Angular + TypeScript Senior Developer Interview Prep Handbook

Core Angular Concepts

Q: What are Angular lifecycle hooks?
A: Angular provides hooks like ngOnInit, ngOnChanges, ngOnDestroy etc.

```
@Component({...})
export class MyComponent implements OnInit, OnDestroy {
  ngOnInit() { console.log('Init'); }
  ngOnDestroy() { console.log('Destroy'); }
}
```

Q: Explain Change Detection strategies.
A: Angular uses Default and OnPush. OnPush checks only when @Input values change or events fire.

Q: What is ViewEncapsulation?
A: It defines style scope: Emulated (default, scoped with attributes), ShadowDom (native shadow DOM), None (global styles).

Advanced Angular Topics

Q: How do you use Route Guards?
A: Guards like CanActivate, CanDeactivate, CanLoad control navigation.

```
@Injectable({providedIn:'root'})
class AuthGuard implements CanActivate {
  canActivate(): boolean { return !!localStorage.getItem('token'); }
}
```

Q: What are HTTP interceptors?
A: They allow modifying requests/responses. Example: add JWT token, handle errors globally.

Q: Explain Resolvers.
A: Resolvers fetch data before a route activates, ensuring components have data upfront.

State Management

Q: What are common state management approaches?
A: Options: NgRx, NGXS, Akita, or simple services with RxJS. Trade-offs: boilerplate vs simplicity.

Q: NgRx example:

```
interface AppState { user: User }
const selectUser = createSelector(...);
store.select(selectUser).subscribe(u => console.log(u));
```

RxJS & Async Programming

Q: Difference between switchMap, mergeMap, concatMap, exhaustMap?
A: - switchMap: cancels previous - mergeMap: concurrent - concatMap: sequential - exhaustMap: ignores new until current completes

Q: Explain Subjects.
A: Subject: multicast observable. BehaviorSubject: last value cached. ReplaySubject: replays multiple past values. AsyncSubject: emits last value on completion.

Q: Error handling in RxJS?
A: Use catchError, retry, finalize to gracefully recover.

Testing

Q: How do you test Angular components?
A: Use TestBed, mock services, detectChanges. For @Input/@Output test property bindings.

Q: Mock HTTP calls?
A: Use HttpTestingController to flush mock responses.

Q: E2E testing?
A: Protractor (deprecated), Cypress, or Playwright for real-browser scenarios.

Core TypeScript Knowledge

Q: Difference between interface and type?
A: Interfaces are extendable, types more flexible with unions/tuples.

Q: Explain any vs unknown vs never vs void.
A: - any: disables type checking - unknown: must be narrowed - never: function never returns - void: no return

Q: Explain structural typing.
A: TS uses 'duck typing': compatibility is based on shape, not explicit declarations.

Generics & Advanced Types

Q: Generics example:
A: Reusable function with type safety.

```
function identity<T>(arg: T): T { return arg; }
let num = identity<number>(10);
```

Q: Utility types?
A: Partial, Pick, Omit, Record simplify manipulation of object types.

Decorators & Angular Context

Q: How do TypeScript decorators work?
A: Decorators are functions applied to classes/properties. Angular uses them for DI metadata.

Q: Custom decorator example:
A: Log property access.

```
function Log(target: any, key: string) {
  let value = target[key];
  Object.defineProperty(target, key, {
    get: () => value,
    set: (v) => { console.log(`Set ${key} = ${v}`); value = v; }
  });
}
```

Real-World Scenarios

Q: Slow Angular app?
A: Use lazy loading, OnPush, trackBy in *ngFor, pure pipes, code-splitting, caching.

Q: Role-based access control?
A: Store roles in JWT, check via guards, hide/show UI with structural directives.

Q: Offline support?
A: Angular Service Worker, caching strategies with Workbox, IndexedDB for persistence.

Q: Migration from AngularJS?
A: Use ngUpgrade for hybrid, migrate gradually, refactor services/components.