Лабораторна робота № 12. Оптимізація в Angular

Мета: навчитися працювати з модульною та standalone архітектурами, застосовувати @defer, CD та Signals для оптимізації роботи застосунку.

Завдання

1) Ознайомитися з матеріалом лекції № 12;
2) Виконати пункти 1-5 ходу роботи;
3) У звіті відобразити кроки 1-5 ходу роботи та додати скріни роботи застосунку (відображення, створення, оновлення, видалення, пошук, фільтрація, навігація);
4) Надати відповіді на контрольні питання.

Хід роботи

1) Першим етапом буде розділення головного модуля на наступні: фічеві за областю (inside/outside), core та shared.

Почнемо з core-модуля, для цього створимо теку app/core і модуль у ній за допомогою команди ***ng g m core***

```
svarog@svarog:~/projects/angular-course$ ng g m core
CREATE src/app/core/core.module.ts (190 bytes)
svarog@svarog:~/projects/angular-course$
```

Він має наступний вигляд

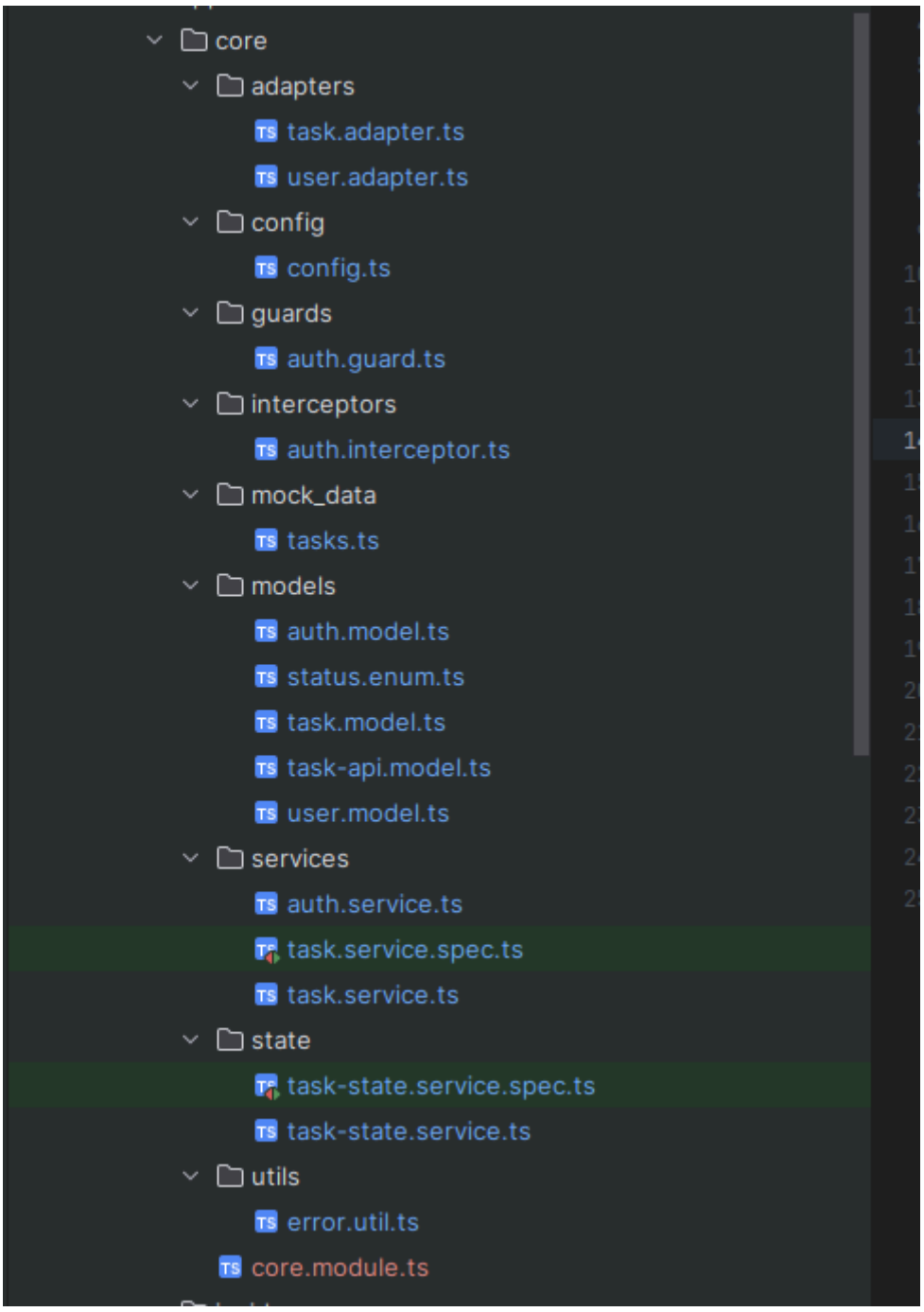
core.module.ts

```ts
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';



@NgModule({  no usages
  declarations: [],
  imports: [
    CommonModule
  ]
})
export class CoreModule { }

```

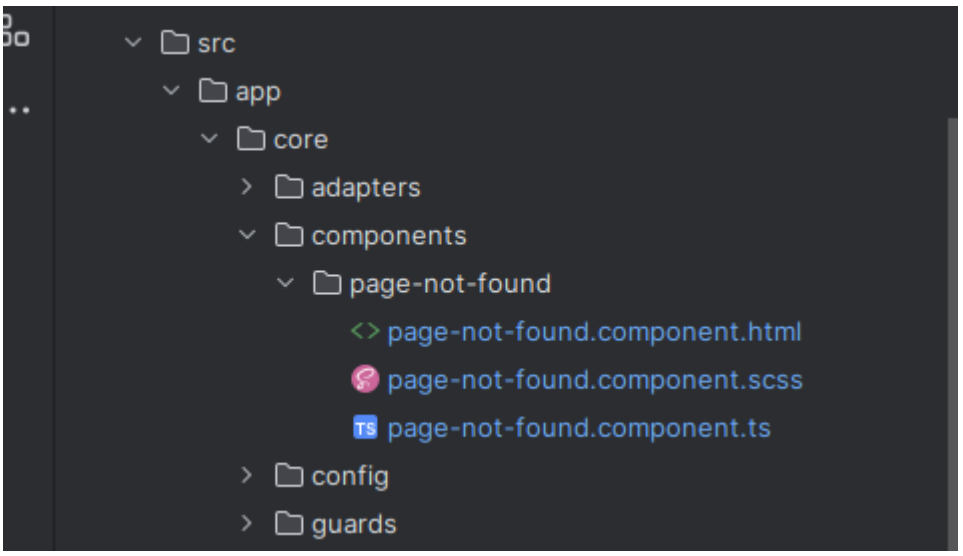
У цьому модулі зазвичай підключаються глобальні сервіси (якщо у них не вказано у декораторі providedIn: ‘root’), інтерсептори, guards, моделі, константи тощо.

Для початку потрібно буде перенести файли, що мають відношення до core-модуля у його теку, а саме: Singleton-сервіси, interceptors, guards, model, enum, adapters тощо.

Загалом структура буде мати наступний вигляд



Також переносимо компонент PageNotFoundComponent



Перейдемо до нашого модуля, так як більшість із того, що ми віднесли до теки core є моделями або Singleton-сервісами, то у модулі нам необхідно тільки прописати provideHttpClient та AuthInterceptor.

```
import { CommonModule } from '@angular/common';
import {HTTP_INTERCEPTORS, provideHttpClient, withInterceptorsFromDi} from '@angular/common/http';
import {AuthInterceptor} from './interceptors/auth.interceptor';
import {PageNotFoundComponent} from './components/page-not-found/page-not-found.component';



@NgModule({  Show usages
  declarations: [
    PageNotFoundComponent
  ],
  imports: [
    CommonModule
  ],
  exports: [
    PageNotFoundComponent
  ],
  providers: [
    provideHttpClient(withInterceptorsFromDi()),
    { provide: HTTP_INTERCEPTORS, useClass: AuthInterceptor, multi: true }
  ]
})
export class CoreModule {
```

Важливим нюансом є те, що CoreModule повинен бути підключеним лише у AppModule. Тому можна додатково захистити підключення модуля, щоб унеможливити використання у інших модулях окрім AppModule.

```
@NgModule({  no usages
  declarations: [],
  imports: [
    CommonModule
  ],
  providers: [
    provideHttpClient(withInterceptorsFromDi()),
    { provide: HTTP_INTERCEPTORS, useClass: AuthInterceptor, multi: true }
  ]
})
export class CoreModule {
  constructor(@Optional() @SkipSelf() parentModule: CoreModule) {  no usages
    if (parentModule) {
      throw new Error(
        'CoreModule is already loaded. Import it in the AppModule only.');
    }
  }
}
```

Підключаємо наш модуль до AppModule.

```
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    CoreModule,
```

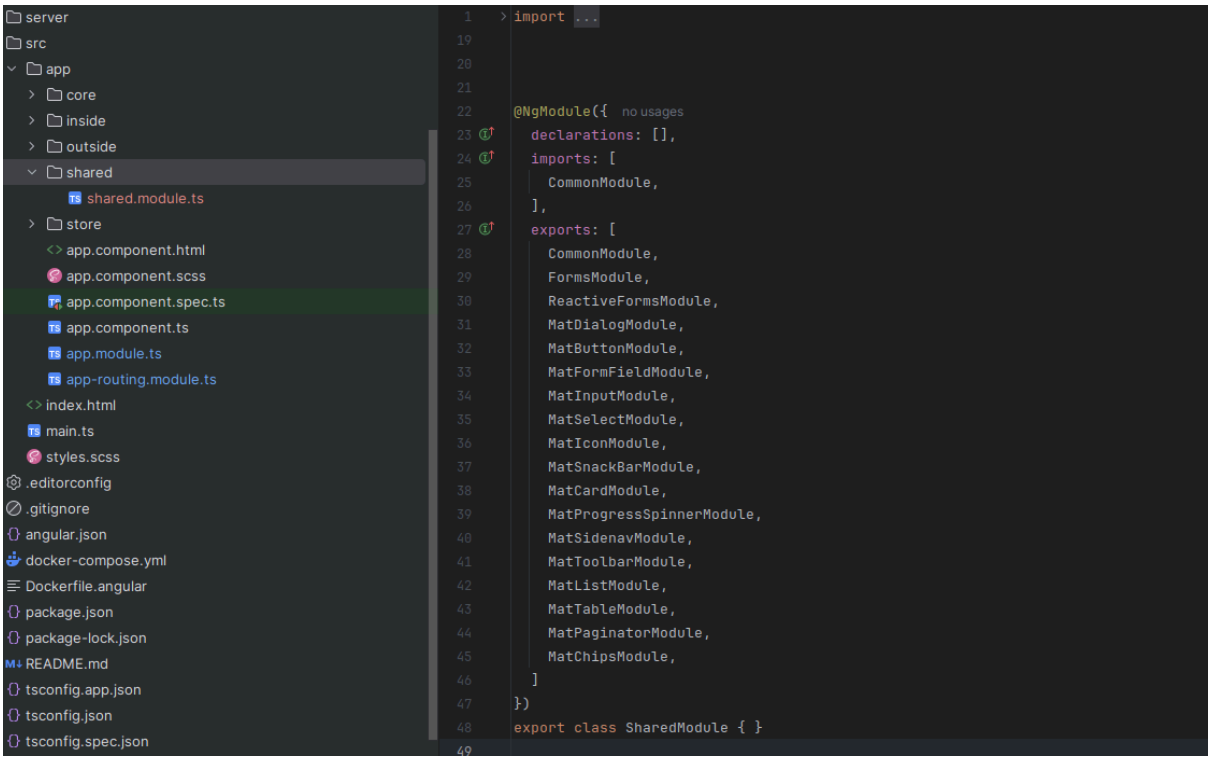
Не забуваємо очистити providers

```
    StoreDevtoolsModule.instrument({ maxAge: 25, logOnly: !isDevMode() }),
    StoreRouterConnectingModule.forRoot({
      stateKey: 'router'
    }),
  ],
  providers: [],
  bootstrap: [AppComponent]
```

Далі переходимо до SharedModule. Він зазвичай містить повторно використовуванні елементи (наприклад, Angular Material). Тобто те, що можуть використовувати декілька фічевих модулів. Сервіси до цієї категорії не входять.

Створимо такий модуль та підключимо необхідні імпорти (файли переносити у нашому випадку не потрібно, бо ті, що підходять (pipes та

validators) мають чітку прив’язку саме на взаємодію з компонентами завдань).



Цей модуль підключається у фічевих модулях (дуже рідко у головному, якщо маленький проєкт). У нашому випадку його також необхідно підключити у CoreModule, бо компонент використовує Angular Material.

```ts
import {HTTP_INTERCEPTORS, provideHttpClient, withInterceptorsFromDi} from '@angular/common/http';
import {AuthInterceptor} from './interceptors/auth.interceptor';
import {PageNotFoundComponent} from './components/page-not-found/page-not-found.component';
import {SharedModule} from '../shared/shared.module';



@NgModule({  Show usages
  declarations: [
    PageNotFoundComponent
  ],
  imports: [
    SharedModule
  ],
  exports: [
    PageNotFoundComponent
  ],
  providers: [
    provideHttpClient(withInterceptorsFromDi()),
    { provide: HTTP_INTERCEPTORS, useClass: AuthInterceptor, multi: true }
  ]
})
export class CoreModule {
  constructor(@Optional() @SkipSelf() parentModule: CoreModule) {  no usages
    if (parentModule) {
      throw new Error(
        'CoreModule is already loaded. Import it in the AppModule only.');
    }
  }
```

За аналогією робимо фічеві модулі.
Створюємо

```
svarog@svarog:~/projects/angular-course$ ng g m inside
CREATE src/app/inside/inside.module.ts (192 bytes)
svarog@svarog:~/projects/angular-course$ ng g m outside
CREATE src/app/outside/outside.module.ts (193 bytes)
svarog@svarog:~/projects/angular-course$
```

InsideModule

```
@NgModule({  no usages
  declarations: [
    // components
    WorkbenchLayoutComponent,
    TaskLayoutComponent,
    TaskListComponent,
    TaskItemComponent,
    TaskFormComponent,
    TaskStatsComponent,
    // pipes
    StatusFilterPipe,
    TaskStatusPipe,
  ],
  imports: [
    SharedModule,
  ]
})
export class InsideModule { }
```

OutsideModule

```
import { NgModule } from '@angular/core';
import {SharedModule} from '../shared/shared.module';
import {LoginComponent} from './components/auth/login/login.component';
import {RegisterComponent} from './components/auth/register/register.component';
import {ConfirmEmailComponent} from './components/auth/confirm-email/confirm-email.component';



@NgModule({  no usages
  declarations: [
    LoginComponent,
    RegisterComponent,
    ConfirmEmailComponent,
  ],
  imports: [
    SharedModule
  ]
})
export class OutsideModule { }
```

Підключаємо їх у головний модуль

```
import {CoreModule} from './core/core.module';
import {InsideModule} from './inside/inside.module';
import {OutsideModule} from './outside/outside.module';

@NgModule({  Show usages  Kryvyi Yaroslav
  declarations: [
    AppComponent,
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    CoreModule,
    InsideModule,
    OutsideModule,
    StoreModule.forRoot<AppState>({
      tasks: taskReducer,
      auth: authReducer,
      router: routerReducer
    }),
    EffectsModule.forRoot([TaskEffects, AuthEffects]),
    StoreDevtoolsModule.instrument({ maxAge: 25, logOnly: !isDevMode() }),
    StoreRouterConnectingModule.forRoot({
      stateKey: 'router'
    }),
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

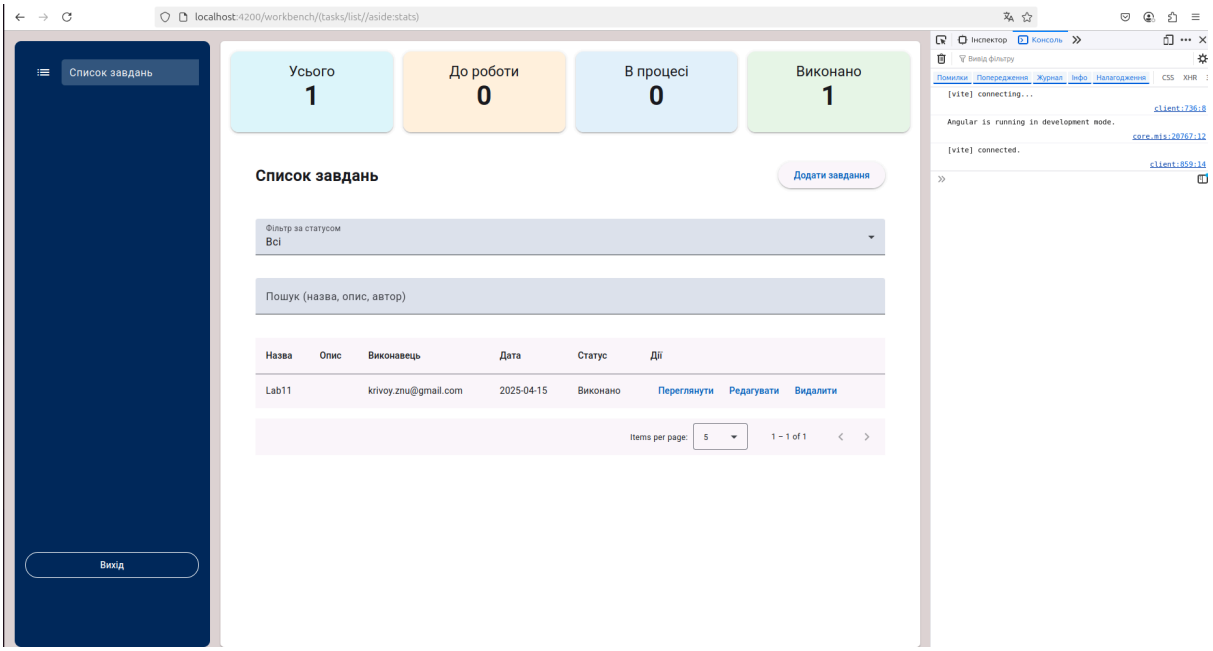
Також, так як наші компоненти використовують routerLink та router-outlet, то необхідно імпортувати RouterModule у фічеві модулі.

```
  ],
  imports: [
    SharedModule,
    RouterModule,
  ],
})
```

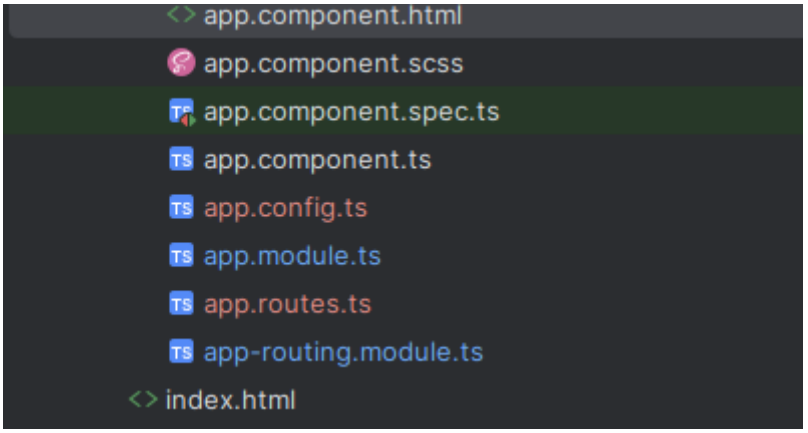
Що нам це дало?

1) Мо розділили логіку між спеціалізованими модулями;
2) Головний модуль тепер став менше та зрозуміліше;
3) Якщо буде необхідно підключити глобальний сервіс або спільний елемент, то зміни робляться у відповідному модулі та автоматично підтягнуться до інших;
4) Легко додавати фічеві модулі.

Перевіримо роботу програми



2) Далі переходимо на переведення компонентів на standalone. Для початку у app/ створимо app.config.ts та app.routes.ts.



Почнемо з **app.routes.ts**, у нього перенесемо всі наші роути.

```ts
import {AuthGuard} from './core/guards/auth.guard';
import {TaskLayoutComponent} from './inside/components/task-layout/task-layout.component';
import {TaskListComponent} from './inside/components/task-layout/task-list/task-list.component';
import {TaskItemComponent} from './inside/components/task-layout/task-item/task-item.component';
import {TaskStatsComponent} from './inside/components/task-layout/task-stats/task-stats.component';
import {LoginComponent} from './outside/components/auth/login/login.component';
import {RegisterComponent} from './outside/components/auth/register/register.component';
import {ConfirmEmailComponent} from './outside/components/auth/confirm-email/confirm-email.component';
import {PageNotFoundComponent} from './core/components/page-not-found/page-not-found.component';

export const routes: Routes = [  no usages
  {path: '', redirectTo: '/workbench/(tasks/list//aside:stats)', pathMatch: 'full'},
  {path: 'workbench', component: WorkbenchLayoutComponent, canActivate: [AuthGuard],
    children: [
      {path: 'tasks', component: TaskLayoutComponent,
        children: [
          {path: 'list', component: TaskListComponent},
          {path: 'view/:id', component: TaskItemComponent},
        ]
      },
      {path: 'stats', component: TaskStatsComponent, outlet: 'aside'},
    ]
  },
  {path: 'login', component: LoginComponent},
  {path: 'register', component: RegisterComponent},
  {path: 'confirm/:token',component: ConfirmEmailComponent},
  {path: '**', component: PageNotFoundComponent}
];
```

Поки залишаємо його так, потім до нього повернемося. Переходимо до app.component.ts - його потрібно зробити standalone та імпортувати у нього **RouterOutlet**.

```ts
import {Component} from '@angular/core';
import {RouterOutlet} from '@angular/router';

@Component({  Show usages  Kryvyi Yaroslav
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.scss',
  standalone: true,
  imports: [RouterOutlet]
})
export class AppComponent {
  title :string  = 'Task Planner';
}
```

Переходимо до app.config.ts, у якому підключаємо наші роути

```ts
import {ApplicationConfig} from '@angular/core';
import {provideRouter} from '@angular/router';
import { routes } from './app.routes';

export const appConfig: ApplicationConfig = {  Show usages
  providers: [
    provideRouter(routes),
  ]
}
```

Далі переходимо до **main.js**, у якому заміняємо bootstrapModule на bootstrapApplication та підключаємо кореневий компонент та конфіг.

```ts
import {bootstrapApplication} from '@angular/platform-browser';
import {AppComponent} from './app/app.component';
import {appConfig} from './app/app.config';

bootstrapApplication(AppComponent, appConfig).catch(error :any => console.log(error));  new *
```

Наступний кроком необхідно перевести на standalone всі наші пайпи та компоненти.

PageNotFoundComponent - використовує тільки mat-icon, mat-button та routerLink у своєму шаблоні, тому необхідно підключити тільки відповідні модулі.

```ts
import { Component } from '@angular/core';
import {MatIconModule} from '@angular/material/icon';
import {MatButtonModule} from '@angular/material/button';
import {RouterLink} from '@angular/router';

@Component({  Show usages  Kryvyi Yaroslav
  selector: 'app-page-not-found',
  templateUrl: './page-not-found.component.html',
  styleUrl: './page-not-found.component.scss',
  standalone: true,
  imports: [MatIconModule, MatButtonModule, RouterLink]
})
export class PageNotFoundComponent {
```

LoginComponent

```
@Component({  Show usages  Kryvyi Yaroslav
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrl: './login.component.scss',
  standalone: true,
  imports: [
    CommonModule,
    FormsModule,
    ReactiveFormsModule,
    MatCardModule,
    MatButtonModule,
    MatFormFieldModule,
    MatInputModule,
  ]
})
export class LoginComponent implements OnInit, OnDestroy {
```

RegisterComponent

```
@Component({  Show usages  Kryvyi Yaroslav
  selector: 'app-register',
  templateUrl: './register.component.html',
  styleUrl: './register.component.scss',
  standalone: true,
  imports: [
    CommonModule,
    FormsModule,
    ReactiveFormsModule,
    MatCardModule,
    MatButtonModule,
    MatFormFieldModule,
    MatInputModule,
  ]
})
export class RegisterComponent implements OnInit, OnDestroy {
```

ConfirmEmailComponent

```
@Component({  Show usages  Kryvyi Yaroslav
  selector: 'app-confirm-email',
  templateUrl: './confirm-email.component.html',
  styleUrl: './confirm-email.component.scss',
  standalone: true,
  imports: [CommonModule, MatCardModule]
})
export class ConfirmEmailComponent implements OnInit, OnDestroy {
```

WorkbenchLayoutComponent

```
@Component({  Show usages  Kryvyi Yaroslav
  selector: 'app-workbench-layout',
  templateUrl: './workbench-layout.component.html',
  styleUrl: './workbench-layout.component.scss',
  standalone: true,
  imports: [
    CommonModule,
    MatSidenavModule,
    MatListModule,
    MatIconModule,
    RouterModule,
    MatButtonModule
  ]
})
export class WorkbenchLayoutComponent implements OnInit{
```

TaskLayoutComponent

```
import { Component } from '@angular/core';
import {RouterOutlet} from '@angular/router';

@Component({  Show usages  Kryvyi Yaroslav
  selector: 'app-task-layout',
  templateUrl: './task-layout.component.html',
  styleUrl: './task-layout.component.scss',
  standalone: true,
  imports: [RouterOutlet]
})
export class TaskLayoutComponent {

}
```

TaskStatusPipe

```

@Pipe({  Show usages  Kryvyi Yaroslav
  name: 'taskStatus',
  standalone: true
})
export class TaskStatusPipe implements PipeTransform {

  transform(status: TaskStatus): string {  Show usages  Kryvyi Yaroslav
    const statusMap: { [key in TaskStatus]: string } = {
      [TaskStatus.TODO]: 'До роботи',
```

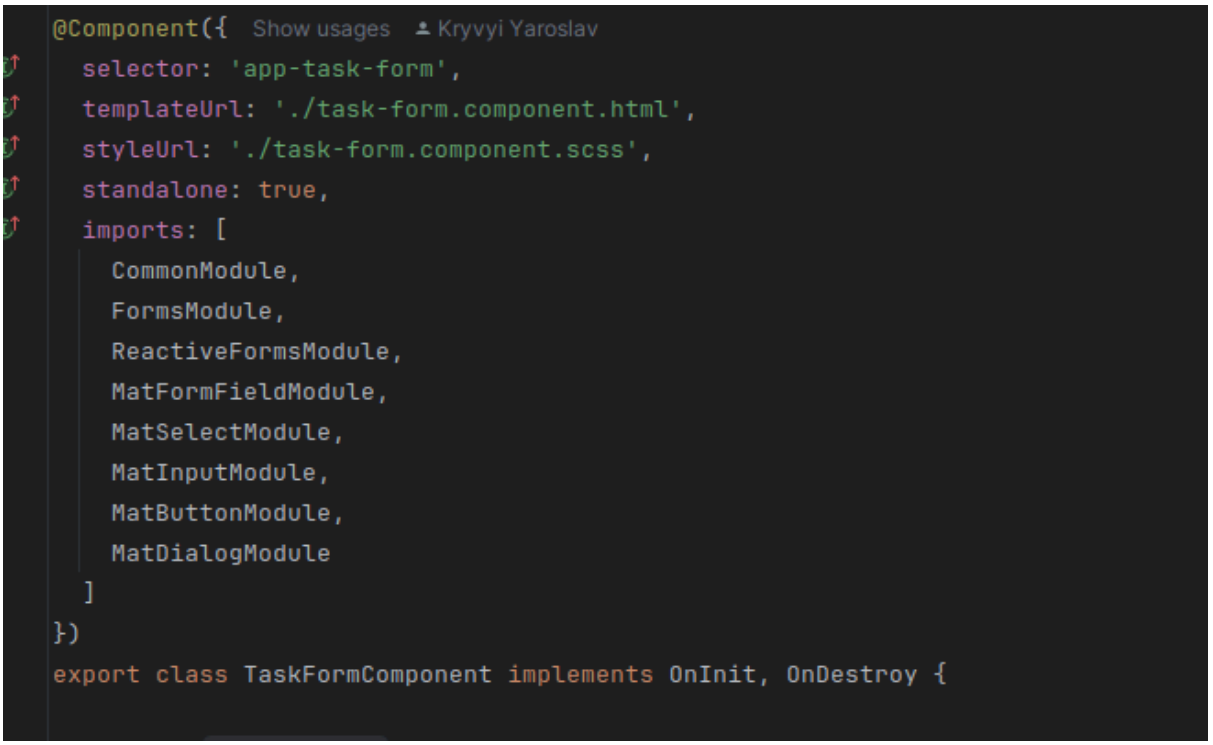
TaskListComponent

```
@Component({  Show usages  Kryvyi Yaroslav
  selector: 'app-task-list',
  templateUrl: './task-list.component.html',
  styleUrl: './task-list.component.scss',
  standalone: true,
  imports: [
    CommonModule,
    FormsModule,
    ReactiveFormsModule,
    MatTableModule,
    MatPaginatorModule,
    MatButtonModule,
    MatSelectModule,
    MatInputModule,
    MatProgressSpinnerModule,
    MatFormFieldModule,
    TaskStatusPipe
  ]
})
export class TaskListComponent implements OnInit, AfterViewInit, OnDestroy {
```

TaskItemComponent

```
@Component({  Show usages  Kryvyi Yaroslav
  selector: 'app-task-item',
  templateUrl: './task-item.component.html',
  styleUrl: './task-item.component.scss',
  standalone: true,
  imports: [
    CommonModule,
    MatChipsModule,
    MatFormFieldModule,
    MatSelectModule,
    TaskStatusPipe,
    MatIconModule,
    RouterModule,
    MatButtonModule
  ]
})
export class TaskItemComponent implements OnInit, OnDestroy {
```

TaskFormComponent



TaskStatsComponent

Повертаємося до app.routes.ts. Ц ньому замінимо всі component на loadComponent у роутах.

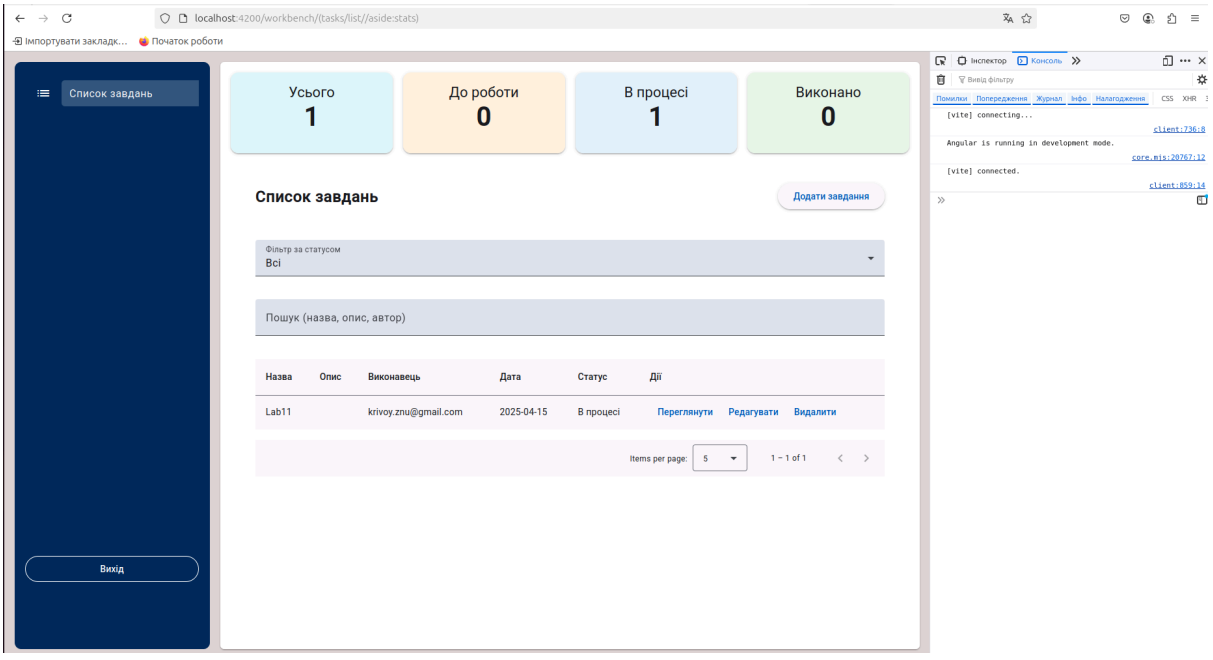
```
import {Routes} from '@angular/router';
import {AuthGuard} from './core/guards/auth.guard';

export const routes: Routes = [
```

```
  {path: '', redirectTo: '/workbench/(tasks/list//aside:stats)', pathMatch: 'full'},
  {
        path: 'workbench',
        loadComponent: () => import('./inside/components/workbench-layout/workbench-layout.component')
        .then(c => c.WorkbenchLayoutComponent),
        canActivate: [AuthGuard],
        children: [
        {
        path: 'tasks',
        loadComponent: () => import('./inside/components/task-layout/task-layout.component')
        .then(c => c.TaskLayoutComponent),
        children: [
        {
        path: 'list',
        loadComponent: () => import('./inside/components/task-layout/task-list/task-list.component')
        .then(c => c.TaskListComponent),
        },
        {
        path: 'view/:id',
        loadComponent: () => import('./inside/components/task-layout/task-item/task-item.component')
        .then(c => c.TaskItemComponent),
        },
        ]
        },
        {
        path: 'stats',
        loadComponent: () => import('./inside/components/task-layout/task-stats/task-stats.component')
        .then(c => c.TaskStatsComponent),
        outlet: 'aside'
        },
        ]
  },
  {
        path: 'login',
        loadComponent: () => import('./outside/components/auth/login/login.component')
        .then(c => c.LoginComponent),
  },
  {
        path: 'register',
        loadComponent: () => import('./outside/components/auth/register/register.component')
        .then(c => c.RegisterComponent),
  },
  {
        path: 'confirm/:token',
        loadComponent: () => import('./outside/components/auth/confirm-email/confirm-email.component')
        .then(c => c.ConfirmEmailComponent),
  },
  {
        path: '**',
        loadComponent: () => import('./core/components/page-not-found/page-not-found.component')
        .then(c => c.PageNotFoundComponent),
  }
];
```

Тепер знов переходимо до app.config.ts. У ньому нам необхідно підключити глобальних провайдерів (наприклад, MatSnackBarModule), Store тощо.

```ts
import {MatDialogModule} from '@angular/material/dialog';

export const appConfig: ApplicationConfig = {  Show usages
  providers: [
    provideRouter(routes),
    provideHttpClient(withInterceptorsFromDi()),
    provideStore({
      tasks: taskReducer,
      auth: authReducer,
      router: routerReducer,
    }),
    provideEffects([TaskEffects, AuthEffects]),
    provideStoreDevtools({
      maxAge: 25,
      logOnly: !isDevMode()
    }),
    provideRouterStore(),
    importProvidersFrom([MatSnackBarModule, MatDialogModule]),
    { provide: HTTP_INTERCEPTORS, useClass: AuthInterceptor, multi: true } // бо клас, а не функція
  ]
}
```

Аналогічно перевіримо роботу



3) Щодо використання @defer, то конкретно у нашому випадку він не має сенсу, бо всі компоненти ми підтягуємо через router-outlet + loadComponent, які вже реалізують схожу логіку. Але якщо би у нас

був компонент, який ми явно би викликали у шаблоні іншого компонента, то його краще обгортати у @defer.

4) Наступний етапом, переведемо всі компоненти на стратегію ChangeDetectionStrategy.OnPush. Для відслідковування запуску CD, використаємо хук ngDoCheck.

У декораторі @Component дадаємо changeDetection: ChangeDetectionStrategy.OnPush (ChangeDetectionStrategy з @angular/core).

```
@Component({  Show usages  Kryvyi Yaroslav *
  selector: 'app-task-list',
  templateUrl: './task-list.component.html',
  styleUrl: './task-list.component.scss',
  standalone: true,
  imports: [
    CommonModule,
    FormsModule,
    ReactiveFormsModule,
    MatTableModule,
    MatPaginatorModule,
    MatButtonModule,
    MatSelectModule,
    MatInputModule,
    MatProgressSpinnerModule,
    MatFormFieldModule,
    TaskStatusPipe
  ],
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class TaskListComponent implements OnInit, AfterViewInit, OnDestroy, DoCheck {
```

Також реалізуємо інтерфейс DoCheck та його метод ngDoCheck() з наступним вмістом.

```
ngDoCheck(): void {  no usages  new *
  console.log('[TaskListComponent] CD triggered');
}
```

За аналогією робимо інші компоненти.

Далі необхідно вручну прописати тригери оновлення для деяких частин компонентів, а саме до тих, що відносяться до локального стану

компонента (прапор завантаження, @VievChild тощо) і при цьому використовуються у шаблоні без async.

Для тригеру потрібно інжектувати ChangeDetectorRef у конструктор

```
82        private router: Router,
83        private cdr: ChangeDetectorRef,
84        ) {
85      }
```

Почнемо з task-list.component.ts. У ньому нас цікавить пагінатор з @ViewChild та змінна hasLoading (так як ми не напряму звертаємося у шаблоні до змінної loading$ у шаблоні через async pipe, то отримаємо нескінченне завантаження). Пропишемо ручний тригер CD.

```
  this.loading$.pipe(takeUntil(this.destroy$)).subscribe((loading) => {
    this.hasLoading = loading;
    this.cdr.markForCheck()
  })
}
```

У методах deleteTask та goToLastPage після this.paginator.pageIndex.

```
deleteTask(id: string): void {  Show usages  Kryvyi Yaroslav *
  this.store.dispatch(TaskActions.deleteTask({id}));
  this.tasks$.pipe(take(1)).subscribe(tasks : Task[]  => {
    const isLastItemOnPage : boolean  = tasks.length === 1;
    const currentPage : number  = this.paginator.pageIndex + 1;

    const goToPage : number  = isLastItemOnPage && currentPage > 1
      ? currentPage - 1
      : currentPage;

    this.loadTasks(
      goToPage,
      this.paginator.pageSize,
      this.filterControl.value ?? '',
      this.statusControl.value ?? ''
    );

    this.paginator.pageIndex = goToPage - 1;
    this.cdr.markForCheck();
  });
}
```

```
private goToLastPage(): void {  Show usages  Kryvyi Yaroslav *
  this.totalTasks$.pipe(take(1)).subscribe(total : number  => {
    const lastPage : number  = Math.ceil((total + 1) / this.paginator.pageSize);
    this.paginator.pageIndex = lastPage - 1;
    this.cdr.markForCheck();
```

У task-item.component.ts логіка ідеально підходить під OnPush, тому його пропускаємо. Трохи цікавіше ситуація з task-form.component.ts, у тому вигляді, як він зараз працює - проблем не буде, але краще для можливих слабких місць прописати тригер, а саме у this.selectedTask$, бо йде запис у форму та оновлення значення isEdit.

```
this.selectedTask$.pipe(takeUntil(this.destroy$)).subscribe((task : Task | null ) : void  => {
  if (task) {
    this.taskForm.patchValue(task);
    this.editMode = true;
  } else {
    this.taskForm.reset({status: TaskStatus.TODO});
    this.editMode = false;
  }
  this.cdr.markForCheck();
})
}
```

У task-stats.component.ts всі змінні у шаблоні виводяться через async pipe, тому додаткового тригеру не потрібно. Та же ситуація і у WorkbenchLayoutComponent, TaskLayoutComponent та PageNotFoundComponent. Схожа ситуація і у компонентів реєстрації та входу до системи, так як всі їх змінні використовують async або викликають snackBar, у них можна було би прописати ручний тригер, якщо, наприклад, після сабміту ми очищали форму.

А ось ConfirmEmailComponent містить локальну змінну message, яку ми виводимо у шаблоні, тому тут потрібен тригер.

```
ngOnInit(): void {  no usages  Kryvyi Yaroslav *
  this.error$ = this.store.select(AuthSelectors.selectAuthError);
  const token : string | null  = this.route.snapshot.paramMap.get('token');

  if (token) {
    this.store.dispatch(AuthActions.confirmEmail({ token }));
  }

  this.store.select(AuthSelectors.selectConfirmEmailSuccess).pipe(takeUntil(this.destroy$))
    .subscribe(success : boolean  => {
      this.message = success ? 'Пошта підтверджена' : 'Очікуємо підтвердження...';
      this.cdr.markForCheck();
    });

}
```

5) Останнім етапом, переведемо деякі з елементів до яких ми прописали тригер оновлення у Signals, що дозволить уникнути зайвий раз виклику markForCheck та переведе змінні у реактивний формат.

ConfirmEmailComponent

Було

```
export class ConfirmEmailComponent implements OnInit, OnDestroy, DoCheck {

  error$!: Observable<string | null>;
  message!: string;

  private destroy$ : Subject<void> = new Subject<void>();

  constructor( no usages  new *
    private route: ActivatedRoute,
    private store: Store<AppState>,
    private cdr: ChangeDetectorRef
  ) {}

  ngOnInit(): void {  no usages  Kryvyi Yaroslav *
    this.error$ = this.store.select(AuthSelectors.selectAuthError);
    const token : string | null = this.route.snapshot.paramMap.get('token');

    if (token) {
      this.store.dispatch(AuthActions.confirmEmail({ token }));
    }

    this.store.select(AuthSelectors.selectConfirmEmailSuccess).pipe(takeUntil(this.destroy$))
      .subscribe(success : boolean => {
        this.message = success ? 'Пошта підтверджена' : 'Очікуємо підтвердження...';
        this.cdr.markForCheck();
      });
```

Стало

```
  error$!: Observable<string | null>;
  message!: WritableSignal<string>;

  private destroy$ : Subject<void> = new Subject<void>();

  constructor( no usages  new *
    private route: ActivatedRoute,
    private store: Store<AppState>,
    private cdr: ChangeDetectorRef
  ) {}

  ngOnInit(): void {  no usages  Kryvyi Yaroslav *
    this.error$ = this.store.select(AuthSelectors.selectAuthError);
    const token : string | null = this.route.snapshot.paramMap.get('token');

    if (token) {
      this.store.dispatch(AuthActions.confirmEmail({ token }));
    }

    this.store.select(AuthSelectors.selectConfirmEmailSuccess).pipe(takeUntil(this.destroy$))
      .subscribe(success : boolean => {
        this.message.set(success ? 'Пошта підтверджена' : 'Очікуємо підтвердження...');
      });
```

І у шаблоні

confirm-email.component.ts | task-form.validator.ts | confirm-email.component.html | register.component.html | logir

```
<div class="confirm-container">
  <mat-card class="confirm-card">
    <mat-card-title>Підтвердження пошти</mat-card-title>
    @if (error$ | async; as error) {
      <mat-card-content>
        <p class="error-text">
          {{ error }}
        </p>
      </mat-card-content>
    } @else {
      <mat-card-content>
        <p>
          {{ message() }}
        </p>
      </mat-card-content>
    }
  </mat-card>
</div>

```

TaskFormComponent

Було

```
this.selectedTask$.pipe(takeUntil(this.destroy$)).subscribe((task : Task | null ) : void  => {
  if (task) {
    this.taskForm.patchValue(task);
    this.editMode = true;
  } else {
    this.taskForm.reset({status: TaskStatus.TODO});
    this.editMode = false;
  }
  this.cdr.markForCheck();
 })
}
```

Стало

```
export class TaskFormComponent implements OnInit, OnDestroy, DoCheck {

 destroy$ = new Subject<void>();
 selectedTask$!: Observable<Task | null>;
 taskForm!: FormGroup;
 editMode: WritableSignal<boolean> = signal(false);
```

```
constructor(
        private store: Store<AppState>,
        private fb: FormBuilder,
        private cdr: ChangeDetectorRef,
        public dialogRef: MatDialogRef<TaskFormComponent>,
) {
}

ngOnInit(): void {
        this.selectedTask$ = this.store.select(TaskSelectors.selectSelectedTask);

        this.taskForm = this.fb.group({
        id: [''],
        title: ['', Validators.required],
        description: ['', TaskFormValidator.forbiddenWordsValidator(['React', 'Vue'])],
        dueDate: ['',  [Validators.required, TaskFormValidator.dateValidator]],
        assignee: [''],
        status: [TaskStatus.TODO, Validators.required]
        });

        this.selectedTask$.pipe(takeUntil(this.destroy$)).subscribe((task) => {
        if (task) {
        this.taskForm.patchValue(task);
        this.editMode.set(true);
        } else {
        this.taskForm.reset({status: TaskStatus.TODO});
        this.editMode.set(false);
        }
        })
}

ngDoCheck(): void {
        console.log('[TaskFormComponent] CD triggered');
}

onSubmit(): void {
        if (this.taskForm.valid) {
        if (this.editMode()) {
        this.store.dispatch(TaskActions.updateTask({task: {...this.taskForm.value}}));
        } else {
        this.store.dispatch(TaskActions.createTask({task: {...this.taskForm.value}}));
        }
        this.store.select(TaskSelectors.selectTaskLoading)
        .pipe(
        filter(isLoading => !isLoading),
        take(1)
        )
        .subscribe(() => {
        this.store.select(TaskSelectors.selectTaskError)
        .pipe(take(1))
        .subscribe(error => {
        if (!error) {
        this.dialogRef.close(this.editMode() ? 'updated' : 'created');
        this.store.dispatch(TaskActions.selectTask({ id: null }));
        }
        });
        });
```

```
        }
    }

    ngOnDestroy() {
        this.store.dispatch(TaskActions.selectTask({ id: null }));
        this.destroy$.next();
        this.destroy$.complete();
    }

    protected readonly TaskStatus = TaskStatus;
}
```

```
export class ConfirmEmailComponent implements OnInit, OnDestroy, DoCheck {

  error$!: Observable<string | null>;
  message!: WritableSignal<string>;

  private destroy$ : Subject<void>  = new Subject<void>();

  constructor(  no usages  new *
    private route: ActivatedRoute,
    private store: Store<AppState>,
    private cdr: ChangeDetectorRef
  ) {}

  ngOnInit(): void {  no usages  Kryvyi Yaroslav *
    this.error$ = this.store.select(AuthSelectors.selectAuthError);
    const token : string | null  = this.route.snapshot.paramMap.get('token');

    if (token) {
      this.store.dispatch(AuthActions.confirmEmail({ token }));
    }

    this.store.select(AuthSelectors.selectConfirmEmailSuccess).pipe(takeUntil(this.destroy$))
      .subscribe(success : boolean  => {
        this.message.set(success ? 'Пошта підтверджена' : 'Очікуємо підтвердження...');
        this.cdr.markForCheck();
      });
```

І у шаблоні

```
  <div class="form-actions">
    <button mat-raised-button color="primary" type="submit" [disabled]="!taskForm.valid" (click)="onSubm
      {{ editMode() ? 'Оновити завдання' : 'Додати нове завдання'}}
    </button>
    <button mat-button mat-dialog-close>Скасувати</button>
  </div>
</form>
```

TaskListComponent

Було

```
hasLoading: boolean = false;

  filterControl = new FormControl('');
  statusControl = new FormControl('');

  constructor(
            private store: Store<AppState>,
            public dialog: MatDialog,
            private snackBar: MatSnackBar,
            private router: Router,
            private cdr: ChangeDetectorRef,
            ) {
  }

  ngOnInit(): void {
            this.loadTasks(1, 5);
            this.tasks$ = this.store.select(TaskSelectors.selectAllTasks);
            this.loading$ = this.store.select(TaskSelectors.selectTaskLoading);
            this.error$ = this.store.select(TaskSelectors.selectTaskError);
            this.totalTasks$ = this.store.select(TaskSelectors.selectTaskTotal);

            this.filterControl.valueChanges.pipe(
            debounceTime(300),
            distinctUntilChanged()
            ).subscribe(() => {
            this.paginator.firstPage();
            });

            this.statusControl.valueChanges.subscribe(() => {
            this.paginator.firstPage();
            });


            this.error$.pipe(takeUntil(this.destroy$)).subscribe(error => {
            if (error) {
            this.snackBar.open(error, 'Закрити', { duration: 4000, panelClass: ['error-snackbar'] });
            }
            });

            this.loading$.pipe(takeUntil(this.destroy$)).subscribe((loading) => {
            this.hasLoading = loading;
            this.cdr.markForCheck()
            })
```

Стало

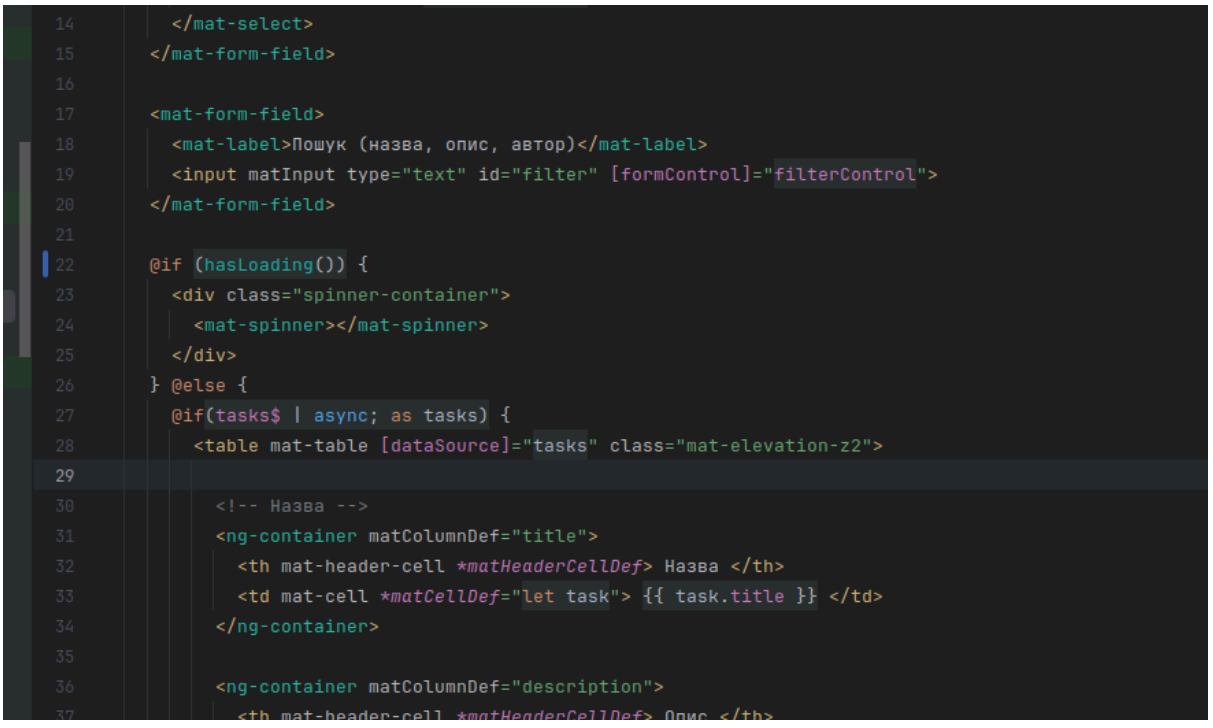
```
export class TaskListComponent implements OnInit, AfterViewInit, OnDestroy, DoCheck {

  displayedColumns: string[] = ['title', 'description',  'assignee', 'dueDate', 'status', 'actions'];
```

```
  @ViewChild(MatPaginator, { static: false }) paginator!: MatPaginator;

  destroy$ = new Subject<void>();
  loading$!: Observable<boolean>;
  error$!: Observable<string | null>;
  totalTasks$!: Observable<number>;

  tasks$!: Observable<Task[]>;

  hasLoading: WritableSignal<boolean> = signal(false);

  filterControl = new FormControl('');
  statusControl = new FormControl('');

  constructor(
            private store: Store<AppState>,
            public dialog: MatDialog,
            private snackBar: MatSnackBar,
            private router: Router,
            private cdr: ChangeDetectorRef,
            ) {
  }

  ngOnInit(): void {
            this.loadTasks(1, 5);
            this.tasks$ = this.store.select(TaskSelectors.selectAllTasks);
            this.loading$ = this.store.select(TaskSelectors.selectTaskLoading);
            this.error$ = this.store.select(TaskSelectors.selectTaskError);
            this.totalTasks$ = this.store.select(TaskSelectors.selectTaskTotal);

            this.filterControl.valueChanges.pipe(
            debounceTime(300),
            distinctUntilChanged()
            ).subscribe(() => {
            this.paginator.firstPage();
            });

            this.statusControl.valueChanges.subscribe(() => {
            this.paginator.firstPage();
            });


            this.error$.pipe(takeUntil(this.destroy$)).subscribe(error => {
            if (error) {
            this.snackBar.open(error, 'Закрити', { duration: 4000, panelClass: ['error-snackbar'] });
            }
            });

            this.loading$.pipe(takeUntil(this.destroy$)).subscribe((loading) => {
            this.hasLoading.set(loading);
            })
  }

  ngAfterViewInit() {
            this.paginator.page.pipe(
            startWith({ pageIndex: 0, pageSize: 5 }),
            switchMap(({ pageIndex, pageSize }) =>
            combineLatest([
```

```
            this.filterControl.valueChanges.pipe(startWith(this.filterControl.value)),
            this.statusControl.valueChanges.pipe(startWith(this.statusControl.value))
            ]).pipe(
            takeUntil(this.destroy$),
            tap(([filter, status]) => {
            this.loadTasks(pageIndex + 1, pageSize, filter ?? '', status ?? '');
            })
            )
            )
            ).subscribe();
  }

  ngDoCheck(): void {
            console.log('[TaskListComponent] CD triggered');
  }


  openDialog() : void {
            const dialogRef = this.dialog.open(TaskFormComponent, {
            height: '70vh',
            width: '80vw',
            });

            dialogRef.afterClosed().pipe(
            filter(result => result === 'created'),
            ).subscribe(() => {
            this.goToLastPage();
            });
  }

  viewTask(id: string): void {
            this.router.navigate([{ outlets: { primary: ['workbench', 'tasks', 'view', id], aside: null } }]);
  }

  editTask(id: string): void {
            this.store.dispatch(TaskActions.selectTask({id}));
            this.openDialog();
  }

  deleteTask(id: string): void {
            this.store.dispatch(TaskActions.deleteTask({id}));
            this.tasks$.pipe(take(1)).subscribe(tasks => {
            const isLastItemOnPage = tasks.length === 1;
            const currentPage = this.paginator.pageIndex + 1;

            const goToPage = isLastItemOnPage && currentPage > 1
            ? currentPage - 1
            : currentPage;

            this.loadTasks(
            goToPage,
            this.paginator.pageSize,
            this.filterControl.value ?? '',
            this.statusControl.value ?? ''
            );

            this.paginator.pageIndex = goToPage - 1;
            this.cdr.markForCheck();
```

```
        });
  }

  ngOnDestroy() {
        this.destroy$.next();
        this.destroy$.complete();
  }

  protected readonly TaskStatus = TaskStatus;

  private loadTasks(page: number, pageSize: number, filter = '', status = ''): void {
        this.store.dispatch(TaskActions.loadTasks({ page, pageSize, filter, status }));
  }

  private goToLastPage(): void {
        this.totalTasks$.pipe(take(1)).subscribe(total => {
        const lastPage = Math.ceil((total + 1) / this.paginator.pageSize);
        this.paginator.pageIndex = lastPage - 1;
        this.cdr.markForCheck();

        this.loadTasks(
        lastPage,
        this.paginator.pageSize,
        this.filterControl.value ?? '',
        this.statusControl.value ?? ''
        );
        });
  }

}
```

І у шаблоні



Контрольні питання

1) Що таке Signals? Для чого вони використовуються?
2) У чому відмінність standalone-компоненту від звичайного (той який оголошується у модулі)?
3) Чи потрібно імпортувати напряму у standalone-компонент модуль, якщо він використовується тільки як інжекція?
4) На які основні модулі ділиться застосунок при модульній архітектурі?
5) Які умови використання @defer? Чи потрібно його використовувати, якщо компонент підвантажується через <router-outlet> + loadComponent()?
6) У яких випадках прописується ручний тригер CD?