Міністерство освіти і науки України
Національний технічний університет
України
“Київський політехнічний інститут імені Ігоря
Сікорського” Факультет інформатики та обчислювальної
техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №4

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Вступ до паттернів проектування»

Виконав:
студент групи ІА-з31
Курбатов Кіріл Андрійович

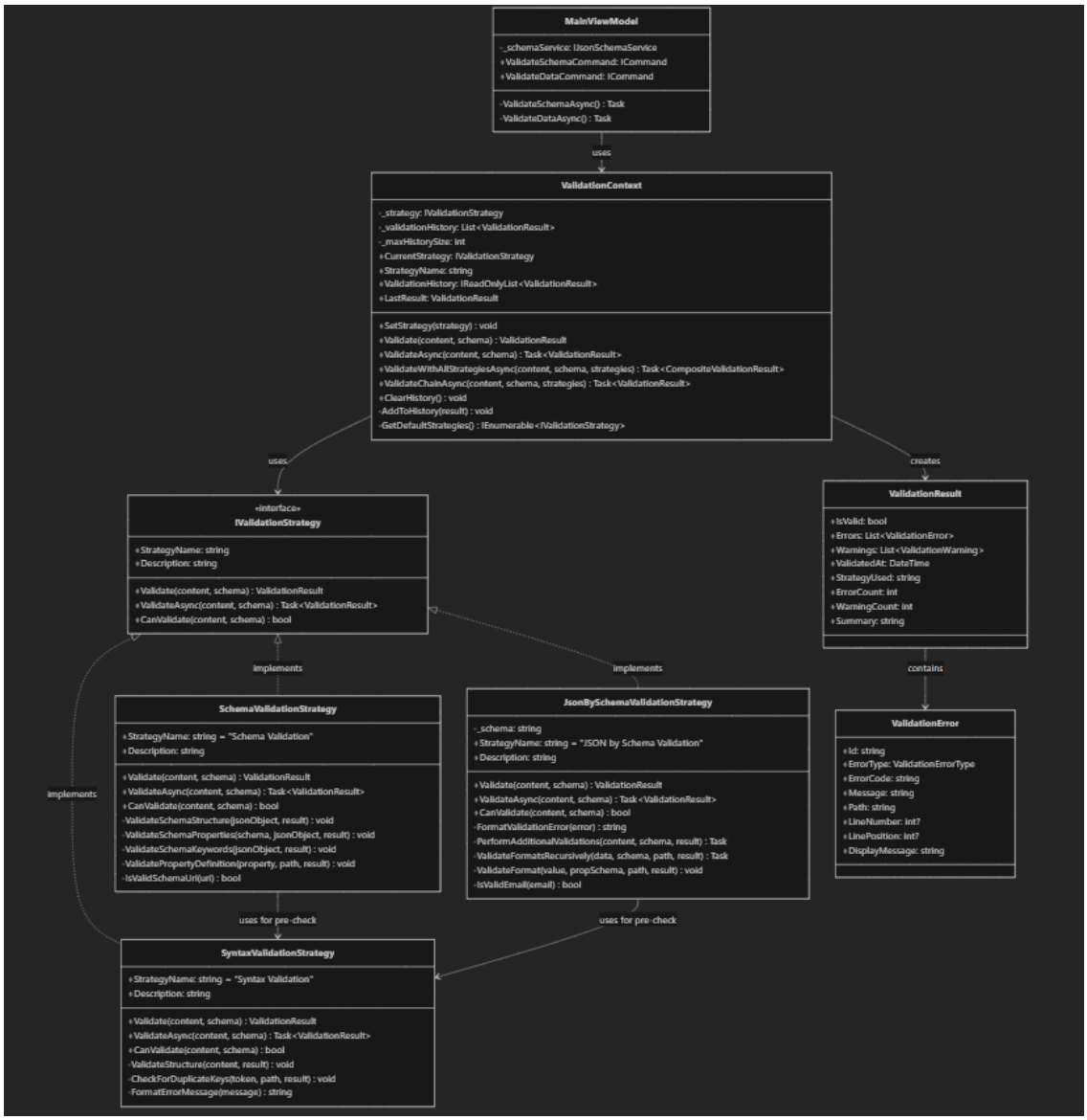
Перевірив:
Амонс Олександр Анатолійович

Київ 2025

Мета: Вивчити структуру шаблонів «Singleton», «Iterator», «Proxy», «State», «Strategy» та навчитися застосовувати їх в реалізації програмної системи.

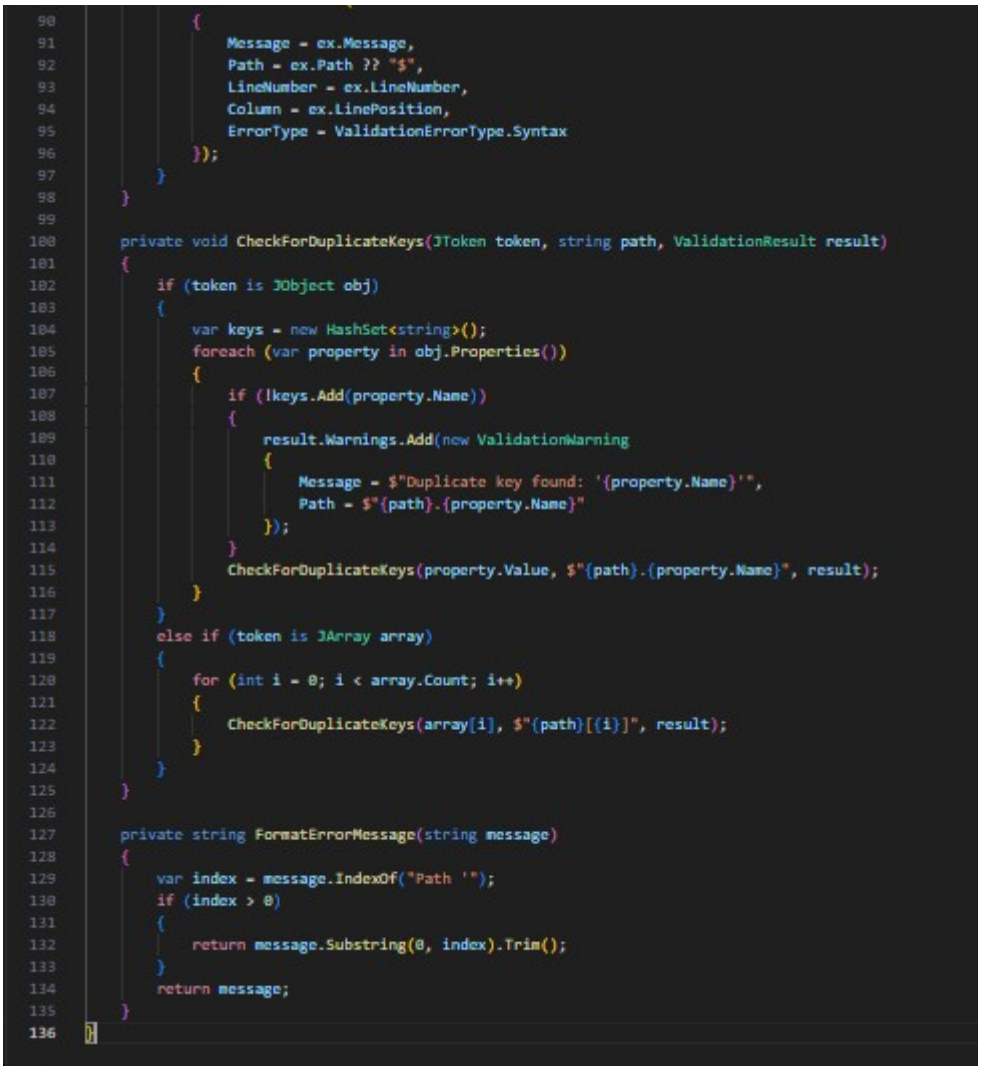
Завдання

• Ознайомитись з короткими теоретичними відомостями.
• Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
• Реалізувати один з розглянутих шаблонів за обраною темою.
• Реалізувати не менше 3-х класів відповідно до обраної теми.
• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону.



Реалізація патерну Strategy

```
using System.IO;
using JsonTool.Core.Models;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;

namespace JsonTool.Core.Strategy.Validation;

public class SyntaxValidationStrategy : IValidationStrategy
{
    public string StrategyName => "Syntax Validation";
    public string Description => "Validates JSON syntax correctness";

    public bool CanValidate(string content, string? schema = null)
    {
        return !string.IsNullOrEmpty(content);
    }

    public ValidationResult Validate(string content, string? schema = null)
    {
        var result = new ValidationResult { IsValid = true };

        if (string.IsNullOrWhiteSpace(content))
        {
            result.IsValid = false;
            result.Errors.Add(new ValidationError
            {
                Message = "JSON content is empty or whitespace",
                Path = "$",
                LineNumber = 1,
                Column = 1,
                ErrorType = ValidationErrorType.Syntax
            });
            return result;
        }

        try
        {
            using var stringReader = new StringReader(content);
            using var jsonReader = new JsonTextReader(stringReader);

            while (jsonReader.Read()) { }
        }
        catch (JsonReaderException ex)
        {
            result.IsValid = false;
            result.Errors.Add(new ValidationError
            {
                Message = FormatErrorMessage(ex.Message),
                Path = ex.Path ?? "$",
                LineNumber = ex.LineNumber,
                Column = ex.LinePosition,
                ErrorType = ValidationErrorType.Syntax
            });
        }

        if (result.IsValid)
        {
            ValidateStructure(content, result);
        }

        return result;
    }

    public Task<ValidationResult> ValidateAsync(string content, string? schema = null)
    {
        return Task.FromResult(Validate(content, schema));
    }

    private void ValidateStructure(string content, ValidationResult result)
    {
        try
        {
            var token = JToken.Parse(content);

            if (token.Type != JTokenType.Object && token.Type != JTokenType.Array)
            {
                result.Warnings.Add(new ValidationWarning
                {
                    Message = $"Root element is {token.Type}, expected Object or Array",
                    Path = "$"
                });
            }

            CheckForDuplicateKeys(token, "$", result);
        }
        catch (JsonReaderException ex)
        {
            result.IsValid = false;
            result.Errors.Add(new ValidationError
```



```
            {
                Message = ex.Message,
                Path = ex.Path ?? "$",
                LineNumber = ex.LineNumber,
                Column = ex.LinePosition,
                ErrorType = ValidationErrorType.Syntax
            });
        }
    }

    private void CheckForDuplicateKeys(JToken token, string path, ValidationResult result)
    {
        if (token is JObject obj)
        {
            var keys = new HashSet<string>();
            foreach (var property in obj.Properties())
            {
                if (!keys.Add(property.Name))
                {
                    result.Warnings.Add(new ValidationWarning
                    {
                        Message = $"Duplicate key found: '{property.Name}'",
                        Path = $"{path}.{property.Name}"
                    });
                }
                CheckForDuplicateKeys(property.Value, $"{path}.{property.Name}", result);
            }
        }
        else if (token is JArray array)
        {
            for (int i = 0; i < array.Count; i++)
            {
                CheckForDuplicateKeys(array[i], $"{path}[{i}]", result);
            }
        }
    }

    private string FormatErrorMessage(string message)
    {
        var index = message.IndexOf("Path '");
        if (index > 0)
        {
            return message.Substring(0, index).Trim();
        }
        return message;
    }
}
```