Задание №2. Создание решения

Описание задания:

1. Создайте новую ветку в локальном репозитории под названием «features/1_add_solution». Перейдите в новую ветку.
2. Создайте в корневой папке локального репозитория три папки: «docs», «src», «demo». Папка docs понадобится в последних заданиях для проектной документации; папка demo понадобится для хранения демонстрационной версии программы после её полного завершения. Папка src (сокращение от sources) предназначена для хранения исходного кода программы.
3. С помощью Visual Studio создайте в папке src решение с проектом «Windows Forms Application (.NET Framework)». Решение должно называться также, как и программа. **Важно:**
   - решение (solution) должен называться именем приложения (например, NoteApp);
   - проект (project) должен называться именем приложения с приставкой «*.View» (например, NoteApp.View), как указание, что этот проект будет содержать классы пользовательского интерфейса;
   - проект обязательно должен быть «Windows Forms Application **(.NET Framework)**». Среди шаблонов проектов в Visual Studio есть и другие шаблоны Windows Forms Application, но не основанные на .NET Framework;
   - созданное решение должно быть в папке sln.
4. Добавьте в решение проект «Class Library (.NET Framework)». Проект должен называться именем приложения с приставкой «*.Model» (например, NoteApp.Model), как указание того, что этот проект будет содержать классы бизнес-логики.
5. В результате предыдущих шагов, структура папок и файлов в папке репозитория должна быть следующая (пример для приложения NoteApp):
6. Если структура папок совпадает с примером выше, сделайте фиксацию в репозитории. Добавьте комментарий к фиксации, например: «feature(NoteApp.sln): добавлено решение приложения с пустыми проектами для пользовательского интерфейса и бизнес-логики». Синхронизируйте ветку с удаленным репозиторием. Через GitHub убедитесь, что новая ветка теперь есть и в удаленном репозитории.
7. В Visual Studio перейдите в ветку develop. Затем выполните слияние (merge) ветки features/1_add_solution в ветку develop. Теперь все изменения из ветки с добавленным решением должны переместиться в ветку develop. Убедитесь в этом с помощью GitHub.

8. Каждая последующая новая задача также должна решаться в отдельной ветке с подзаголовком «features» (или «fixes» в случае исправления ошибок). Когда задача выполнена и проверена, рабочая ветка сливается с веткой develop. Таким образом, в develop всегда находится работающий код, а если в вашей рабочей ветке что-то пойдет не так, вы всегда сможете её удалить и создать заново из develop. Такой подход важен при командной разработке в одном репозитории.

Типовые ошибки и возможные проблемы:

Наиболее частая проблема – неправильно названное решение или проекты. Например, если приложение называется NoteApp, то неправильно будет называть решение Noteapp, noteapp, NotesApp, NotesAp и т.д. Также неправильно называть проект Noteapp.View, NoteAppView, ViewNoteapp, NoteApp.view. Правильное название – NoteApp.View.

Будьте внимательными к регистрам и грамматическим ошибкам. Это важно по нескольким причинам:

- название приложения должно соответствовать ТЗ.
- название приложения должно везде указываться единообразно.
- операционные системы чувствительны к регистрам в названиях файлов и папок. Папки «NoteApp» и «Noteapp» с точки зрения любой ОС – это две разные папки. В результате некоторые программы, работающие с файлами исходного кода, или сценарии по автоматизации сборки приложения, могут начать работать некорректно.
- автодополнение в Visual Studio или другой среде разработке не будет находить варианты с грамматическими ошибками, что будет мешать вам писать код быстро.
- сдавать приложение заказчику или отдавать пользователям, когда в нём есть грамматические ошибки в названии или в интерфейсе – стыдно и непрофессионально.

Вторая частая проблема – не тот тип проекта (проекты не относятся к .NET Framework). Во-первых, в Visual Studio есть несколько шаблонов проектов с ***похожими*** названиями, но они отличаются фреймворком (например, .NET Framework или .NET Core – два разных фреймворка). Это существенное отличие. Во-вторых, в зависимости от версии среды разработки или самой среды (если вы используете не Visual Studio), у вас может не быть варианта проекта с .NET Framework. В этом случае, можете выбрать другой шаблон проекта, например,

«Windows Forms Application **(.NET Core)**». Но в таком случае, и проект бизнес-логики тоже должен относиться к этому же фреймворку («Class Library **.NET Core**»). Ключевое при создании проектов убедиться, что все проекты вашего решения используют один и тот же фреймворк. Если фреймворки будут разные (один проект для .NET Framework, а другой для .NET Core), в последующих заданиях могут возникнуть самые непредвиденные ошибки.

Защита задания:

1. Во время защиты студент должен показать структуру папок созданного решения.
2. Решение (sln-файл) должен быть назван по имени приложения (например, NoteApp.sln) и лежать в папке src.
3. Рядом с sln-файлом должны лежать две папки проектов пользовательского интерфейса и бизнес-логики (например, NoteApp.View и NoteApp.Model). Наличие приставок, указанных через точку и соблюдение регистров обязательно.
4. В папках проектов должны лежать соответствующие *.csproj-файлы (файлы проектов для языка C#).
5. Рядом с папкой src также должны быть две пустые папки docs и demo.
6. Решение должно корректно открываться в Visual Studio.
7. Имя решения и проектов в Visual Studio (панель Solution Explorer) должны быть правильными (наличие приставок, соблюдение регистров).
8. Проект пользовательского интерфейса должен быть проектом Windows Forms Application для .NET Framework; проект бизнес-логики должен быть проектом Class Library для .NET Framework. Тип проекта и фреймворка можно проверить в свойствах проекта (правый клик мыши по названию проекта на панели Solution Explorer, пункт «Свойства» в контекстном меню).
9. Студент должен показать на GitHub, что при выполнении задания была создана отдельная ветка features/1_add_solution, и она была слита в develop.