02 Modeling

Edit New Page

rofr edited this page 13 hours ago · 5 commits

This module goes a bit deeper into modeling.

Reading material

- Slides PDF
- Modeling basics

Exercise 1

In this exercise you will extend the Todo.Core model from module 1 to support categories, and create commands and queries with emphasis on the topics presented in the slides.

Task 1 - Add categories to the data model

1. Open the Todo.Core solution in Visual studio located in the `Ex01` folder
2. Add a class named `Category` with a `Name` and `HashSet<Todo>` property
3. Add a `HashSet<Category>` property to the `Todo` class
4. Add a `Dictionary<string,Category>` to the `TodoModel` class

Task 2 - Implement the following commands

1. `SetCategoryCommand` - add a todo item to a category, create the category if necessary
2. `ClearCategoryCommand` - remove a todo item from a category
3. `SetCategoriesCommand` - set the category associations for a given todo

Task 3 - Test the model using scriptcs

1. Create a todo.csx file in the excercise folder with the following content:

```
#r Todo.Core\bin\debug\Todo.Core.dll

using Todo.Core;
using OrigoDB.Core;

var engine = Engine.LoadOrCreate<TodoModel>();

var haskell = engine.Execute(new AddTodoCommand("Learn Haskell"));
var cook = engine.Execute(new AddTodoCommand("Lamb Roast on Sunday"));
var dishes = engine.Execute(new AddTodoCommand("Do the dishes"));

engine.Execute(new SetCategoriesCommand{
   TodoId = haskell,
   Categories = new[]{"work", "play"}
});

engine.Execute(new SetCategoryCommand(cook,"play"));
engine.Execute(new SetCategoryCommand(dishes, "work"));
```

2. Launch the script and remain in the REPL.
3. Have a look at a dump of the model by typing `engine.GetModel()`

Task 4 - Working with lambda queries

In the scriptcs repl, type lambdas to return the following:

1. Category names
2. Todo item names in alphabetical order
3. Unfinished todo items
4. Overdue todo items

See the Completed folder for example solutions in the file `lambdas.csx`

Task 5 - Queries and Views

1. In Visual Studio, create the following view class:

```
[Serializable]
public class TodoView
{
    public readonly int Id;
    public readonly string Title;
    public readonly string[] Categories;

    public TodoView(Todo todo)
    {
        Id = todo.Id;
        Title = todo.Title;
        Categories = todo.Categories.Select(c => c.Name).ToArray();
    }
}
```

2. Create a query to retrieve todos by id:

```
[Serializable]
public class GetTodoByIdQuery : Query<TodoModel, TodoView>
{
    public readonly int Id;

    public GetTodoByIdQuery(int id)
    {
        Id = id;
    }

    public override TodoView Execute(TodoModel model)
    {
        return new TodoView(model.Todos[Id]);
    }
}
```

3. Create a query named PagedTodosQuery returning TodoView[] with Skip and Take properties (order by Id).
4. Rewrite one or more of the lambdas from task 4 as queries, try LINQ syntax

+ Add a custom footer