# Angular Complete Guide: Routing, JWT Auth, NgRx, WebSockets

## 1. Multiple Routes and Authentication

### Step 1: Install Angular Router

```bash
ng add @angular/router
```

### Step 2: Basic Route Configuration

**app-routing.module.ts**

```typescript
```

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { LoginComponent } from './login/login.component';
import { DashboardComponent } from './dashboard/dashboard.component';
import { ProfileComponent } from './profile/profile.component';
import { AuthGuard } from './guards/auth.guard';

const routes: Routes = [
  { path: '', redirectTo: '/login', pathMatch: 'full' },
  { path: 'login', component: LoginComponent },
  {
    path: 'dashboard',
    component: DashboardComponent,
    canActivate: [AuthGuard]
  },
  {
    path: 'profile',
    component: ProfileComponent,
    canActivate: [AuthGuard]
  },
  { path: '**', redirectTo: '/login' }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

## Step 3: Create Auth Guard

### guards/auth.guard.ts

```
typescript
```

```
import { Injectable } from '@angular/core';
import { CanActivate, Router } from '@angular/router';
import { AuthService } from '../services/auth.service';

@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {
  constructor(private auth: AuthService, private router: Router) {}

  canActivate(): boolean {
    if (this.auth.isAuthenticated()) {
      return true;
    }
    this.router.navigate(['/login']);
    return false;
  }
}
```

### Step 4: Lazy Loading Routes

#### feature-routing.module.ts

```
typescript

const routes: Routes = [
  {
    path: 'admin',
    loadChildren: () => import('./admin/admin.module').then(m => m.AdminModule),
    canActivate: [AuthGuard]
  },
  {
    path: 'user',
    loadChildren: () => import('./user/user.module').then(m => m.UserModule)
  }
];
```

---

## 2. JWT Authentication with Angular

### Step 1: Install Dependencies

```
bash
```

```
npm install @auth0/angular-jwt
```

## Step 2: Create Auth Service

### services/auth.service.ts

```typescript
```

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { BehaviorSubject, Observable } from 'rxjs';
import { map } from 'rxjs/operators';
import { JwtHelperService } from '@auth0/angular-jwt';

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  private currentUserSubject: BehaviorSubject<any>;
  public currentUser: Observable<any>;
  private jwtHelper = new JwtHelperService();

  constructor(private http: HttpClient) {
    this.currentUserSubject = new BehaviorSubject<any>(
      JSON.parse(localStorage.getItem('currentUser') || '{}')
    );
    this.currentUser = this.currentUserSubject.asObservable();
  }

  public get currentUserValue() {
    return this.currentUserSubject.value;
  }

  login(email: string, password: string): Observable<any> {
    return this.http.post<any>('/api/auth/login', { email, password })
      .pipe(map(user => {
        localStorage.setItem('currentUser', JSON.stringify(user));
        localStorage.setItem('token', user.token);
        this.currentUserSubject.next(user);
        return user;
      }));
  }

  logout() {
    localStorage.removeItem('currentUser');
    localStorage.removeItem('token');
    this.currentUserSubject.next(null);
  }

  isAuthenticated(): boolean {
    const token = localStorage.getItem('token');
```

```
    return token ? !this.jwtHelper.isTokenExpired(token) : false;
  }

  getToken(): string | null {
    return localStorage.getItem('token');
  }
}
```

## Step 3: Create JWT Interceptor

### interceptors/jwt.interceptor.ts

typescript

```
import { Injectable } from '@angular/core';
import { HttpRequest, HttpHandler, HttpInterceptor } from '@angular/common/http';
import { AuthService } from '../services/auth.service';

@Injectable()
export class JwtInterceptor implements HttpInterceptor {
  constructor(private authService: AuthService) {}

  intercept(request: HttpRequest<any>, next: HttpHandler) {
    const currentUser = this.authService.currentUserValue;
    const token = this.authService.getToken();

    if (currentUser && token) {
      request = request.clone({
        setHeaders: {
          Authorization: `Bearer ${token}`
        }
      });
    }

    return next.handle(request);
  }
}
```

## Step 4: Login Component

### login/login.component.ts

typescript

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { AuthService } from '../services/auth.service';

@Component({
  selector: 'app-login',
  template: `
    <form [formGroup]="loginForm" (ngSubmit)="onSubmit()">
      <div>
        <input type="email" formControlName="email" placeholder="Email" />
      </div>
      <div>
        <input type="password" formControlName="password" placeholder="Password" />
      </div>
      <button type="submit" [disabled]="loginForm.invalid">Login</button>
    </form>
  `
})
export class LoginComponent {
  loginForm: FormGroup;

  constructor(
    private fb: FormBuilder,
    private authService: AuthService,
    private router: Router
  ) {
    this.loginForm = this.fb.group({
      email: ['', [Validators.required, Validators.email]],
      password: ['', Validators.required]
    });
  }

  onSubmit() {
    if (this.loginForm.valid) {
      const { email, password } = this.loginForm.value;
      this.authService.login(email, password).subscribe(
        () => this.router.navigate(['/dashboard']),
        error => console.error('Login failed', error)
      );
    }
```

```
  }
}
```

---

## 3. State Management with NgRx

### Step 1: Install NgRx

```
bash

ng add @ngrx/store @ngrx/effects @ngrx/store-devtools
```

### Step 2: Define Actions

#### store/auth/auth.actions.ts

```
typescript

import { createAction, props } from '@ngrx/store';

export const login = createAction(
  '[Auth] Login',
  props<{ email: string; password: string }>()
);

export const loginSuccess = createAction(
  '[Auth] Login Success',
  props<{ user: any; token: string }>()
);

export const loginFailure = createAction(
  '[Auth] Login Failure',
  props<{ error: any }>()
);

export const logout = createAction('[Auth] Logout');
```

### Step 3: Create Reducer

#### store/auth/auth.reducer.ts

```
typescript
```

```typescript
import { createReducer, on } from '@ngrx/store';
import * as AuthActions from './auth.actions';

export interface AuthState {
  user: any;
  token: string | null;
  isAuthenticated: boolean;
  loading: boolean;
  error: any;
}

const initialState: AuthState = {
  user: null,
  token: null,
  isAuthenticated: false,
  loading: false,
  error: null
};

export const authReducer = createReducer(
  initialState,
  on(AuthActions.login, state => ({
    ...state,
    loading: true,
    error: null
  })),
  on(AuthActions.loginSuccess, (state, { user, token }) => ({
    ...state,
    user,
    token,
    isAuthenticated: true,
    loading: false
  })),
  on(AuthActions.loginFailure, (state, { error }) => ({
    ...state,
    error,
    loading: false
  })),
  on(AuthActions.logout, () => initialState)
);
```

## Step 4: Create Effects

### store/auth/auth.effects.ts

typescript

```
import { Injectable } from '@angular/core';
import { Actions, createEffect, ofType } from '@ngrx/effects';
import { of } from 'rxjs';
import { map, mergeMap, catchError, tap } from 'rxjs/operators';
import { AuthService } from '../../services/auth.service';
import * as AuthActions from './auth.actions';
import { Router } from '@angular/router';

@Injectable()
export class AuthEffects {
  login$ = createEffect(() =>
    this.actions$.pipe(
      ofType(AuthActions.login),
      mergeMap(action =>
        this.authService.login(action.email, action.password).pipe(
          map(response => AuthActions.loginSuccess({
            user: response.user,
            token: response.token
          })),
          catchError(error => of(AuthActions.loginFailure({ error })))
        )
      )
    )
  );

  loginSuccess$ = createEffect(() =>
    this.actions$.pipe(
      ofType(AuthActions.loginSuccess),
      tap(() => this.router.navigate(['/dashboard']))
    ), { dispatch: false }
  );

  constructor(
    private actions$: Actions,
    private authService: AuthService,
    private router: Router
  ) {}
}
```

## Step 5: Create Selectors

**store/auth/auth.selectors.ts**

```typescript
import { createSelector, createFeatureSelector } from '@ngrx/store';
import { AuthState } from './auth.reducer';

export const selectAuthState = createFeatureSelector<AuthState>('auth');

export const selectUser = createSelector(
  selectAuthState,
  (state: AuthState) => state.user
);

export const selectIsAuthenticated = createSelector(
  selectAuthState,
  (state: AuthState) => state.isAuthenticated
);

export const selectAuthLoading = createSelector(
  selectAuthState,
  (state: AuthState) => state.loading
);
```

## Step 6: Configure Store in App Module

### app.module.ts

```typescript
import { StoreModule } from '@ngrx/store';
import { EffectsModule } from '@ngrx/effects';
import { authReducer } from './store/auth/auth.reducer';
import { AuthEffects } from './store/auth/auth.effects';

@NgModule({
  imports: [
    StoreModule.forRoot({ auth: authReducer }),
    EffectsModule.forRoot([AuthEffects])
  ]
})
export class AppModule { }
```

## Step 7: Use Store in Component

### components/login.component.ts

typescript

```typescript
import { Component } from '@angular/core';
import { Store } from '@ngrx/store';
import { Observable } from 'rxjs';
import * as AuthActions from '../store/auth/auth.actions';
import { selectAuthLoading } from '../store/auth/auth.selectors';

@Component({
  selector: 'app-login',
  template: `
    <form (ngSubmit)="onLogin()">
      <input [(ngModel)]="email" type="email" placeholder="Email" />
      <input [(ngModel)]="password" type="password" placeholder="Password" />
      <button type="submit" [disabled]="loading$ | async">
        {{ (loading$ | async) ? 'Logging in...' : 'Login' }}
      </button>
    </form>
  `
})
export class LoginComponent {
  email = '';
  password = '';
  loading$: Observable<boolean>;

  constructor(private store: Store) {
    this.loading$ = this.store.select(selectAuthLoading);
  }

  onLogin() {
    this.store.dispatch(AuthActions.login({
      email: this.email,
      password: this.password
    }));
  }
}
```

---

## 4. WebSocket with RxJS

### Step 1: Create WebSocket Service

#### services/websocket.service.ts

typescript

```typescript
import { Injectable } from '@angular/core';
import { Observable, Subject, BehaviorSubject } from 'rxjs';
import { webSocket, WebSocketSubject } from 'rxjs/webSocket';
import { retry, tap, catchError } from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
})
export class WebSocketService {
  private socket$: WebSocketSubject<any> | undefined;
  private messagesSubject$ = new Subject<any>();
  private isConnected$ = new BehaviorSubject<boolean>(false);

  public messages$ = this.messagesSubject$.asObservable();
  public connectionStatus$ = this.isConnected$.asObservable();

  connect(url: string): void {
    if (!this.socket$ || this.socket$.closed) {
      this.socket$ = webSocket({
        url,
        openObserver: {
          next: () => {
            console.log('WebSocket connected');
            this.isConnected$.next(true);
          }
        },
        closeObserver: {
          next: () => {
            console.log('WebSocket disconnected');
            this.isConnected$.next(false);
          }
        }
      });

      this.socket$.pipe(
        retry(3),
        catchError(error => {
          console.error('WebSocket error:', error);
          this.isConnected$.next(false);
          throw error;
        })
      ).subscribe(
        message => this.messagesSubject$.next(message),
```

```
        error => console.error('WebSocket error:', error)
      );
    }
  }

  sendMessage(message: any): void {
    if (this.socket$) {
      this.socket$.next(message);
    }
  }

  disconnect(): void {
    if (this.socket$) {
      this.socket$.complete();
      this.isConnected$.next(false);
    }
  }

  // Subscribe to specific message types
  onMessage(type: string): Observable<any> {
    return this.messages$.pipe(
      tap(message => console.log('Received message:', message)),
      // Filter messages by type if needed
    );
  }
}
```

## Step 2: Create Chat Service (Example)

### services/chat.service.ts

typescript

```typescript
import { Injectable } from '@angular/core';
import { Observable, BehaviorSubject } from 'rxjs';
import { filter, map } from 'rxjs/operators';
import { WebSocketService } from './websocket.service';

export interface ChatMessage {
  id: string;
  user: string;
  message: string;
  timestamp: Date;
  type: 'message' | 'join' | 'leave';
}

@Injectable({
  providedIn: 'root'
})
export class ChatService {
  private messagesSubject$ = new BehaviorSubject<ChatMessage[]>([]);
  private onlineUsersSubject$ = new BehaviorSubject<string[]>([]);

  public messages$ = this.messagesSubject$.asObservable();
  public onlineUsers$ = this.onlineUsersSubject$.asObservable();

  constructor(private wsService: WebSocketService) {
    // Listen for chat messages
    this.wsService.messages$.pipe(
      filter(msg => msg.type === 'chat')
    ).subscribe(msg => {
      const currentMessages = this.messagesSubject$.value;
      this.messagesSubject$.next([...currentMessages, msg.data]);
    });

    // Listen for user list updates
    this.wsService.messages$.pipe(
      filter(msg => msg.type === 'users')
    ).subscribe(msg => {
      this.onlineUsersSubject$.next(msg.data);
    });
  }

  connect(): void {
    this.wsService.connect('ws://localhost:8080');
  }
```

```
  disconnect(): void {
    this.wsService.disconnect();
  }

  sendMessage(message: string, user: string): void {
    const chatMessage = {
      type: 'chat',
      data: {
        id: Date.now().toString(),
        user,
        message,
        timestamp: new Date(),
        type: 'message'
      }
    };
    this.wsService.sendMessage(chatMessage);
  }

  joinRoom(room: string, user: string): void {
    this.wsService.sendMessage({
      type: 'join',
      room,
      user
    });
  }
}
```

## Step 3: Chat Component

### components/chat.component.ts

```
typescript
```

```
import { Component, OnInit, OnDestroy } from '@angular/core';
import { Observable } from 'rxjs';
import { ChatService, ChatMessage } from '../services/chat.service';

@Component({
  selector: 'app-chat',
  template: `
    <div class="chat-container">
      <div class="messages">
        <div *ngFor="let message of messages$ | async" class="message">
          <strong>{{ message.user }}:</strong> {{ message.message }}
          <small>{{ message.timestamp | date:'short' }}</small>
        </div>
      </div>

      <div class="online-users">
        <h4>Online Users</h4>
        <ul>
          <li *ngFor="let user of onlineUsers$ | async">{{ user }}</li>
        </ul>
      </div>

      <div class="message-input">
        <input
          [(ngModel)]="newMessage"
          (keyup.enter)="sendMessage()"
          placeholder="Type a message..."
        />
        <button (click)="sendMessage()">Send</button>
      </div>
    </div>
  `,
  styles: [`
    .chat-container {
      display: flex;
      flex-direction: column;
      height: 500px;
    }
    .messages {
      flex: 1;
      overflow-y: auto;
      border: 1px solid #ccc;
      padding: 10px;
```

```
    }
    .message {
      margin: 5px 0;
      padding: 5px;
      border-bottom: 1px solid #eee;
    }
    .message-input {
      display: flex;
      gap: 10px;
      padding: 10px;
    }
    .message-input input {
      flex: 1;
      padding: 8px;
    }
  `]
})
export class ChatComponent implements OnInit, OnDestroy {
  messages$: Observable<ChatMessage[]>;
  onlineUsers$: Observable<string[]>;
  newMessage = '';
  currentUser = 'User' + Math.floor(Math.random() * 1000);

  constructor(private chatService: ChatService) {
    this.messages$ = this.chatService.messages$;
    this.onlineUsers$ = this.chatService.onlineUsers$;
  }

  ngOnInit(): void {
    this.chatService.connect();
    this.chatService.joinRoom('general', this.currentUser);
  }

  ngOnDestroy(): void {
    this.chatService.disconnect();
  }

  sendMessage(): void {
    if (this.newMessage.trim()) {
      this.chatService.sendMessage(this.newMessage, this.currentUser);
      this.newMessage = '';
    }
```

```
  }
}
```

## Step 4: Real-time Notifications Service

### services/notification.service.ts

typescript

```typescript
import { Injectable } from '@angular/core';
import { Observable, Subject } from 'rxjs';
import { filter } from 'rxjs/operators';
import { WebSocketService } from './websocket.service';

export interface Notification {
  id: string;
  title: string;
  message: string;
  type: 'info' | 'warning' | 'error' | 'success';
  timestamp: Date;
}

@Injectable({
  providedIn: 'root'
})
export class NotificationService {
  private notificationsSubject$ = new Subject<Notification>();
  public notifications$ = this.notificationsSubject$.asObservable();

  constructor(private wsService: WebSocketService) {
    // Listen for notifications
    this.wsService.messages$.pipe(
      filter(msg => msg.type === 'notification')
    ).subscribe(msg => {
      this.notificationsSubject$.next(msg.data);
    });
  }

  showNotification(notification: Notification): void {
    this.notificationsSubject$.next(notification);
  }
}
```

# Complete App Module Setup

## app.module.ts

typescript

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ReactiveFormsModule, FormsModule } from '@angular/forms';
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';
import { StoreModule } from '@ngrx/store';
import { EffectsModule } from '@ngrx/effects';
import { JwtModule } from '@auth0/angular-jwt';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { LoginComponent } from './components/login.component';
import { ChatComponent } from './components/chat.component';

import { JwtInterceptor } from './interceptors/jwt.interceptor';
import { authReducer } from './store/auth/auth.reducer';
import { AuthEffects } from './store/auth/auth.effects';

export function tokenGetter() {
  return localStorage.getItem('token');
}

@NgModule({
  declarations: [
    AppComponent,
    LoginComponent,
    ChatComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    ReactiveFormsModule,
    FormsModule,
    HttpClientModule,
    StoreModule.forRoot({ auth: authReducer }),
    EffectsModule.forRoot([AuthEffects]),
    JwtModule.forRoot({
      config: {
        tokenGetter: tokenGetter,
        allowedDomains: ['localhost:3000'],
        disallowedRoutes: ['localhost:3000/api/auth/login']
      }
    })
  ],
```

```
  providers: [
    {
      provide: HTTP_INTERCEPTORS,
      useClass: JwtInterceptor,
      multi: true
    }
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

This guide covers all the essential patterns for building a complete Angular application with routing, authentication, state management, and real-time features.