Development guide

Code Style

TypeScript

- Use TypeScript strict mode
- Define interfaces for all data structures
- Use type annotations for function parameters and returns

Angular

- Follow Angular style guide
- Use Angular CLI for scaffolding
- One component per file
- Use services for business logic

Naming Conventions

- Components: `PascalCase` (e.g., `WaybillListComponent`)
- Services: `PascalCase` with `Service` suffix (e.g., `WaybillService`)
- Modules: `PascalCase` with `Module` suffix (e.g., `WaybillModule`)
- Files: `kebab-case` (e.g., `waybill-list.component.ts`)

Component Structure

```
import { Component, OnInit } from '@angular/core';


@Component({
  selector: 'app-component-name',
  templateUrl: './component-name.component.html',
  styleUrls: ['./component-name.component.css']
})
export class ComponentNameComponent implements OnInit {
  constructor(private service: SomeService) { }
```

```
  ngOnInit(): void {
    // Initialization logic
  }
}
```

Service Structure

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class SomeService {
  constructor(private http: HttpClient) { }

  getData(): Observable<any> {
    const headers = new HttpHeaders({
      'Authorization': localStorage.getItem('jwt')!
    });
    return this.http.get<any>(API_URL, { headers });
  }
}
```

Best Practices

1. **Component Communication**

- Use `@Input()` for parent-to-child data flow
- Use `@Output()` with EventEmitter for child-to-parent events
- Use services for complex state management

2. **Error Handling**

- Always handle Observable errors
- Provide user-friendly error messages
- Log errors for debugging

```
this.service.getData().subscribe(
  data => this.data = data,
  error => {
    console.error('Error:', error);
    // Show error message to user
  }
);
```

3. **Memory Management**

- Unsubscribe from Observables in `ngOnDestroy()`
- Use async pipe in templates when possible
- Avoid memory leaks

```
private subscription: Subscription;

ngOnInit() {
  this.subscription = this.service.getData().subscribe(/*...*/);
}

ngOnDestroy() {
  this.subscription?.unsubscribe();
}
```

4. **Performance**

- Use `trackBy` with `*ngFor` for large lists
- Implement `OnPush` change detection where appropriate
- Lazy load images and heavy components

5. **Security**

- Never store sensitive data in localStorage (use httpOnly cookies)
- Sanitize user input
- Validate all data from API
- Use HTTPS in production

Testing

Unit Tests

```
npm test
# Runs Karma test runner with Jasmine
```

E2E Tests

```
npm run e2e
# Runs Protractor for end-to-end tests
```

Writing Tests

```
describe('ComponentName', () => {
  let component: ComponentName;
  let fixture: ComponentFixture<ComponentName>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [ ComponentName ]
    }).compileComponents();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Linting

```
npm run lint
# Runs TSLint to check code quality
```

Fix linting issues automatically:

```
ng lint --fix
```

Git Workflow

1. Create feature branch: `git checkout -b feature/feature-name`
2. Make changes and commit: `git commit -m "feat: description"`

3. Push to remote: `git push origin feature/feature-name`
4. Create pull request for review

Commit Message Format:

- `feat:` New feature
- `fix:` Bug fix
- `docs:` Documentation changes
- `style:` Code style changes
- `refactor:` Code refactoring
- `test:` Test additions/changes
- `chore:` Build/tooling changes

Revision #1
Created 24 February 2026 07:15:54 by ondeliveloper
Updated 24 February 2026 07:16:33 by ondeliveloper