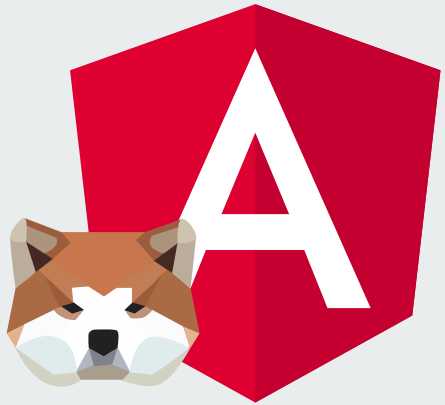
Angular State Management with Akita



Presenters

Rishanthakumar Rasarathinam

SSE@99X

rishanthakumar@gmail.com

github.com/rishanthakumar

@rishanthakumar

Dishan Rajapaksha

SSE@99X

hello@dishanrajapaksha.com

github.com/ dishanrajapaksha

@dishrajapaksha

Agenda

1. Angular State Management
2. State Management with Akita
3. Demo
4. Q & A

Angular State Management

Application State

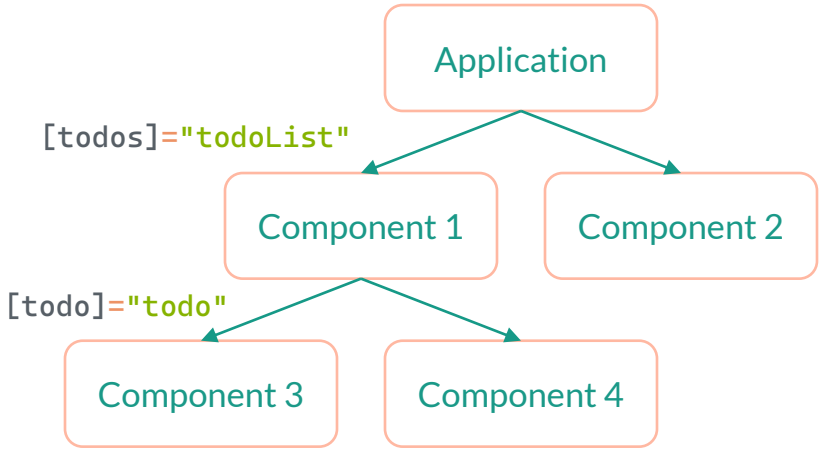
- What's stored in an application's memory
- State of the variables
    - Data from the APIs
    - State of the presentation UI
    - State of the local user preferences

Angular Application State

- Multiple components
- Each component has its own state

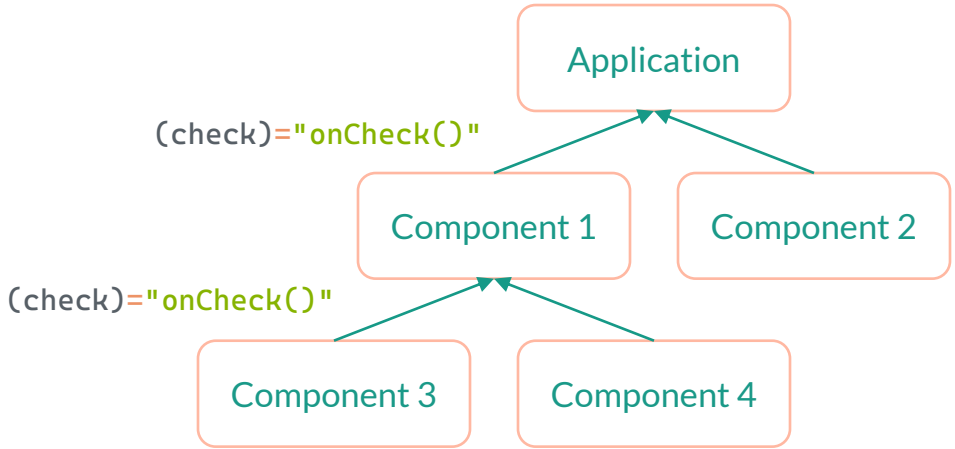
Angular Application State

- Property Binding @Input



Angular Application State

- Event Binding @Output



Angular Application State

- Sharing data using a Service

```
@Injectable()
export class DataService {

  private messageSource = new BehaviorSubject('default');

  currentMessage = this.messageSource.asObservable();

  constructor() { }

  changeMessage(message: string) {
    this.messageSource.next(message)
  }
}
```

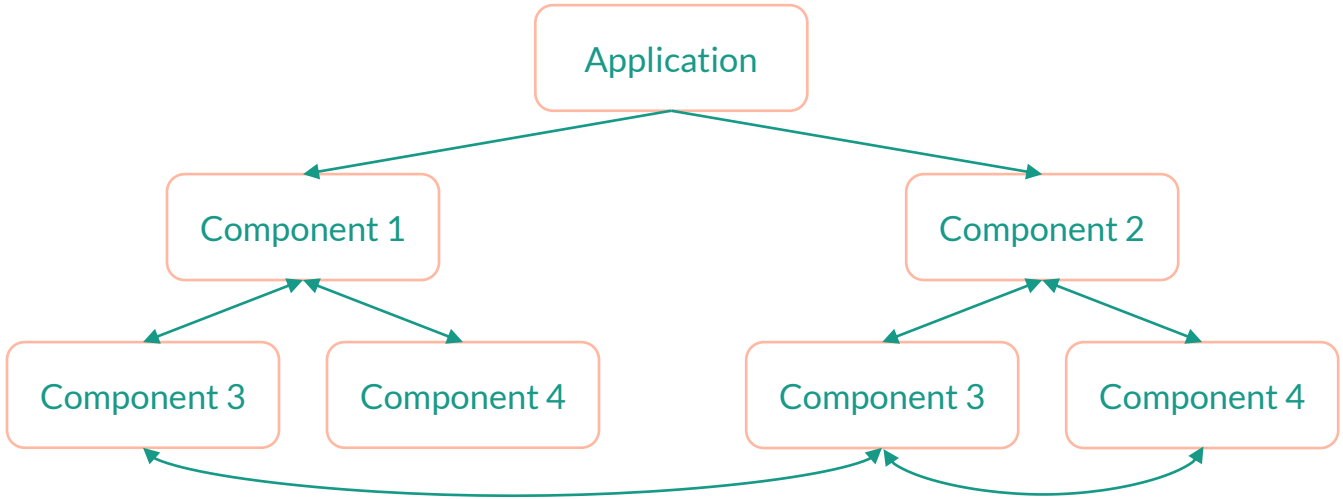
Angular Application State

- Sharing data using a Service

```
export class Component1 {

    constructor(private data: DataService) { }

    newMessage() {
      this.data.changeMessage("Hello from Sibling");
    }
}

export class Component2 implements OnInit {
  message:string;

  constructor(private data: DataService) { }

  ngOnInit() {
    this.data.currentMessage.subscribe((message) =>{
       this.message = message;
    });
  }
}
```

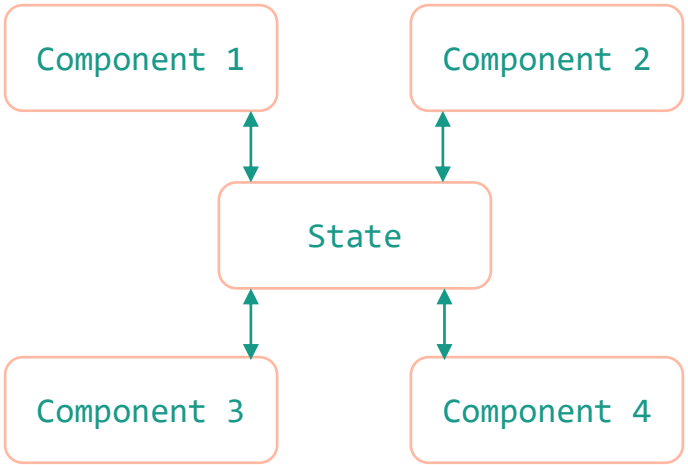
Angular Application State

- Will become complex and harder to scale with multiple components
- Harder to track changes and debug *Who is changing what in where?*
- Mutable



State Management

- Single source of truth for all the application data
- Easier to track changes
- Easier debugging

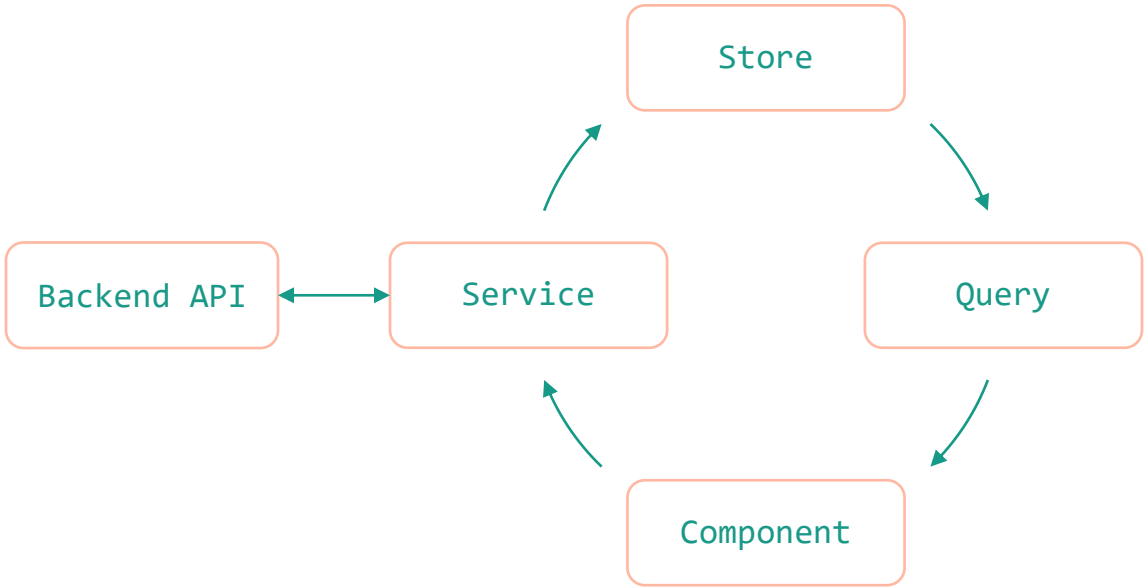


State management with Akita

Why Akita?

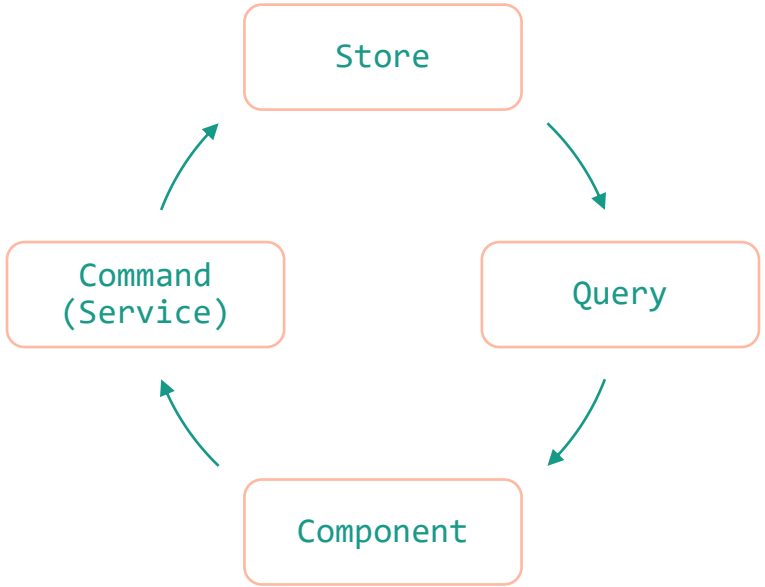
- Encourages simplicity
- Beginner friendly
- Built in enhancers
- Less boilerplate and code verbosity
- Based on object-oriented design principles
- Fast growing community

Akita Life Cycle



Akita Life Cycle

- Enables Command–query separation (CQRS)
- Every method should be either a command that performs an action or a query that returns a data



Store

- Single object
- Store operations
  - update()
  - setLoading()
  - setError()
  - reset()

```
export interface AuthState {
   username: string;
}

export function createInitialState(): AuthState {
  return {
    username: ''
  };
}

@Injectable({ providedIn: 'root' })
@StoreConfig({ name: 'auth', resettable: true })
export class AuthStore extends Store<AuthState> {

  constructor() {
    super(createInitialState());
  }
}
```

Service

- Update store values
- Async updates

Service

- Update store values
- Async updates

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

  constructor(private authStore: AuthStore){
  }

  updateUsername(username: string): void {
    this.authStore.update({
      username
    });
  }
}
```

Service

- Update store values
- Async updates

```
export class TodoService {

  constructor(private todoStore: TodoStore,
              private http: HttpClient) {
    super(todoStore);
  }

  get() {
    return this.http.get<Todo[]>('/api/todos').pipe(
      tap(entities => {
        this.todoStore.set(entities);
      })
    );
  }
}
```

Query

- Querying the store data
- Query operations
  - select()
  - getValue()
  - selectLoading()
  - selectError()

```
@Injectable({ providedIn: 'root' })
export class AuthQuery extends Query<AuthState> {

  constructor(protected store: AuthStore) {
    super(store);
  }
}
```

Query

- Querying the store data
- Query operations
  - select()
  - getValue()
  - selectLoading()
  - selectError()

```
@Injectable({ providedIn: 'root' })
export class AuthQuery extends Query<AuthState> {

  constructor(protected store: AuthStore) {
    super(store);
  }

  selectUsername(): Observable<string> {
    return this.select(state => state.username);
  }
}
```

Query

- Querying the store data
- Query operations
  - select()
  - getValue()
  - selectLoading()
  - selectError()

```
@Injectable({ providedIn: 'root' })
export class AuthQuery extends Query<AuthState> {

  constructor(protected store: AuthStore) {
    super(store);
  }

  selectUsername(): Observable<string> {
    return this.select(state => state.username);
  }

  getUsername(): string {
    return this.getValue().username;
  }
}
```

Entity Store

- Collection of entities
- Easier CRUD operations
  - add()
  - remove()
  - set()
  - update()
  - upsert()
  - replace()

```
export interface TodoState extends EntityState<Todo> {}

@Injectable({ providedIn: 'root' })
@StoreConfig({ name: 'todo', resettable: true })
export class TodoStore extends EntityStore<TodoState> {

  constructor() {
    super();
  }
}


@Injectable({ providedIn: 'root' })
export class TodoService extends NgEntityService<TodoState> {

  constructor(protected store: TodoStore) {
    super(store);
  }
}
```

Entity Store

- Collection of entities
- Easier CRUD operations
  - add()
  - remove()
  - set()
  - update()
  - upsert()
  - replace()

```
add(): void {
  const title = this.formControl.value;
  if (title?.trim()) {
    this.todoService.add({title});
  }
}
```

Entity Store

- Collection of entities
- Easier CRUD operations
  - add()
  - remove()
  - set()
  - update()
  - upsert()
  - replace()

```
onDelete(id: string): void {
  this.todoService.delete(id);
}
```

Entity Query

- SQL like querying
  - selectAll()
  - select()
  - selectCount()
  - selectmany()
- Getters
  - getAll()
  - getEntity()
  - hasEntity()

```
@Injectable({ providedIn: 'root' })
export class TodoQuery extends QueryEntity<TodoState>{

  constructor(protected store: TodoStore) {
    super(store);
  }
}
```

Entity Query

- SQL like querying
  - selectAll()
  - select()
  - selectCount()
  - selectmany()
- Getters
  - getAll()
  - getEntity()
  - hasEntity()

```
export class TodosComponent implements OnInit {

  todos$: Observable<Todo[]>;

  constructor(private todoQuery: TodoQuery) { }

  ngOnInit(): void {
    this.todos$ = this.todoQuery.selectAll();
  }
}
```

Entity Query

- SQL like querying
  - selectAll()
  - select()
  - selectCount()
  - selectmany()
- Getters
  - getAll()
  - getEntity()
  - hasEntity()

```
export class TodosComponent implements OnInit {

  completedTodos$: Observable<ToDo>

  constructor(private todoQuery: TodoQuery) { }

  ngOnInit(): void {
    this.completedTodos$ = query.selectAll({
            filterBy: [
             (entity, index) => index % 2 === 0,
             ({ completed }) => !!completed
           ]
         });
  }
}
```

Entity Query

- SQL like querying
  - selectAll()
  - select()
  - selectCount()
  - selectmany()
- Getters
  - getAll()
  - getEntity()
  - hasEntity()

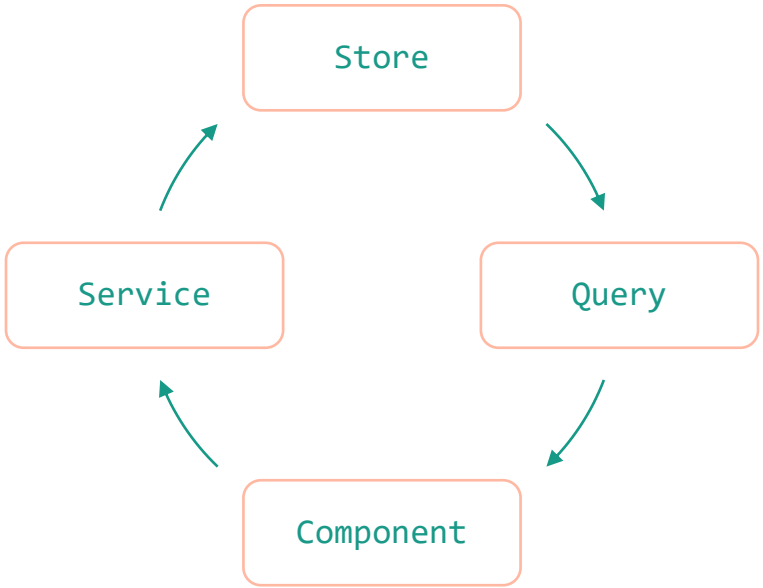
```
export class TodosComponent implements OnInit {

  todos: Todo[];

  constructor(private todoQuery: TodoQuery) { }

  ngOnInit(): void {
    this.todos = this.todoQuery.getAll();
  }
}
```

Combined Queries

- Combine queries from multiple stores

```
export class MoviesQuery extends QueryEntity<MoviesState>
{

    constructor(protected store: MoviesStore,
                private actorsQuery: ActorsQuery,
                private genresQuery: GenresQuery) {
      super(store);
    }

    selectMovies() {
      return combineQueries([
        this.selectAll(),
        this.actorsQuery.selectAll({ asObject: true }),
        this.genresQuery.selectAll({ asObject: true })
       ]
      )
      .pipe(
        map(([movies, actors, genres]) => {
          return movies.map(movie => {
            return {
              ...movie,
              actors: movie.actors.map(id => actors[id]),
              genres: movie.genres.map(id => genres[id])
            };
          });
        })
      );
    }
  }
```

Akita Life Cycle



Features

- Transactions
- Local storage management
- Store middleware
- WebSocket support
- State history
- Dirty check

Features

- Transactions
- Local storage management
- Store middleware
- WebSocket support
- State history
- Dirty check

```
const authState$ = authQuery.select().subscribe();

@transaction()
update(token: string) {
  this.store.update({ token });
  this.store.setLoading(false);
}

update() {
  return http.get().pipe(
    withTransaction(response => {
       this.store.update(response);
       this.store.setActive(1);
    })
  );
}
```

Features

- Transactions
- Local storage management
- Store middleware
- WebSocket support
- State history
- Dirty check

```
persistState({
  include: ['auth'],
  storage: sessionStorage
});

persistState({
  include: ['auth.username'],
  storage: localStorage
});
```

Features

- Transactions
- Local storage management
- Store middleware
- WebSocket support
- State history
- Dirty check

```
@StoreConfig({ name: 'books' })
export class BooksStore extends EntityStore<BooksState>
{
  constructor() {
    super();
  }

  akitaPreAddEntity(book: Book) {
    if(book.price === 100) {
      return {
        ...book,
        price: limitedPrice
      }
    }

    return book;
  }
}
```

Akita Angular

- Angular schematics
- Ng Entity Service
- Local component states
- Angular forms and router state management

```
ng g @datorama/akita:feature todos/todo
```

```
todo
├ todo.model.ts
├ todo.query.ts
├ todo.service.ts
└ todo.store.ts
```

```
ng g @datorama/akita:as  todos  //Store
ng g @datorama/akita:aes todos // Entity store

ng g @datorama/akita:query todos
ng g @datorama/akita:entity-query todos

ng g @datorama/akita:service todos
```

Akita Angular

- Angular schematics
- Ng Entity Service
- Local component states
- Angular forms
- Angular router

```
@NgModule({
  ...
  providers: [
    {
      provide: NG_ENTITY_SERVICE_CONFIG,
      useValue: {
        baseUrl: 'https://jsonplaceholder.typicode.com'
      }
    }
  ],
  bootstrap: [AppComponent]
})
```

```
@Injectable({ providedIn: 'root' })
export class TodoService extends NgEntityService<TodoState> {
  constructor(protected store: TodoStore) {
    super(store);
  }
}
```

Akita Angular

- Angular schematics
- Ng Entity Service
- Local component states
- Angular forms
- Angular router

```
@Component()
export class TodoComponent {
  constructor(
    private todoService: TodoService
  ) {}

  get(id) {
    this.todoService.get(id).subscribe();
  }

  add() {
    this.todoService.add({ title: 'Hello', body: '' }).subscribe();
  }

  update(id) {
    this.todoService.update(id, { title: 'Hello world' }).subscribe();
  }

  remove(id) {
    this.todoService.delete(id).subscribe();
  }
}
```

Demo

References

- Akita documentation
- Akita Gitter
- Front-end repo
- API repo

Q & A

Thank you