Лабораторна робота № 10. Маршрутизація в Angular

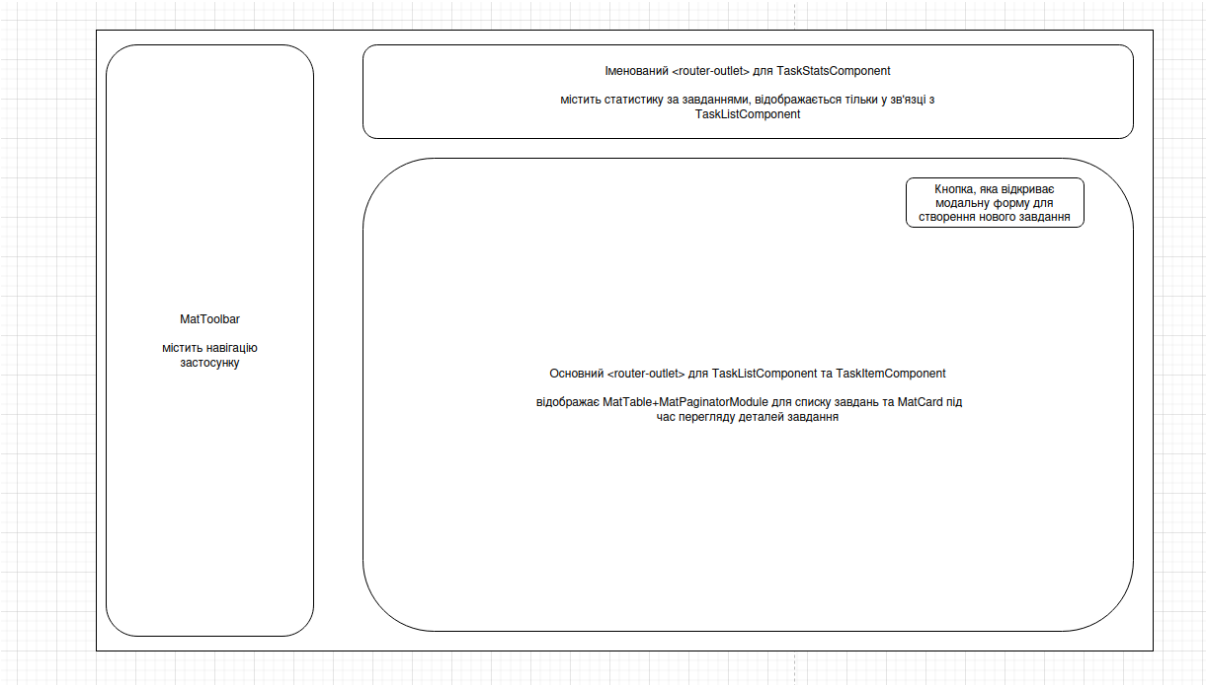
Мета: навчитися створювати і обробляти маршрути та їх параметри.

Завдання

1) Ознайомитися з матеріалом лекції № 10;
2) Виконати пункти 1-7 ходу роботи;
3) У звіті відобразити кроки 1-7 ходу роботи та додати скріни роботи застосунку (відображення, створення, оновлення, видалення, пошук, фільтрація, навігація);
4) Надати відповіді на контрольні питання.

Хід роботи

1) Для початку визначимо вид нашого застосунку. Так як ми вводимо маршрутизацію, то необхідно внести зміни у структуру. Надамо цю структуру схематично



Для позиціонування елементів на сторінці будемо використовувати flex, а для створення необхідної структури - Material. Імпортуємо необхідні модулі у app.module.ts.

```
60      MatCardModule,
61      MatProgressSpinnerModule,
62      MatSidenavModule,
63      MatToolbarModule,
64      MatListModule,
65      StoreModule.forRoot<AppState>({tasks: taskReducer, router: routerReducer}),
66      EffectsModule.forRoot([TaskEffects]),
```

Перед тим, як створити структуру, нам необхідно вирішити одну задачу, а саме: “Як взаємодіяти з маршрутами та їх параметрами?”. Є традиційний спосіб через ActivatedRoute, а є @ngrx/router-store. Саме через нього ми і будемо працювати з нашими роутами. Тому для початку встановимо його за допомогою команди:

ng add @ngrx/router-store

```
svarog@svarog:~/projects/angular-course$ ng add @ngrx/router-store
✔ Determining Package Manager
  › Using package manager: npm
✔ Searching for compatible package version
  › Found compatible package version: @ngrx/router-store@19.1.0.
✔ Loading package information from registry
✔ Confirming installation
✔ Installing package
UPDATE src/app/app.module.ts (2647 bytes)
UPDATE package.json (1242 bytes)
✔ Packages installed successfully.
```

Далі у теці нашого Store створимо теку router з файлами **router.state.ts** та **router.selectors.ts**

У **router.state.ts** прописуємо інтерфейс

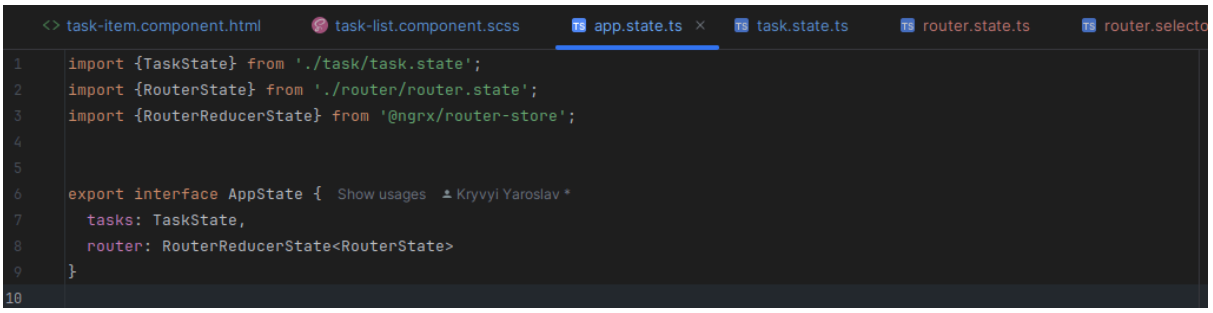
```ts
import {Data, Params} from '@angular/router';

export interface RouterState {  Show usages
  url: string;
  params: Params;
  queryParams: Params;
  fragment: string | null;
  data: Data;
}
```

У **router.selectors.ts** експортуємо вбудовані selectors

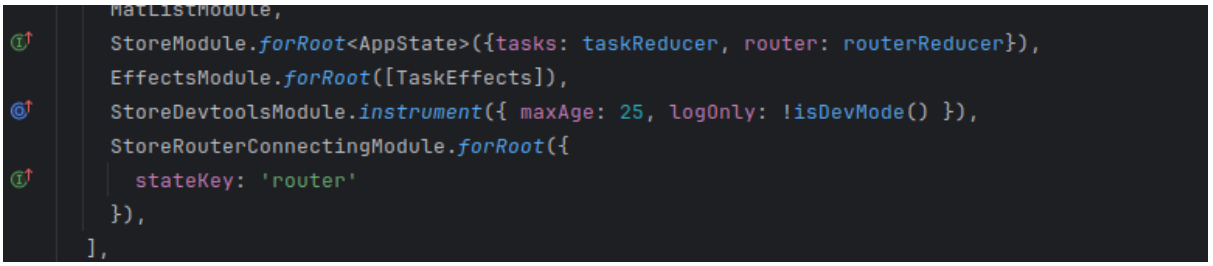
task-item.component.html | task-list.component.scss | app.state.ts | task.state.ts | router.state.ts | router.selectors.ts

```ts
import {createFeatureSelector} from '@ngrx/store';
import {RouterReducerState, getRouterSelectors} from '@ngrx/router-store';
import {RouterState} from './router.state';

export const selectRouter : MemoizedSelector<object, RouterReducerSta... =  Show usages
  createFeatureSelector<RouterReducerState<RouterState>>('router');

export const {  no usages
  selectCurrentRoute, // select the current route
  selectFragment, // select the current route fragment
  selectQueryParams, // select the current route query params
  selectQueryParam, // factory function to select a query param
  selectRouteParams, // select the current route params
  selectRouteParam, // factory function to select a route param
  selectRouteData, // select the current route data
  selectRouteDataParam, // factory function to select a route data param
  selectUrl, // select the current url
  selectTitle, // select the title if available
} = getRouterSelectors(selectRouter);
```

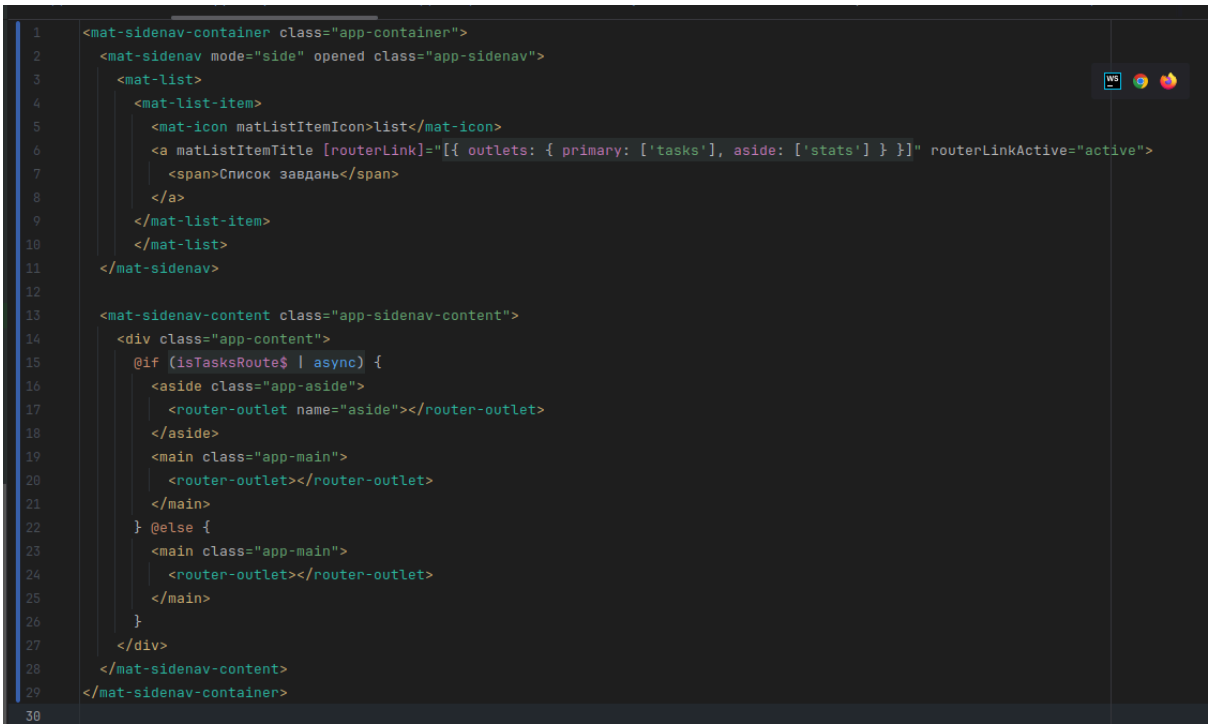
У **app.state.ts** доповнюємо інтерфейс



У **app.module.ts** підключаємо reducers та модуль у imports



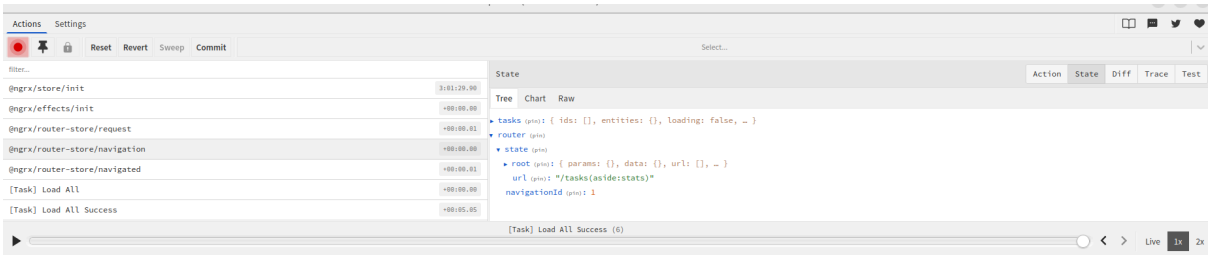
2) Переходимо до базової структури. У **app.component.html** створюємо необхідну структуру



Роути у нас поки не створено, але відразу можемо у [routerLink] вказати, що будемо відкривати іменований outlet разом із основним. Також відразу пропишемо логіку, у якому випадку і нас буде відображатися іменований outlet, а у якому - ні. Для цього використаємо змінну isTasksRoute$, яка буде за допомогою selectUrl з route.selectors.ts отримувати поточний URL та перевіряти його. реалізуємо логіку у **app.component.ts**. Оператор filter необхідний для відкидання undefined та null, бо на момент ініціалізації компонента router-store ще не ініціалізовано, у map перевіряємо, що роут починається на /tasks

```
constructor(private store: Store<AppState>) {}  no usages  new *

ngOnInit(): void {  no usages  new *
  this.isTasksRoute$ = this.store.select(selectUrl).pipe(
    filter(url : string  => !!url),
    map(url : string  => url.startsWith('/tasks(aside:stats)')),
  );
}
```

Також можемо перевірити стан router у DevTools



Стилі для **app.component.scss**

```
$color_1: white;
$background-color_1: #dfd6d6;
$background-color_2: rgba(255, 255, 255, 0.1);
$background-color_3: rgba(255, 255, 255, 0.2);
$background-color_4: #fff;

.app-container {
  width: 100vw;
  height: 100vh;
  display: flex;
  flex-direction: row;
  background-color: $background-color_1;
  .mat-toolbar {
        background: #004a9f;
        color: $color_1;
        padding-left: 32px;
```

```
  }
  .app-sidenav {
                margin: 16px;
                width: 300px;
                background: #002b5c;
                color: $color_1;
                border-radius: 10px;
                padding: 16px;
                text-align: center;
                mat-list {
                display: flex;
                flex-direction: column;
                gap: 10px;
                }
                mat-list-item {
                padding: 0;
                font-size: 20px;
                a {
                display: flex;
                align-items: center;
                gap: 12px;
                color: $color_1;
                text-decoration: none;
                width: 100%;
                padding: 8px 12px;
                border-radius: 5px;
                transition: background 0.3s ease;
                &:hover {
                background-color: $background-color_2;
                }
                span {
                font-size: 1rem;
                }
                }
                a.active {
                border: none;
                background-color: $background-color_3;
                }
                mat-icon {
                height: 24px;
                width: 24px;
                color: $color_1;
                }
                }
  }
  .app-sidenav-content {
                flex-grow: 1;
                .app-content {
                display: flex;
                flex-direction: column;
                gap: 16px;
                margin: 16px 16px 16px 32px;
                padding-bottom: 10px;
                flex-grow: 1;
                border-radius: 6px;
                box-shadow: 0 2px 1px -1px rgba(0, 0, 0, 0.2), 0 1px 1px 0 rgba(0, 0, 0, 0.14), 0 1px 3px 0 rgba(0, 0, 0, 0.12);
                overflow: hidden;
                background-color: $background-color_4;
                min-height: calc(100vh - 42px);
                }
  }
}
```

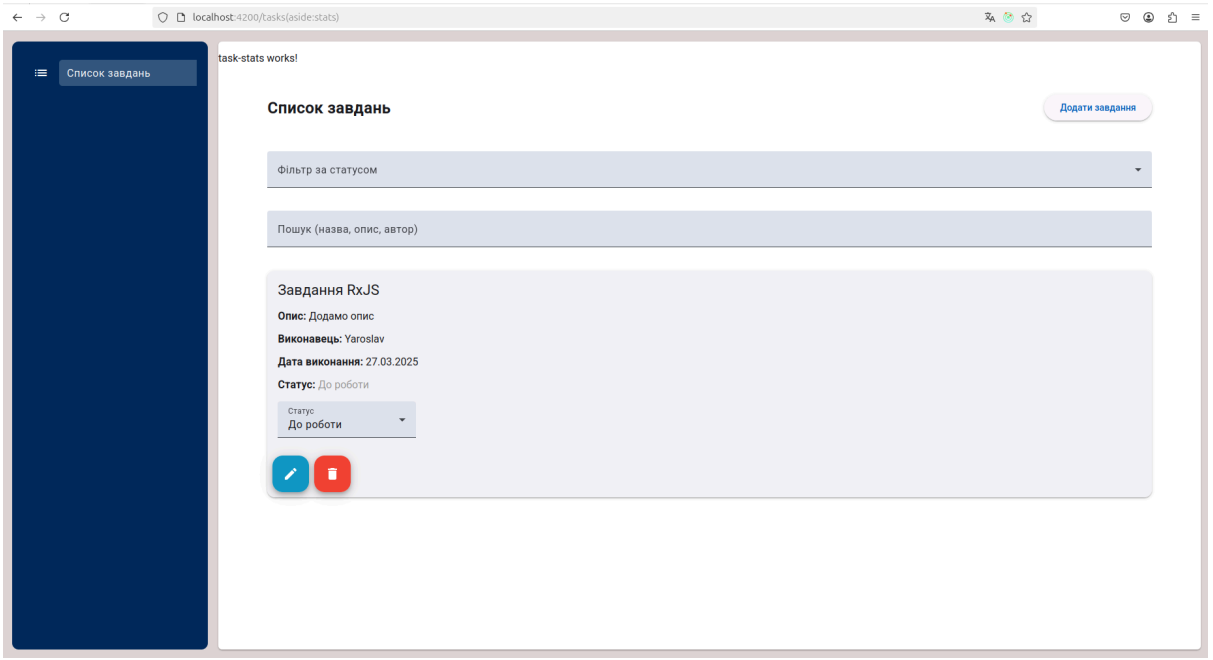
3) Наступним кроком пропишемо роути у app-routing.module.ts (якщо немає у проєкті, то **ng generate module app-routing --flat --module=app**)

```
const routes: Routes = [
  { path: '', redirectTo: '/(primary:tasks//aside:stats)', pathMatch: 'full' },
  {path: 'tasks', component: TaskListComponent},
  {path: 'stats', component: TaskStatsComponent, outlet: 'aside'},
  {path: 'tasks/view/:id', component: TaskItemComponent},
  {path: '**', component: PageNotFoundComponent}
];
```

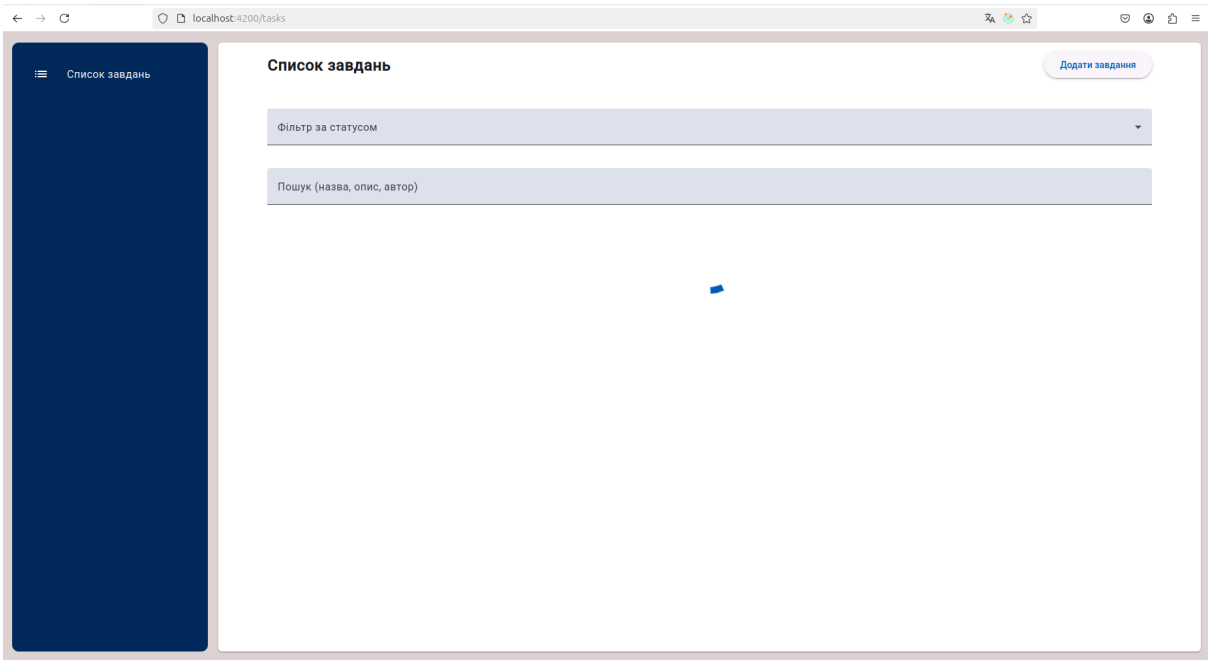
Також створюємо та імпортуємо у поточний модуль нові компоненти

```
svarog@svarog:~/projects/angular-course$ ng g c components/task-stats --skip-tests
CREATE src/app/components/task-stats/task-stats.component.scss (0 bytes)
CREATE src/app/components/task-stats/task-stats.component.html (25 bytes)
CREATE src/app/components/task-stats/task-stats.component.ts (236 bytes)
UPDATE src/app/app.module.ts (2954 bytes)
svarog@svarog:~/projects/angular-course$ ng g c components/page-not-found --skip-tests
CREATE src/app/components/page-not-found/page-not-found.component.scss (0 bytes)
CREATE src/app/components/page-not-found/page-not-found.component.html (29 bytes)
CREATE src/app/components/page-not-found/page-not-found.component.ts (251 bytes)
UPDATE src/app/app.module.ts (3012 bytes)
svarog@svarog:~/projects/angular-course$
```

Перевіримо результат роботи застосунку



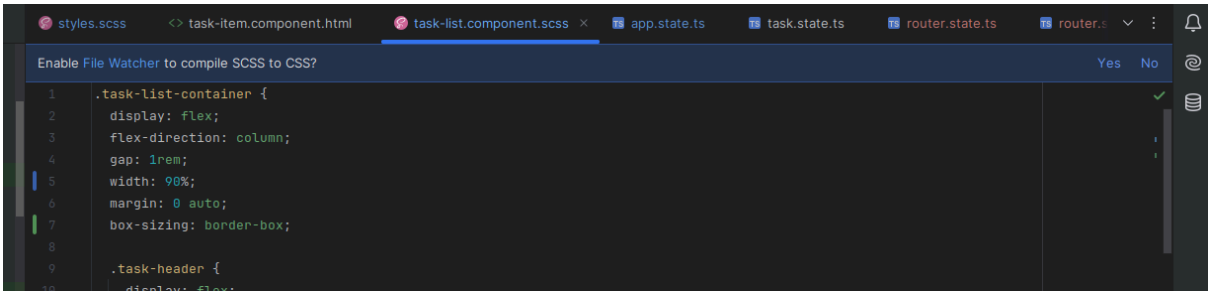
Перейдемо просто за роутом tasks



Як бачимо, логіка відображення іменованого outlet працює

4) Переходимо до зміни task-list.component

Для початку трохи оновимо стилі, щоб відображення на попередніх рисунках відповідало дійсності



Змінюємо значення ширини на відсоткове.
Далі необхідно підключити MatTableModule та MatPaginatorModule

Оновлюємо шаблон task-list.component.html

```
<section class="task-list-container">
 <div class="task-header">
            <h2>Список завдань</h2>
            <button mat-raised-button (click)="openDialog()">Додати завдання</button>
 </div>

 <mat-form-field>
            <mat-label>Фільтр за статусом</mat-label>
            <mat-select (selectionChange)="onSelected($event)" [formControl]="statusControl">
            <mat-option [value]="">Всі</mat-option>
            <mat-option [value]="TaskStatus.TODO">До роботи</mat-option>
            <mat-option [value]="TaskStatus.IN_PROGRESS">В процесі</mat-option>
            <mat-option [value]="TaskStatus.DONE">Виконано</mat-option>
            </mat-select>
 </mat-form-field>

 <mat-form-field>
            <mat-label>Пошук (назва, опис, автор)</mat-label>
            <input matInput type="text" id="filter" [formControl]="filterControl">
 </mat-form-field>

 @if (hasLoading) {
            <div class="spinner-container">
            <mat-spinner></mat-spinner>
            </div>
 } @else {
            <table mat-table [dataSource]="dataSource" class="mat-elevation-z2">

            <!-- Назва -->
            <ng-container matColumnDef="title">
            <th mat-header-cell *matHeaderCellDef> Назва </th>
            <td mat-cell *matCellDef="let task"> {{ task.title }} </td>
```

```
            </ng-container>

            <ng-container matColumnDef="description">
            <th mat-header-cell *matHeaderCellDef> Опис </th>
            <td mat-cell *matCellDef="let task"> {{ task.description }} </td>
            </ng-container>

            <!-- Виконавець -->
            <ng-container matColumnDef="assignee">
            <th mat-header-cell *matHeaderCellDef> Виконавець </th>
            <td mat-cell *matCellDef="let task"> {{ task.assignee }} </td>
            </ng-container>

            <!-- Дата -->
            <ng-container matColumnDef="dueDate">
            <th mat-header-cell *matHeaderCellDef> Дата </th>
            <td mat-cell *matCellDef="let task"> {{ task.dueDate }} </td>
            </ng-container>

            <!-- Статус -->
            <ng-container matColumnDef="status">
            <th mat-header-cell *matHeaderCellDef> Статус </th>
            <td mat-cell *matCellDef="let task"> {{ task.status | taskStatus}} </td>
            </ng-container>

            <!-- Дії -->
            <ng-container matColumnDef="actions">
            <th mat-header-cell *matHeaderCellDef> Дії </th>
            <td mat-cell *matCellDef="let task">
            <button mat-button (click)="viewTask(task.id)">Переглянути</button>
            <button mat-button (click)="editTask(task.id)">Редагувати</button>
            <button mat-button color="warn" (click)="deleteTask(task.id)">Видалити</button>
            </td>
            </ng-container>

            <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
            <tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>
            </table>
 }
  <mat-paginator #paginator [pageSize]="5" [pageSizeOptions]="[1, 5, 10, 25]" [length]="totalTasks$ | async"></mat-paginator>
</section>
```

Оновлюємо task-list.component.ts

Вказуємо колонки, які будуть відображатися у шаблоні, а також створюємо змінну, яка буде відповідати за взаємодію з таблицею (дані, пагінація тощо). Підключаємо декоратор @ViewChild для взаємодії з MatPaginator. У конструкторі додаємо інжект Router.

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

  displayedColumns: string[] = ['title', 'description',  'assignee', 'dueDate', 'status', 'actions'];
  dataSource = new MatTableDataSource<Task>();

  @ViewChild(MatPaginator) paginator!: MatPaginator;

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

  hasLoading: boolean = false;

  filterControl = new FormControl('');

  constructor(  no usages  Kryvyi Yaroslav *
    private store: Store<AppState>,
    public dialog: MatDialog,
    private snackBar: MatSnackBar,
    private router: Router,
    ) {
  }

  ngOnInit(): void {  no usages  Kryvyi Yaroslav *
    this.store.dispatch(TaskActions.loadTasks({}));
```

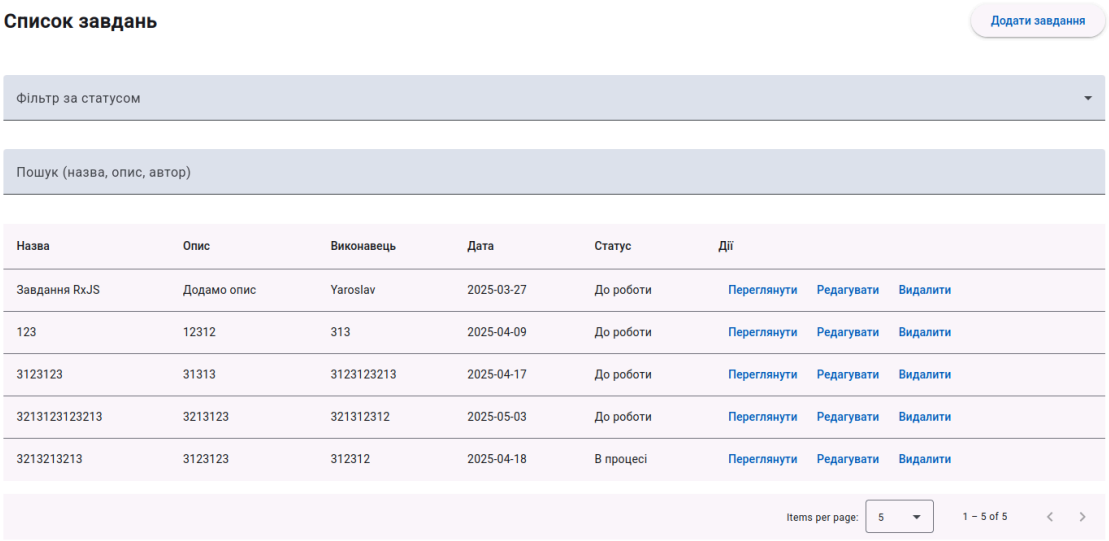
Оновимо логіку у ngOnInit

```
    this.totalTasks$ = this.store.select(TaskSelectors.selectTaskTotal);

    combineLatest([
      this.store.select(TaskSelectors.selectFilteredTasks),
      this.filterControl.valueChanges.pipe(
        startWith(''),
        debounceTime(300),
        distinctUntilChanged()
      )
    ]).pipe(
      map(([tasks, filter] : [Task[], string | null] ) : Task[] =>
          tasks.filter(task : Task =>
            task.title.toLowerCase().includes(filter ?? ''.toLowerCase()) ||
            task.description?.toLowerCase().includes(filter ?? ''.toLowerCase()) ||
            task.assignee.toLowerCase().includes(filter ?? ''.toLowerCase())
          )
      ),
      takeUntil(this.destroy$)
    ).subscribe( tasks : Task[] => this.dataSource.data = tasks);
```

Так як task-item відтепер буде відповідати за відображення інформації про компонент та виключно зміну статусу, то керування завданням тепер лежить на task-list. Реалізуємо методи

```
  }

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

  editTask(id: string): void {  Show usages  Kryvyi Yaroslav *
    this.store.dispatch(TaskActions.selectTask({id}));
    this.openDialog();
  }

  deleteTask(id: string): void {  Show usages  new *
    this.store.dispatch(TaskActions.deleteTask({id}));
  }
```

Перевіривши роботу, можна побачити, що все працює



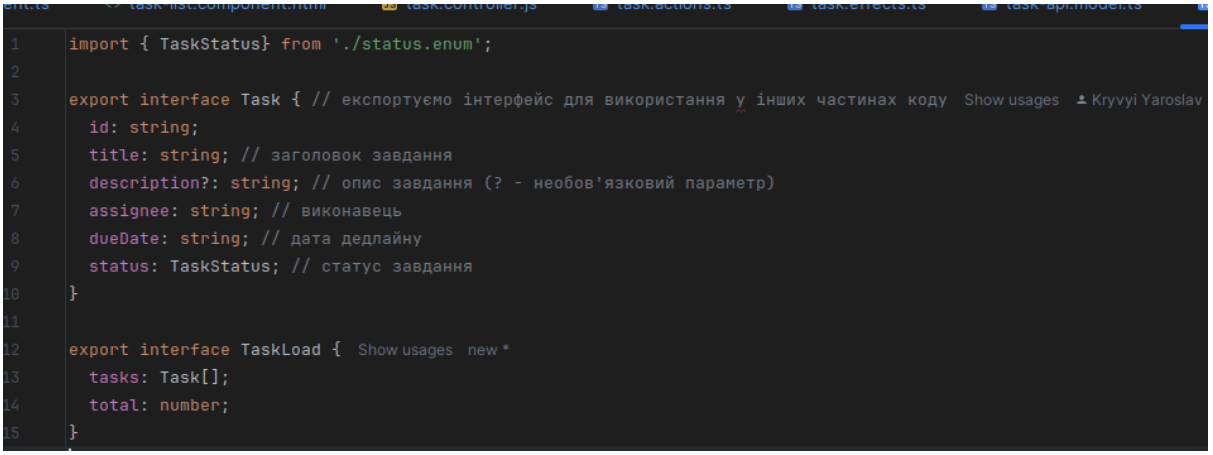
Але тепер є декілька нюансів:

1) Із-за використання combineLatest не працює логіка MatPaginator;
2) Повністю поламане відображення у task-item із-за того, що він більше не отримує завдання.

Вирішимо для початку першу задачу.

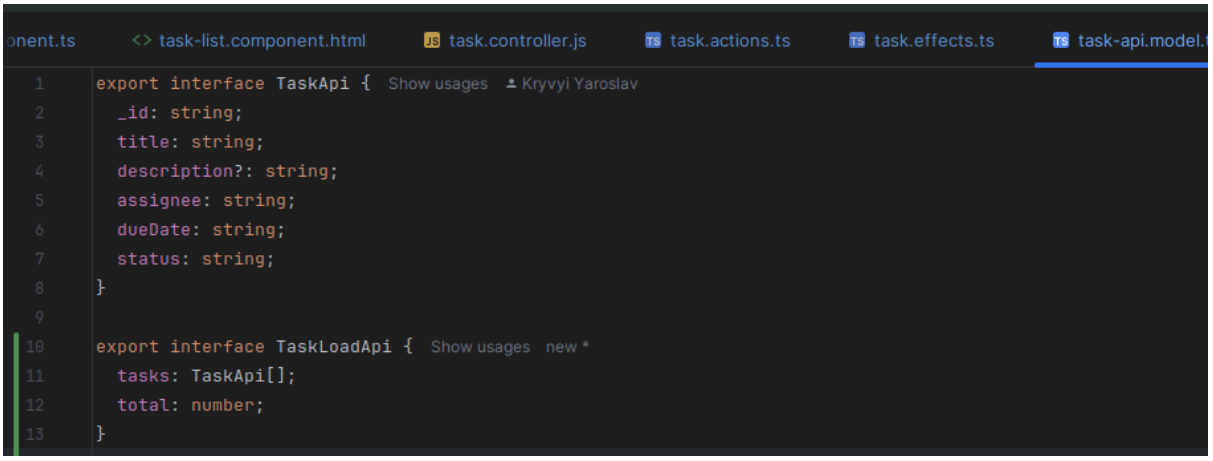
Є багато різних способів, як це можна зробити, ми підемо найскладнішим (для нашого поточного проєкту), а саме будемо використовувати серверну пагінацію + Store.

Для початку створимо декілька нових інтерфейсів, які нам знадобляться для прийому даних від GET-запиту за новою структурою, а саме



```ts
import { TaskStatus} from './status.enum';

export interface Task { // експортуємо інтерфейс для використання у інших частинах коду  Show usages  Kryvyi Yaroslav
  id: string;
  title: string; // заголовок завдання
  description?: string; // опис завдання (? - необов'язковий параметр)
  assignee: string; // виконавець
  dueDate: string; // дата дедлайну
  status: TaskStatus; // статус завдання
}

export interface TaskLoad {  Show usages  new *
  tasks: Task[];
  total: number;
}
```

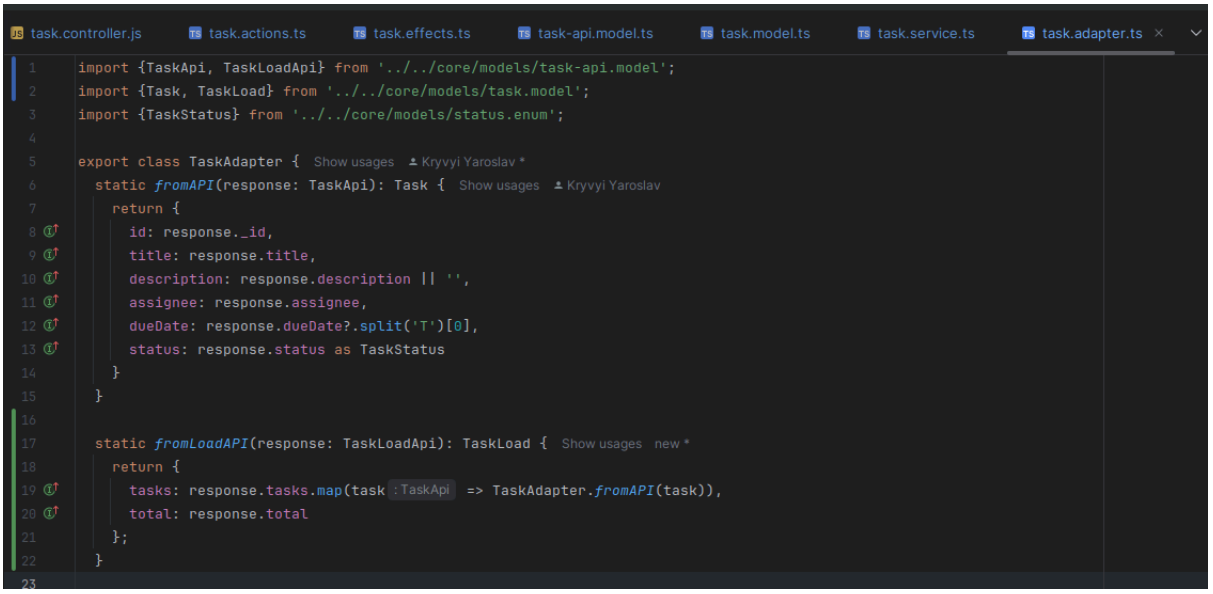
У нас додається властивість total, яка буде містити загальну кількість завдань (через selector store не підходить, бо при пагінації/фільтрації/пошуку ми будемо отримувати кількість за цими параметрами). Так як ми використовуємо адаптери, то необхідно оновити і їх, а також додати новий інтерфейс для роботи з серверними даними.



onent.ts | task-list.component.html | task.controller.js | task.actions.ts | task.effects.ts | task-api.model.

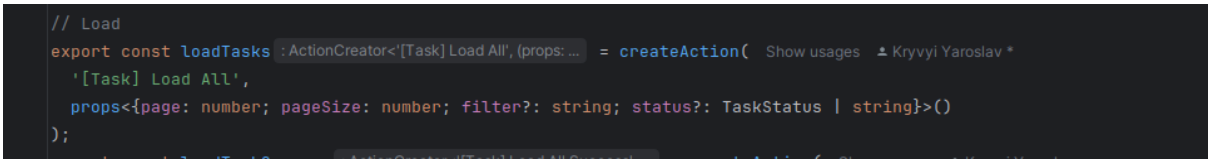
```ts
export interface TaskApi {  Show usages  Kryvyi Yaroslav
  _id: string;
  title: string;
  description?: string;
  assignee: string;
  dueDate: string;
  status: string;
}

export interface TaskLoadApi {  Show usages  new *
  tasks: TaskApi[];
  total: number;
}
```

І створюємо новий адаптер



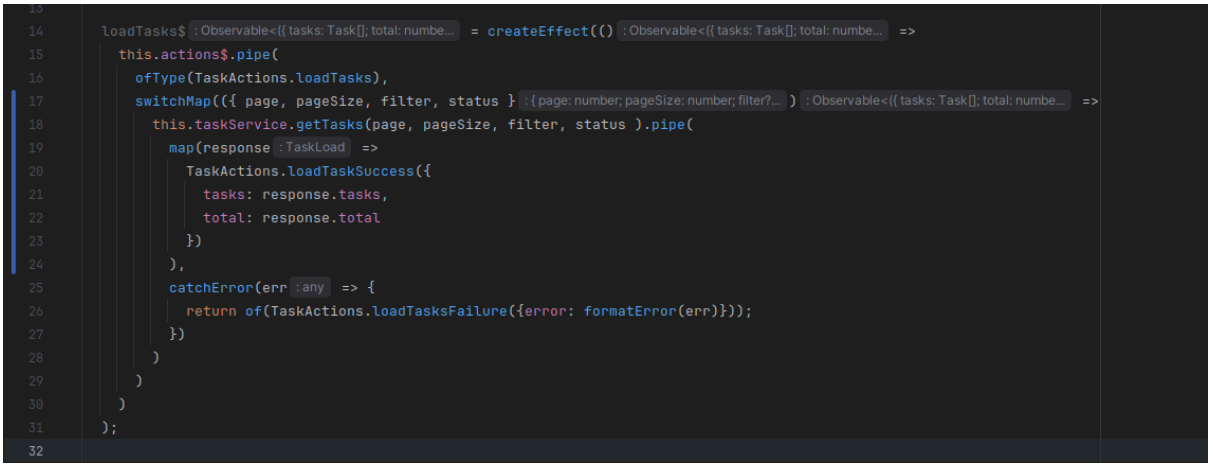
Далі оновлюємо **task.state.ts**

Оновимо Actions loadTasks у **task.actions.ts**



Ми додали до payload номер сторінки, кількість елементів на сторінку, фільтр та статус.

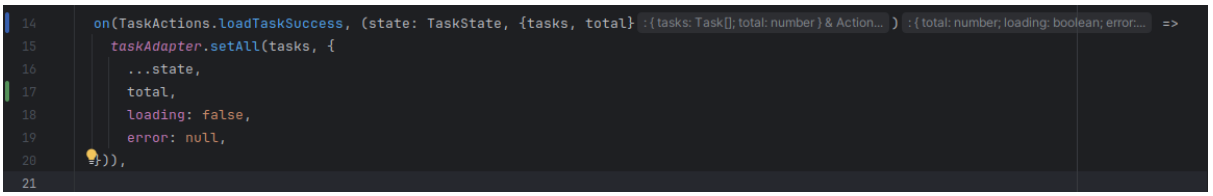
Далі оновлюємо **task.effects.ts**



```
loadTasks$ : Observable<({ tasks: Task[]; total: numbe... = createEffect(() : Observable<({ tasks: Task[]; total: numbe... =>
  this.actions$.pipe(
    ofType(TaskActions.loadTasks),
    switchMap(({ page, pageSize, filter, status } : { page: number; pageSize: number; filter?... ) : Observable<({ tasks: Task[]; total: numbe... =>
      this.taskService.getTasks(page, pageSize, filter, status ).pipe(
        map(response : TaskLoad =>
          TaskActions.loadTaskSuccess({
            tasks: response.tasks,
            total: response.total
          })
        ),
        catchError(err : any => {
          return of(TaskActions.loadTasksFailure({error: formatError(err)}));
        })
      )
    )
  )
);
```

Тепер він приймає додаткові параметри та записує у payload завдання та їх кількість.

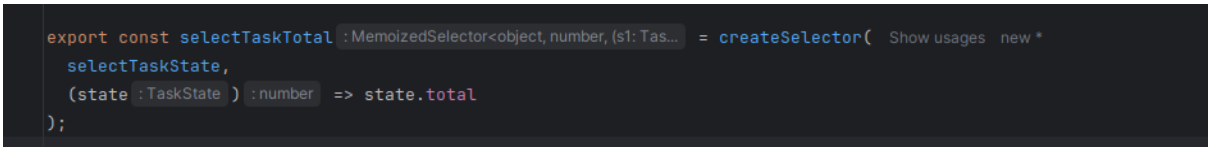
Оновлюємо **task.reducer.ts**



```
on(TaskActions.loadTaskSuccess, (state: TaskState, {tasks, total} : { tasks: Task[]; total: number } & Action... ) : { total: number; loading: boolean; error:... =>
  taskAdapter.setAll(tasks, {
    ...state,
    total,
    loading: false,
    error: null,
  })),
```

У ньому ми приймаємо оновлений payload з Actions та записуємо у стан total.

Також необхідно оновити selectors, який відповідає за кількість завдань, зараз він бере дані з entity. Переробимо, щоб брав властивість total.



```
export const selectTaskTotal : MemoizedSelector<object, number, (s1: Tas... = createSelector( Show usages  new *
  selectTaskState,
  (state : TaskState ) : number => state.total
);
```

Далі оновлюємо **task.service.ts**

```
getTasks(page: number, pageSize: number, filter?: string, status?: string): Observable<TaskLoad> {  Show usages  Kryvyi Yaroslav *
  let params :HttpParams  = new HttpParams()
    .set('page', page)
    .set('limit', pageSize);

  if (filter) params = params.set('filter', filter);
  if (status) params = params.set('status', status);

  return this.http.get<TaskLoadApi>(`${this.config.apiUrl}/v2/tasks`, {params: params}).pipe(
    map(TaskAdapter.fromLoadAPI), delay(1000)
  );
}
```

Тепер він приймає TaskLoadApi та повертає TaskLoad. Також використовуємо новий адаптер.

У **task-list.component.ts** створюємо приватну функцію loadTasks

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

Змінюємо виклик в ngOnInit

```
ngOnInit(): void {  no usages  Kryvyi Yaroslav *
  this.loadTasks(1, 5);

  this.loading$ = this.store.select(TaskSelectors.selectTaskLoading);
  this.error$ = this.store.select(TaskSelectors.selectTaskError);
```

Перед тим, як перейти до серверної обробки, необхідно трохи змінити підхід до фільтрації та вибору статусу.

Зробимо вибір статусу FormControl.

```
filterControl :FormControl<string | null>  = new FormControl('');
statusControl :FormControl<string | null>  = new FormControl('');

constructor(  no usages  Kryvyi Yaroslav *
```

```
<mat-form-field>
  <mat-label>Фільтр за статусом</mat-label>
  <mat-select (selectionChange)="onSelected($event)" [formControl]="statusControl">
    <mat-option [value]="''">Всі</mat-option>
    <mat-option [value]="TaskStatus.TODO">До роботи</mat-option>
```

У ngOnInit підпишемося на зміни цих полів. При кожній зміні будемо переходити на першу сторінку.

```
57
58  ngOnInit(): void {  no usages  Kryvyi Yaroslav *
59    this.loadTasks(1, 5);
60
61    this.loading$ = this.store.select(TaskSelectors.selectTaskLoading);
62    this.error$ = this.store.select(TaskSelectors.selectTaskError);
63    this.totalTasks$ = this.store.select(TaskSelectors.selectTaskTotal);
64
65    this.filterControl.valueChanges.pipe(
66      startWith(''),
67      debounceTime(300),
68      distinctUntilChanged()
69    ).subscribe(() : void  => this.paginator.firstPage());
70
71    this.statusControl.valueChanges.subscribe(() : void  => this.paginator.firstPage());
72
73    this.store.select(TaskSelectors.selectAllTasks)
74      .pipe(takeUntil(this.destroy$))
75      .subscribe( tasks : Task[]  => this.dataSource.data = tasks);
76
77    this.error$.pipe(takeUntil(this.destroy$)).subscribe(error : string | null  => {
78      if (error) {
79        this.snackBar.open(error, 'Закрити', { duration: 4000, panelClass: ['error-snackbar'] });
80      }
81    });
82
83    this.loading$.pipe(takeUntil(this.destroy$)).subscribe((loading : boolean ) : void  => {
84      this.hasLoading = loading;
85    })
86  }
87
```

Оновлюємо ngAfterViewInit, підписуємося на подію (page), яка емітиться кожен раз, як пагінатор змінює розмір або номер сторінки.

```
91
92  ngAfterViewInit() : void  {  no usages  new *
93    this.paginator.page.pipe(
94      startWith({ pageIndex: 0, pageSize: 5 }),
95      switchMap(({ pageIndex, pageSize } : PageEvent | { pageIndex: number; pageSize... ) : Observable<[string | null, string | null]...  =>
96        combineLatest([
97          this.filterControl.valueChanges.pipe(startWith(this.filterControl.value)),
98          this.statusControl.valueChanges.pipe(startWith(this.statusControl.value))
99        ]).pipe(
100         takeUntil(this.destroy$),
101         tap(([filter, status] : [string | null, string | null] ) : void  => {
102           this.loadTasks(pageIndex + 1, pageSize, filter ?? '', status ?? '');
103         })
104       )
105     )
106   ).subscribe();
107 }
```

Тепер переходимо до серверної обробки

Оновлюємо **task.controller.js**

```
const getTasks = async (req, res) :Promise<void> => {  Show usages  Kryvyi Yaroslav *
  try {
    const page :number = parseInt(req.query.page) || 1;
    const limit :number = parseInt(req.query.limit) || 5;
    const skip :number = (page - 1) * limit;

    const filter = req.query.filter?.trim().toLowerCase() || '';
    const status = req.query.status;

    const searchQuery :{…}|{…} = filter
      ? {
        $or: [
          { title: { $regex: filter, $options: 'i' } },
          { description: { $regex: filter, $options: 'i' } },
          { assignee: { $regex: filter, $options: 'i' } }
        ]
      }
      : {};

    const statusQuery :{…}|{…} = status ? { status } : {};

    const query :{} = {...searchQuery, ...statusQuery};

    const [tasks :… | number, total :… | number] = await Promise.all([
      TaskModel.find(query).skip(skip).limit(limit),
      TaskModel.countDocuments(query)
    ]);

    res.json({ tasks, total });
  } catch (error) {
    res.status(500).json({ message: "Error retrieving tasks", errors: error });
  }
```

Також у зв’язку з оновленням, змінимо код у task-list
Знов введемо змінну для завдань

```
  totalTasks$!: Observable<number>;

  tasks$!: Observable<Task[]>;
```

Оновимо ngOnInit

```
ngOnInit(): void {  no usages  Kryvyi Yaroslav *
  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;
  })
}
```

Оновлюємо методи, які відповідають за реакцію на додавання і видалення завдання. Це необхідно для правильної логіки пагінації

```
openDialog() : void {  Show usages  Kryvyi Yaroslav *
  const dialogRef : MatDialogRef<TaskFormComponent, any>  = this.dialog.open(TaskFormComponent, {
    height: '70vh',
    width: '80vw',
  });

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

Реакцію на створення пропишемо у openDialog. Будемо реагувати на закриття вікна у разі успіху, і переходити на останню сторінку. Для цього реалізуємо метод переходу

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

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

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

І відкорегуємо відправку форми у task-form

```
  onSubmit(): void {  Show usages  Kryvyi Yaroslav *
    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 : boolean  => !isLoading),
          take(1)
        )
        .subscribe(() : void  => {
          this.store.select(TaskSelectors.selectTaskError)
            .pipe(take(1))
            .subscribe(error : string | null  => {
              if (!error) {
                this.dialogRef.close(this.editMode ? 'updated' : 'created');
                this.store.dispatch(TaskActions.selectTask({ id: null }));
              }
            });
        });
    }
  }
```

Тепер ми підписані на подію очікування виконання операції. По її завершенню ми підписуємося на помилку, і залежно від стану форми надсилаємо повідомлення закриття форми і анулюємо обране завдання. На помилку необхідно підписатися, щоб форма не закривалася, якщо буде помилка створення або оновлення.

Також оновлюємо шаблон task-list, обгорнувши table в @if

```
@if(tasks$ | async; as tasks) {
        <table mat-table [dataSource]="tasks" class="mat-elevation-z2">

        <!-- Назва -->
        <ng-container matColumnDef="title">
```

```
            <th mat-header-cell *matHeaderCellDef> Назва </th>
            <td mat-cell *matCellDef="let task"> {{ task.title }} </td>
            </ng-container>

            <ng-container matColumnDef="description">
            <th mat-header-cell *matHeaderCellDef> Опис </th>
            <td mat-cell *matCellDef="let task"> {{ task.description }} </td>
            </ng-container>

            <!-- Виконавець -->
            <ng-container matColumnDef="assignee">
            <th mat-header-cell *matHeaderCellDef> Виконавець </th>
            <td mat-cell *matCellDef="let task"> {{ task.assignee }} </td>
            </ng-container>

            <!-- Дата -->
            <ng-container matColumnDef="dueDate">
            <th mat-header-cell *matHeaderCellDef> Дата </th>
            <td mat-cell *matCellDef="let task"> {{ task.dueDate }} </td>
            </ng-container>

            <!-- Статус -->
            <ng-container matColumnDef="status">
            <th mat-header-cell *matHeaderCellDef> Статус </th>
            <td mat-cell *matCellDef="let task"> {{ task.status | taskStatus}} </td>
            </ng-container>

            <!-- Дії -->
            <ng-container matColumnDef="actions">
            <th mat-header-cell *matHeaderCellDef> Дії </th>
            <td mat-cell *matCellDef="let task">
            <button mat-button (click)="viewTask(task.id)">Переглянути</button>
            <button mat-button (click)="editTask(task.id)">Редагувати</button>
            <button mat-button color="warn" (click)="deleteTask(task.id)">Видалити</button>
            </td>
            </ng-container>

            <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
            <tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>
            </table>
            }
```

Далі обробимо дію видалення. У task-list оновлюємо функцію deleteTask

```
    this.openDialog();
  }

  deleteTask(id: string): void {  Show usages  new *
    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;
    });
  }

  ngOnDestroy() : void  {  no usages  Kryvyi Yaroslav
```

Але з нею не так просто, як з додаванням/оновлення. Після якого ми просто завантажували Store. Нам необхідно отримати для початку кількість документів після видалення (і так, це знов потрібно, щоб пагінація працювала).

Для цього оновимо метод видалення у контролері, щоб він повертав кількість документів. Використовувати Promise.all у цьому випадку не вийде, так як він виконує запити паралельно.

```
    }
    res.status(400).json({ message: "Error partially updating task", errors: error });
  }
};

const deleteTask = async (req, res) : Promise<...>  => {  Show usages  Kryvyi Yaroslav *
  try {

    const deletedTask : Query<...> & ObtainSchemaGeneric<module:mon...  = await TaskModel.findByIdAndDelete(req.params.id);

    if (!deletedTask) {
      return res.status(404).json({ message: "Task not found" });
    }

    const total : Query<...> & ObtainSchemaGeneric<module:mon...  = await TaskModel.countDocuments();

    res.json({ message: "Task deleted", total });
  } catch (error) {
    res.status(500).json({ message: "Error deleting task", errors: error });
  }
};

module.exports = { getTasks, createTask, updateTask, patchTask, deleteTask };
```

Модернізуємо deleteTaskSuccess, додавши у payload кількість завдань

```
);

export const deleteTaskSuccess : ActionCreator<'[Task] Delete Success', (p... = createAction( Show usages  Kryvyi Yaroslav *
  '[Task] Delete Success',
  props<{ id: string, total: number }>()
);
```

Модернізуємо обробник у reducer

```
    loading: true,
  })),

  on(TaskActions.deleteTaskSuccess, (state: TaskState, {id, total} : { id: string; total: number } & Action<"[... ) : { total: number; loading: boo
    taskAdapter.removeOne(id, {...state, total, loading: false}),
  ),
```

Оновлюємо effects

```
deleteTask$ : Observable<({ id: string; total: number }... = createEffect(() : Observable<({ id: string; total: number }... =>
  this.actions$.pipe(
    ofType(TaskActions.deleteTask),
    mergeMap(({id} : { id: string } & Action<'[Task] Delete'> ) : Observable<({ id: string; total: number }... =>
      this.taskService.deleteTask(id).pipe(
        map((res : { message: string; total: number } ) : { id: string; total: number } & Action<'[... => TaskActions.deleteTaskSuccess({id, total: res.total})),
        catchError(err : any => {
          return of(TaskActions.deleteTaskFailure({error: formatError(err)}));
        })
      )
    )
  )
);
```

І також оновимо два методи у task.service. А саме приберемо delay з getTasks (він буде заважати, так як буде повертати кількість до видалення)

```
getTasks(page: number, pageSize: number, filter?: string, status?: string): Observable<TaskLoad> {  Show usages  Kryvyi Yaroslav *
  let params : HttpParams = new HttpParams()
    .set('page', page)
    .set('limit', pageSize);

  if (filter) params = params.set('filter', filter);
  if (status) params = params.set('status', status);

  return this.http.get<TaskLoadApi>(`${this.config.apiUrl}/v2/tasks`, {params: params}).pipe(
    map(TaskAdapter.fromLoadAPI),
  );
}
```

А також змінимо тип Observable у deleteTask на {message:string, total: number}

```
47
48    deleteTask(id: string): Observable<{message:string, total: number}> {  Show usages  new *
49      return this.http.delete<{message:string, total: number}>(`${this.config.apiUrl}/v2/tasks/${id}`);
50    }
51
```

Тепер при видаленні буде змінюватися кількість завдань, що і потрібно для правильної роботи пагінації.

5) Тепер переходимо до task-item. Нам необхідно оновити логіку, так як ми більше не отримуємо завдання через @Input, будемо брати завдання зв Store. ID будемо брати за допомогою @ngrx/router-store

```
export class TaskItemComponent implements OnInit, OnDestroy {

  task$!: Observable<Task | null>;
  destroy$ : Subject<void> = new Subject<void>();

  constructor(private store: Store<AppState>) {  no usages  Kryvyi Yaroslav
  }

  ngOnInit(): void {  no usages  new *
    this.task$ = this.store.select(selectRouteParams).pipe(
      takeUntil(this.destroy$),
      switchMap(params : Params  => this.store.select(selectTaskById(params['id'])))
    );
  }
```

Для отримання параметрів, використаємо селектор, який надає router-store, а саме selectRouteParams (можна використати і фабричний селектор). Обов’язково прописуємо логіку відписки та через switchMap будемо отримувати останній ID. Далі нам необхідно створити селектор для отримання завдання за ID. Оновимо файл **task.selectors.ts.**

```
);


export const selectTaskById : (id: string) => MemoizedSelector<object, ... = (id: string) : MemoizedSelector<object, Task | null, (s1... =>  Show usages  new *
  createSelector(
    selectAllTasks,
    state : Task[] => {
      return state.find(task : Task => task.id === id) ?? null;
    }
  )
```

Також з цього компонента ми зняли відповідальність за видалення/виклику оновлення, тому видаляємо ці методи. Також трохи відкорегуємо методи оновлення статусу та отримання класу за статусом.

```
  );
}

updateStatus( id: string, event: MatSelectChange) :void { Show usages  Kryvyi Yaroslav *
  const selectedValue :any = event.value;
  this.store.dispatch(TaskActions.patchTask({id: id, changes: {status: selectedValue}}));
}

getStatusClass(status: TaskStatus): string { Show usages  new *
  return `chip-${status.toLowerCase()}`;
}

ngOnDestroy() :void { no usages  new *
  this.destroy$.next();
  this.destroy$.complete();
}

protected readonly TaskStatus :typeof TaskStatus = TaskStatus;

}
```

Оновимо шаблон

```
@if (task$ | async; as task) {
  <section class="item-wrapper">
            <div class="item-card">
            <div class="task-header">{{ task.title }}</div>

            <div class="task-field">
            <span class="label">Виконавець:</span>
            <span class="value">{{ task.assignee }}</span>
            </div>

            @if (task.description) {
            <div class="task-field">
            <span class="label">Опис:</span>
            <span class="value">{{ task.description }}</span>
            </div>
            }

            <div class="task-field">
            <span class="label">Дедлайн:</span>
            <span class="value">{{ task.dueDate }}</span>
            </div>

            <div class="task-field">
            <span class="label">Статус:</span>
            <mat-chip [ngClass]="getStatusClass(task.status)">
            {{ task.status | taskStatus}}
            </mat-chip>
            </div>

            <div class="task-field">
            <mat-form-field appearance="fill" class="status-select">
            <mat-label>Статус</mat-label>
            <mat-select [value]="task.status" (selectionChange)="updateStatus(task.id, $event)">
            <mat-option [value]="TaskStatus.TODO">До роботи</mat-option>
            <mat-option [value]="TaskStatus.IN_PROGRESS">В процесі</mat-option>
            <mat-option [value]="TaskStatus.DONE">Виконано</mat-option>
```

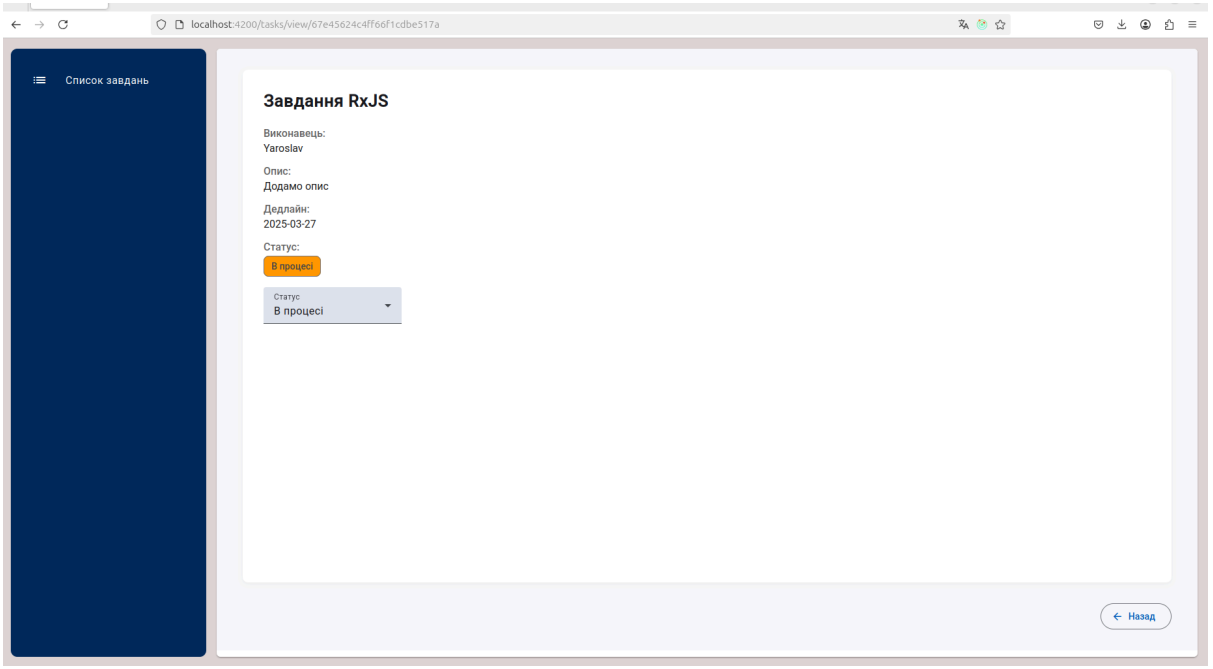
```
            </mat-select>
            </mat-form-field>
            </div>

            </div>

            <div class="actions">
            <button mat-stroked-button color="primary" [routerLink]="['/', { outlets: { primary: ['tasks'], aside: ['stats'] } }]"
            >
            <mat-icon>arrow_back</mat-icon>
            Назад
            </button>
            </div>
  </section>
}
```

Та стилі

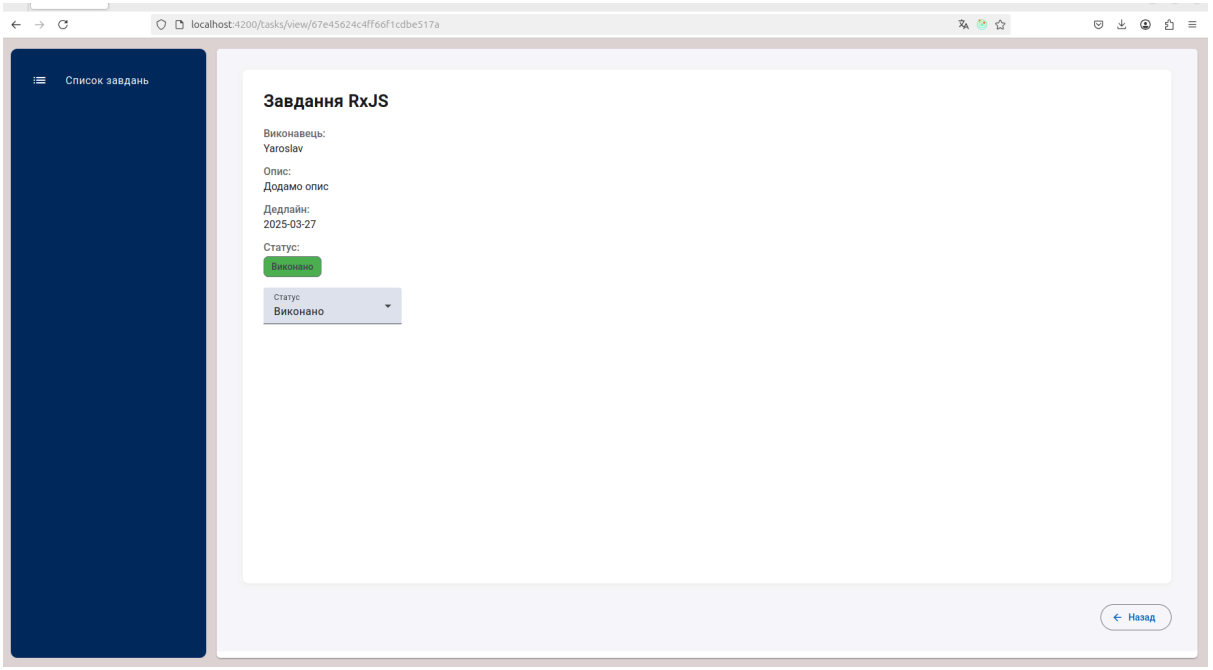
```
:host {
  display: flex;
  flex-direction: column;
  height: 100%;
  width: 100%;
  flex-grow: 1;
}

.item-wrapper {
  flex-grow: 1;
  display: flex;
  flex-direction: column;
  padding: 32px 40px;
  background-color: #f8f9fa;
  overflow-y: auto;
}

.item-card {
  background: #ffffff;
  padding: 32px;
  border-radius: 8px;
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.05);
  flex: 1;
  display: flex;
  flex-direction: column;
}

.task-header {
  font-size: 26px;
  font-weight: 600;
  margin-bottom: 24px;
}

.task-field {
  margin-bottom: 16px;

  .label {
```

```
            font-weight: 500;
            color: #666;
            display: block;
            margin-bottom: 4px;
  }

  .value {
            font-size: 16px;
            color: #222;
  }

  mat-chip.chip-todo {
            background-color: #f44336;
            color: white;
  }

  mat-chip.chip-in_progress {
            background-color: #ff9800;
            color: white;
  }

  mat-chip.chip-done {
            background-color: #4caf50;
            color: white;
  }
}

.actions {
  display: flex;
  justify-content: flex-end;
  margin-top: 32px;
}
```

Перевіримо відображення



Зміна статусу



6) Наступним пунктом реалізуємо task-stats (той компонент, який ми будемо відображати в іменованому outlet)

Він буде відображати:

1) Загальну кількість завдань;
2) Кількість завдань за кожним статусом;

Для отримання кількості завдань у нас вже є селектор, потрібно зробити для статусів.

```
export const selectTasksByStatus : (status: TaskStatus) => MemoizedSelector<... = (status: TaskStatus) : MemoizedSelector<object, number, (s1: Tas... => Sh
  createSelector(
    selectAllTasks,
    state : Task[] => {
      return state.filter(task : Task => task.status === status).length;
    }
  );
```

У **task-stats.component.ts** підписуємо через селектори на відповідні частини Store.

```
export class TaskStatsComponent implements OnInit {
  total$!: Observable<number>;
  countTodo$!: Observable<number>;
  countInProgress$!: Observable<number>;
  countDone$!: Observable<number>;

  constructor(private store: Store<AppState>) {}  no usages

  ngOnInit(): void {  no usages
    this.total$ = this.store.select(selectTaskTotal);
    this.countTodo$ = this.store.select(selectTasksByStatus(TaskStatus.TODO));
    this.countInProgress$ = this.store.select(selectTasksByStatus(TaskStatus.IN_PROGRESS));
    this.countDone$ = this.store.select(selectTasksByStatus(TaskStatus.DONE));
  }
}
```

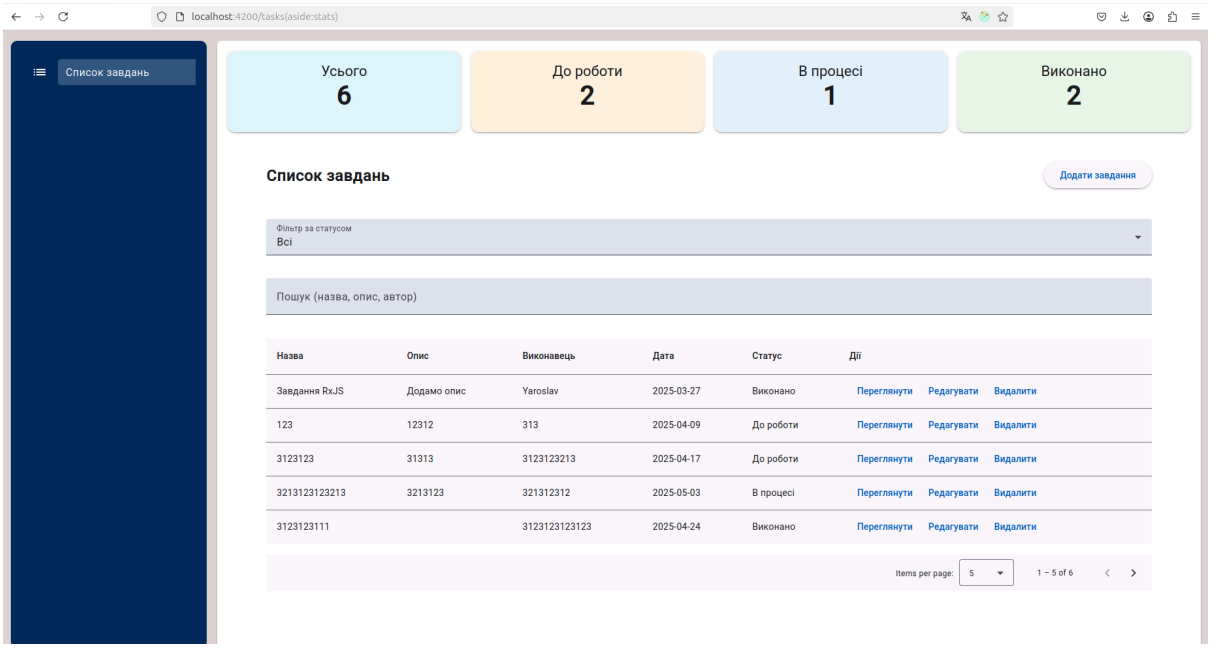
Формуємо шаблон

```
<div class="stats-wrapper">
  <mat-card class="stat-card total">
    <mat-card-title>Усього</mat-card-title>
    <mat-card-content>
      <h2>{{ total$ | async }}</h2>
    </mat-card-content>
  </mat-card>

  <mat-card class="stat-card todo">
    <mat-card-title>До роботи</mat-card-title>
    <mat-card-content>
      <h2>{{ countTodo$ | async }}</h2>
    </mat-card-content>
  </mat-card>

  <mat-card class="stat-card in-progress">
    <mat-card-title>В процесі</mat-card-title>
    <mat-card-content>
      <h2>{{ countInProgress$ | async }}</h2>
    </mat-card-content>
  </mat-card>

  <mat-card class="stat-card done">
    <mat-card-title>Виконано</mat-card-title>
    <mat-card-content>
      <h2>{{ countDone$ | async }}</h2>
    </mat-card-content>
  </mat-card>
</div>
```

Та стилі

```
.stats-wrapper {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  gap: 1rem;
  padding: 1rem;
}

.stat-card {
  padding: 1rem;
  display: flex;
  flex-direction: column;
  align-items: center;
  text-align: center;

  h2 {
        font-size: 2.5rem;
        margin: 0;
  }

  &.total {
        background-color: #e0f7fa;
  }

  &.todo {
        background-color: #fff3e0;
  }

  &.in-progress {
        background-color: #e3f2fd;
  }

  &.done {
        background-color: #e8f5e9;
  }
}
```

Отримуємо



Загальну кількість ми беремо зі значення Store, яке оновлюється кожен раз при запиті списку завдань та видалені. За статусами ми вже працюємо з наявними у Store даними. Якщо необхідно відображати кількість за статусами усіх наявних завдань, то необхідно повертати ці дані з серверу та записувати у Store.

7) Далі переходимо до page-not-found

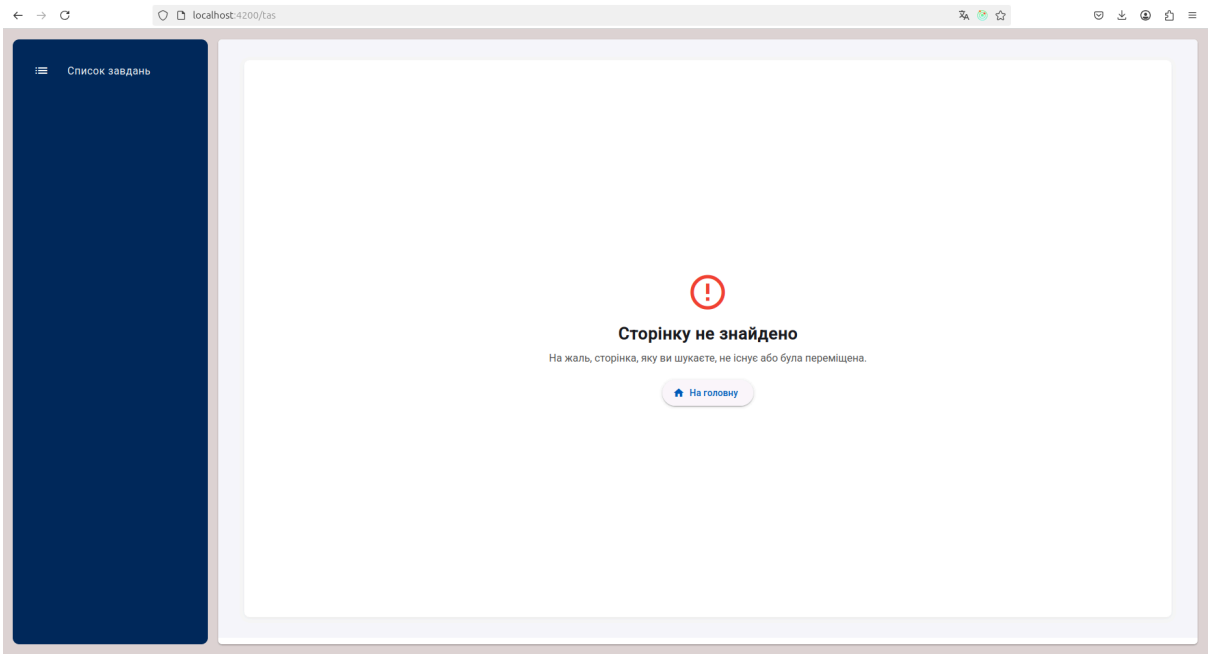
Тут нам потрібен тільки шаблон, який буде інформувати користувача, що сторінки не існує та буде містити кнопку навігації до головну сторінку

```
<div class="not-found-wrapper">
  <div class="not-found-card">
    <mat-icon class="icon">error_outline</mat-icon>
    <div class="title">Сторінку не знайдено</div>
    <div class="description">
      На жаль, сторінка, яку ви шукаєте, не існує або була переміщена.
    </div>
    <div class="actions">
      <button mat-raised-button color="primary" routerLink="/">
        <mat-icon>home</mat-icon>
        На головну
      </button>
    </div>
  </div>
</div>

```

Та стилі

```
:host {
  display: flex;
  flex-direction: column;
  height: 100%;
  width: 100%;
  flex-grow: 1;
}

.not-found-wrapper {
  flex-grow: 1;
  display: flex;
  flex-direction: column;
  padding: 32px 40px;
  background-color: #f8f9fa;
  overflow-y: auto;
}

.not-found-card {
  background: #ffffff;
  padding: 32px;
  border-radius: 8px;
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.05);
  display: flex;
  flex-direction: column;
  justify-content: center;
  flex-grow: 1;
  align-items: center;
  text-align: center;
}

.icon {
  font-size: 64px;
  color: #f44336;
  margin-bottom: 16px;

  display: inline-flex;
  align-items: center;
  justify-content: center;

  width: 64px;
  height: 64px;
}

.title {
  font-size: 26px;
  font-weight: 600;
  margin-bottom: 12px;
}

.description {
  font-size: 16px;
  color: #555;
  margin-bottom: 24px;
}

.actions {
  display: flex;
  justify-content: center;
}
```

Перевіримо відображення



Контрольні питання

1) Що таке Angular Router?
2) Яку структуру має Route? Яка з властивостей відповідає за переадресацію з одного шляху на інший?
3) Які є типи маршрутів?
4) Як отримати параметри маршруту?
5) Що таке Guard?