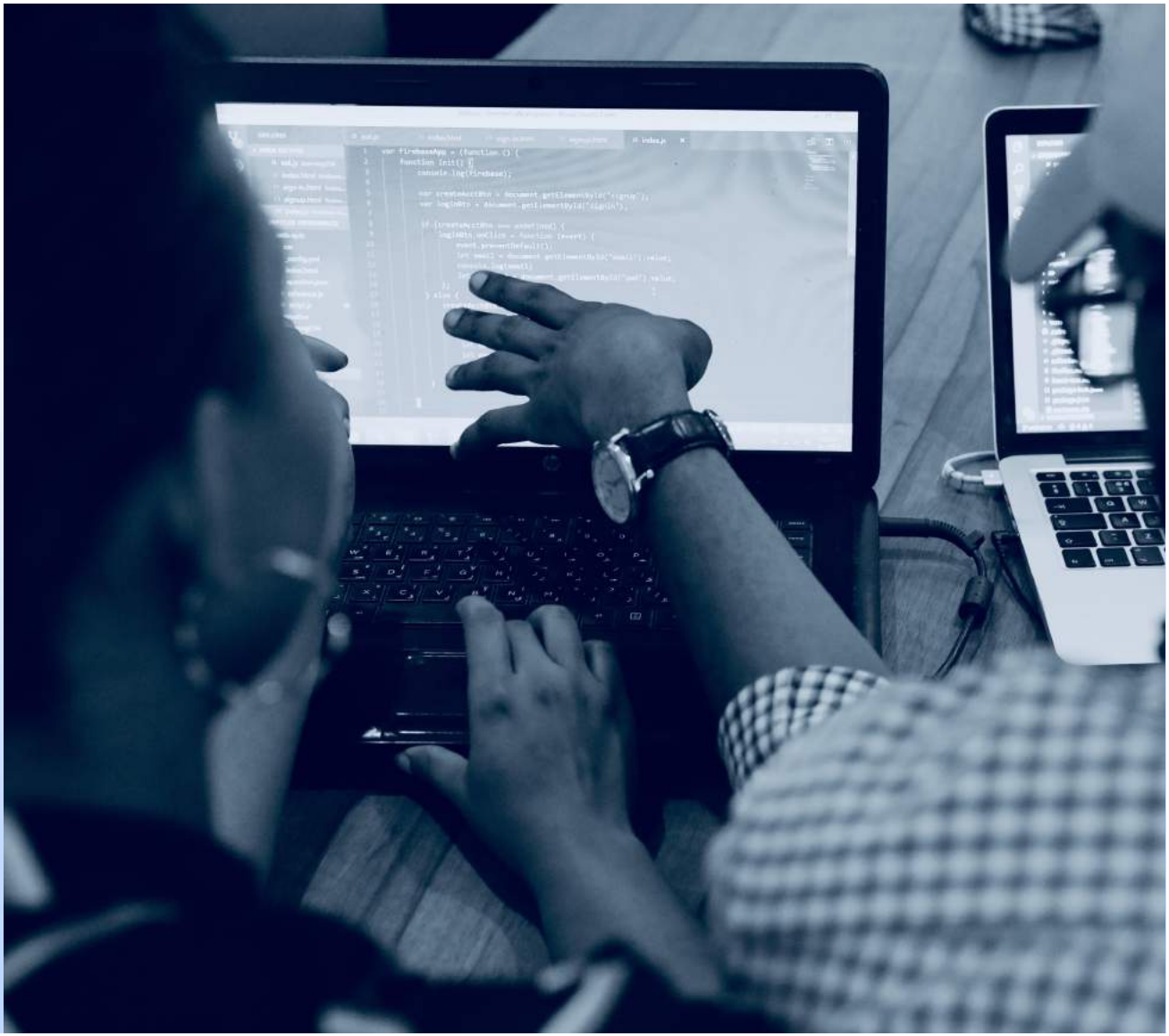


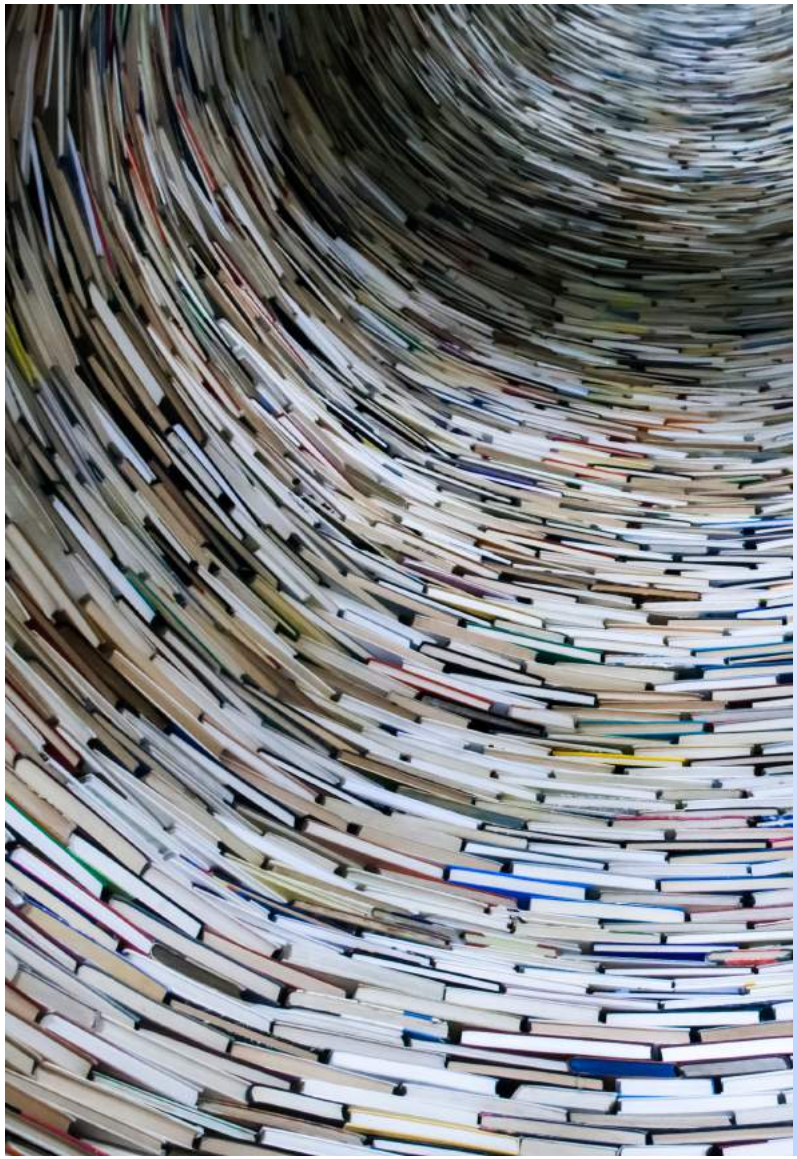
Debug with Redux DevTools

app.module.ts

```
import { Store, StoreModule } from '@ngrx/store';
import { EffectsModule } from '@ngrx/effects';
import { StoreDevtoolsModule } from '@ngrx/store-devtools';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    ...
    RouterModule.forRoot(appRoutes),
    StoreModule.forRoot(reducers, { initialState }),
    EffectsModule.forRoot(effects),
    StoreDevtoolsModule.instrument({
      maxAge: 25
    })
  ],
  providers: [ ],
  bootstrap: [
    AppComponent
  ]
})
```

DEMO
Redux DevTools

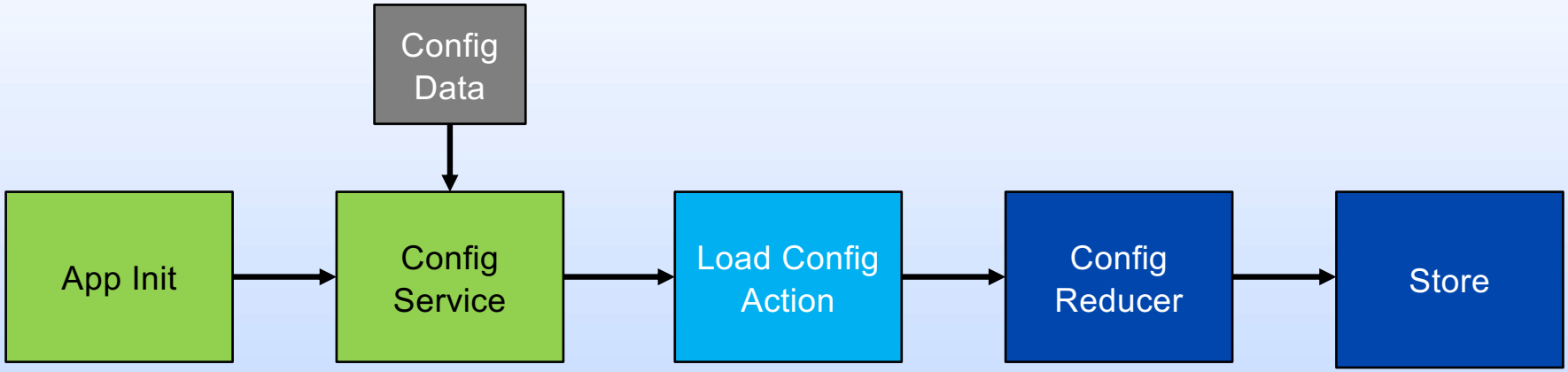




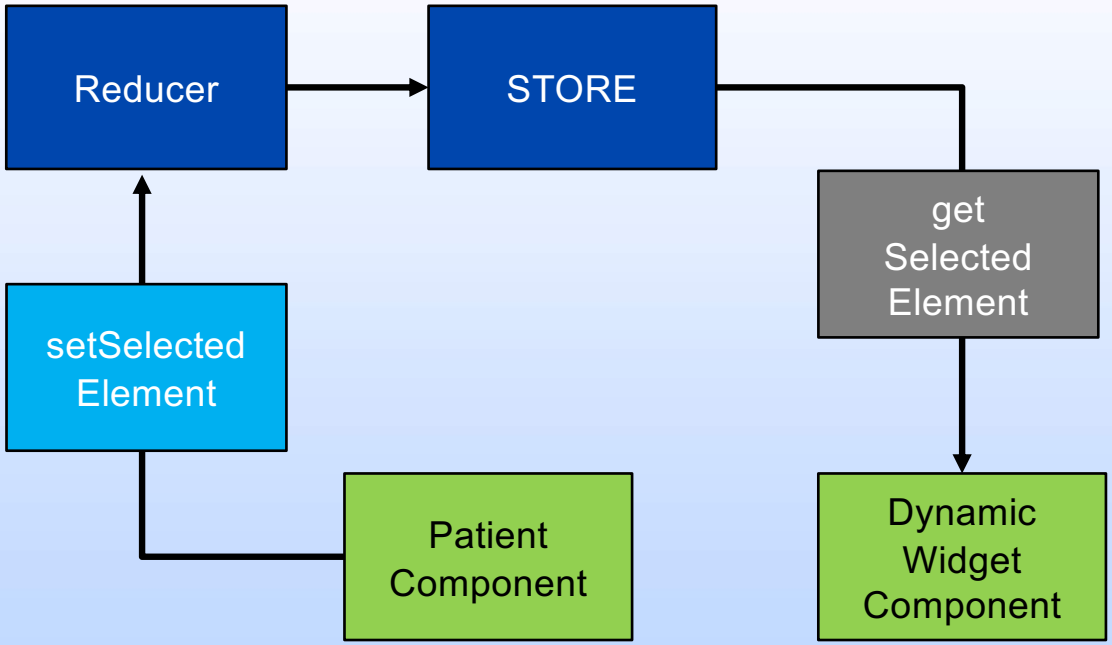
Extending NgRx complexity

- Introducing modules
- Config load at app initialization
- Integration with dynamic components

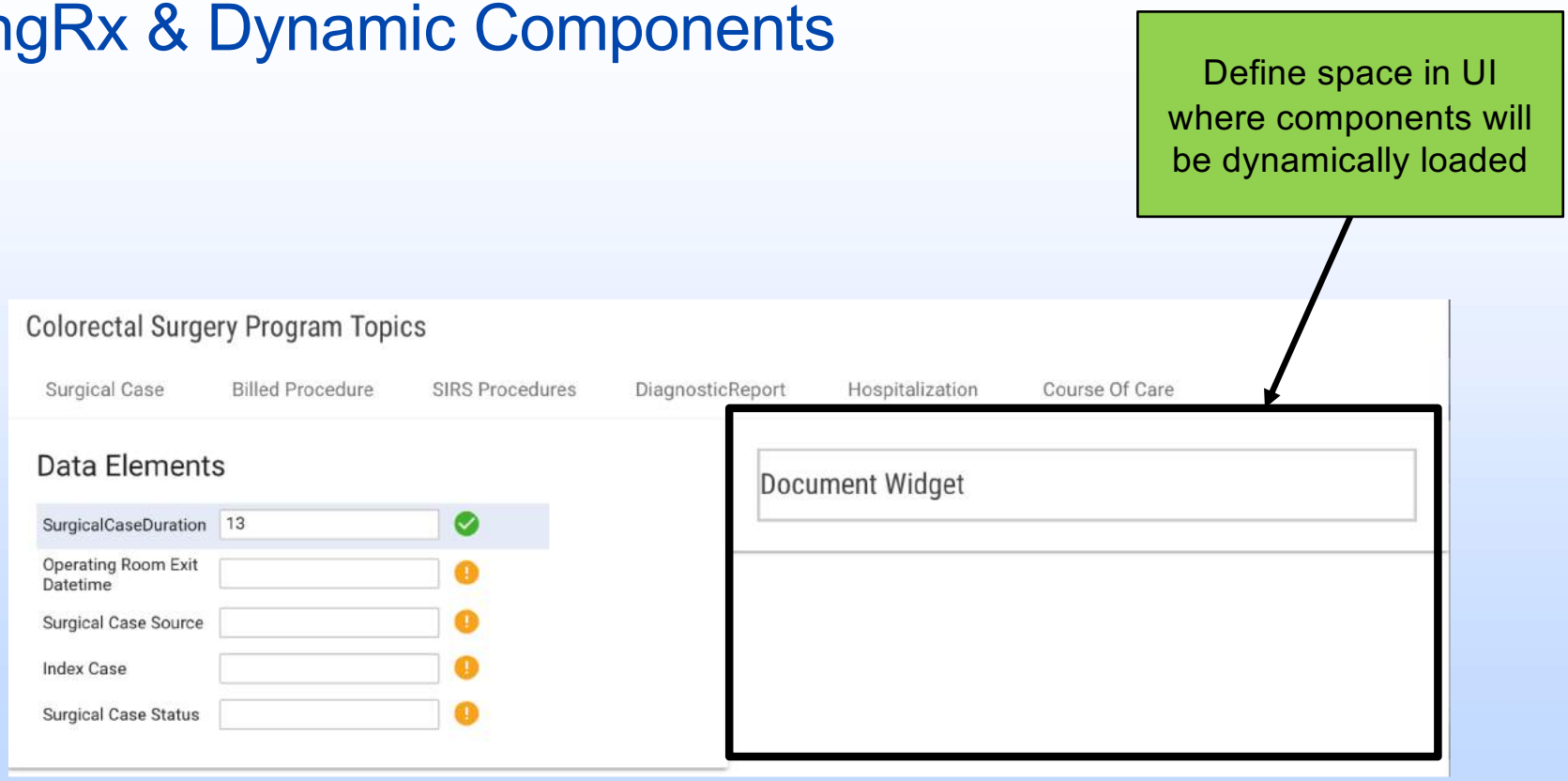
ngRx & App Initialization



ngRx & Dynamic Components



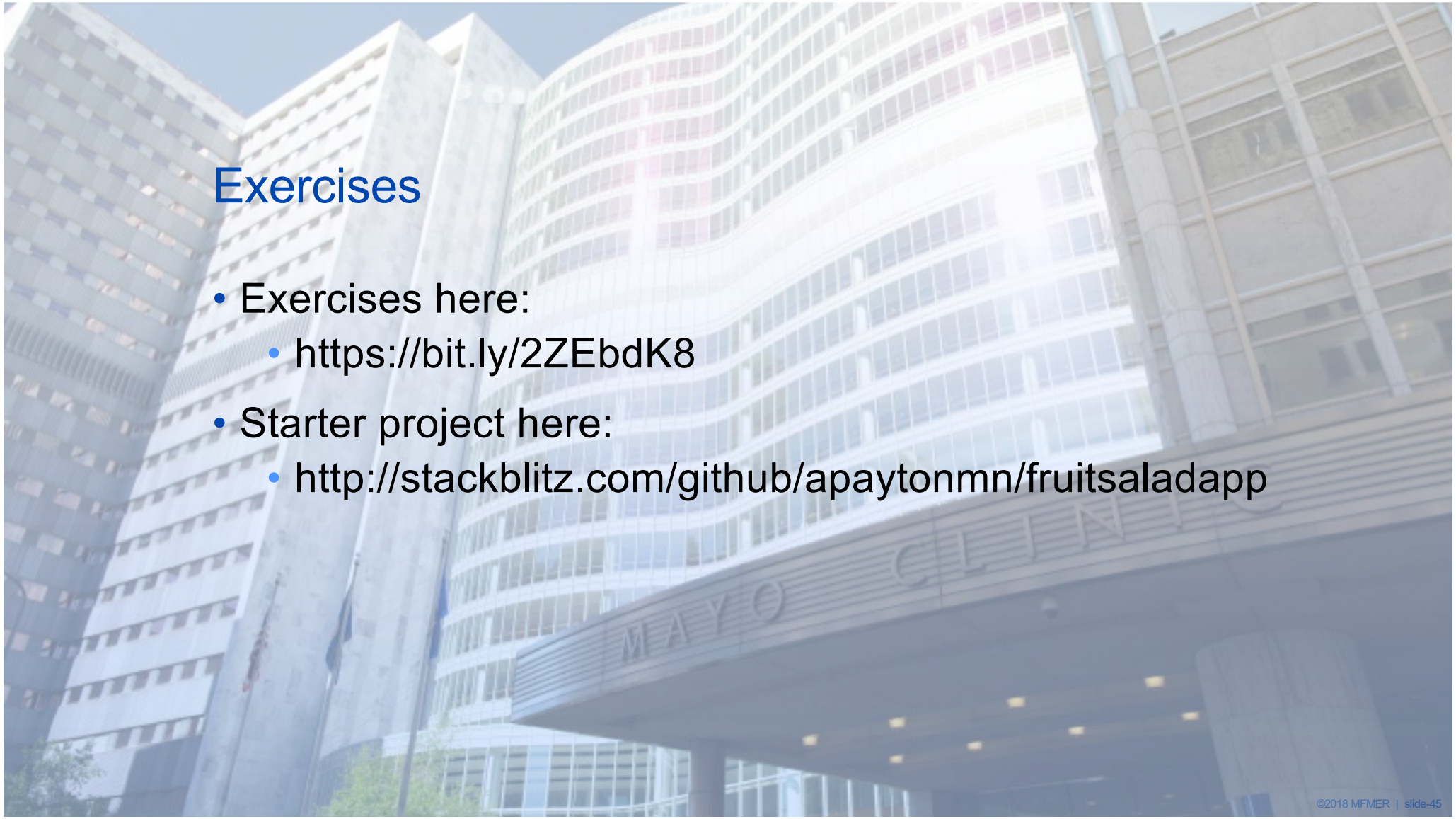
ngRx & Dynamic Components



Resources

- Presentation Slides
  - https://bit.ly/2GV8XY9
- Explain Redux Like I'm Five
  - https://dev.to/hemanth/explain-redux-like-im-five
- NgRx: Patterns and Techniques
  - https://blog.nrwl.io/ngrx-patterns-and-techniques-f46126e2b1e5
- Mayo Clinic
  - www.mayo.edu


MAYO CLINIC
Midwest JS
Contact Me
Aspen Payton
Email: payton.aspen@mayo.edu
Twitter: @paytonmn
Github: apaytonmn
©2018 MFMER | slide-44



Exercises

- Exercises here:
  - https://bit.ly/2ZEbdK8
- Starter project here:
  - http://stackblitz.com/github/apaytonmn/fruitsaladapp