Rush 00. Breaking the Habit

## Общие требования

* Убедитесь, что на вашем компьютере установлен [SDK для разработки на .NET 5](https://dotnet.microsoft.com/download) и вы используете именно его.
* Помните, ваш код будут читать! Обратите особое внимание на оформление вашего кода и именование переменных. Обязательно придерживайтесь общепринятых стандартов [C# Coding Conventions](https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions).
* Самостоятельно выберите удобную для себя IDE.
* Программа должна иметь возможность запуска через командную строку dotnet.
* В задании указаны примеры ввода и вывода. Решение должно использовать их как верный формат.
* Если затрудняетесь в решении задачи, обратитесь с вопросами к другим участникам бассейна, интернету, Google, посмотрите на StackOverflow.
* С основными возможностями языка C# можно ознакомиться в [официальной спецификации](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/language-specification/introduction).
* Избегайте **хардкода** и “**магических чисел**”.
* Вы демонстрируете все решение, верный результат работы программы – лишь один из способов проверки ее корректной работы. Поэтому когда необходимо получить определенный вывод в результате работы ваших программ, запрещено показывать пред рассчитанный результат.
* Обращайте особое внимание на термины, выделенные **bold** шрифтом: их лучше сразу погуглить, их изучение пригодится вам как в выполнении текущего задания, так и в вашей дальнейшей карьере .NET разработчика.
* Have fun :)

## Требования к заданиям дня

* Используйте **var**.
* Название решения (и его отдельного каталога) должно выглядеть как d{*xx*}, где *xx* - цифры текущего дня. Названия проектов указаны в задании.
* Для форматирования выходных данных используйте **культуру** en-GB: N2 для вывода денежных сумм, d для дат.

# Интро

До 2003 года профессиональных велосипедистов Великобритании не воспринимали всерьез. С 1908 года они завоевали лишь одну золотую медаль на Олимпийских играх. А в гонке «Тур де Франс» не выиграли ни разу за 110 лет.

Всё изменилось, когда руководство наняло тренера Дэйва Бреилсфорда. Он не просто тренировал команду, а менял среду, чтобы улучшить здоровье и комфорт спортсменов. По его указанию велосипедные сидения заменили на более удобные, в шины втирали спирт для лучшего сцепления с дорогой. Спортсменам выдали одежду, которая поддерживает нужную температуру во время езды. Дэйв следил за нагрузками спортсменов с помощью датчиков, подобрал гель для массажа для быстрого восстановления мышц и поменял подушки и матрасы, чтобы спортсмены лучше высыпались.

Это лишь часть того, что он сделал. В комплексе его действия привели к поразительным результатам. Через 5 лет велосипедисты из Великобритании завоевали 60% золотых медалей на Олимпийских играх в Пекине. А на Олимпиаде в Лондоне они установили 9 олимпийских рекордов и 7 мировых.

Журналист издания «Нью-Йорк Таймс» Джеймс Клир исследует привычки, пишет о них [в своём блоге](https://jamesclear.com/articles) и [книге «Атомные привычки»](https://www.labirint.ru/books/727534/). В статьях он рассказывает, что люди склонны думать о себе лучше, чем есть на самом деле. Это происходит, потому что так устроен наш мозг.

В качестве примера он приводит исследования, в которых первая группа худеющих вела пищевые дневники, а вторая нет. Так вот люди из первой группы похудели в два раза сильнее, чем из второй, за одно и то же время. Такую разницу объяснили тем, что первая группа видела реальную картину и лучше контролировала себя. В то время как люди из второй группы порой не замечали, что съели лишний бутерброд или выпили колу вместо минералки.

Для визуализации подобной картины существуют так называемые трекеры привычек. Как они работают? Они помогают формировать:

1. Визуальный триггер. Трекер сам по себе напоминает о том, что нужно сделать задание.
2. Наглядность прогресса. Когда мы видим закрашенные квадратики, появляется стимул не бросать начатое.
3. Вознаграждение. Нам приятно вычеркнуть еще один день, но чтобы сделать это, нужно выполнить задание.

При этом не стоит браться сразу за несколько привычек. Человеку трудно привыкать одновременно к нескольким изменениям, так куда ярче появляется сопротивление и желание все бросить.

# 00. Предметная модель

“Если мы регулярно повторяем какое-то действие, в нашем мозге формируются нейронные связи. Можно считать это алгоритмом или готовой программой, которая в дальнейшем позволяет нам выполнять эту задачу куда легче и быстрее и даже машинально.”

- [Psychology of Habit](https://www.researchgate.net/publication/281679387_Psychology_of_Habit)

В рамках этого дня мы напишем свой трекер привычек. В интро можно почерпнуть немало полезной информации, и именно она поможет нам сделать первый и один из самых важных шагов в разработке любого проекта - разработка модели предметной области.

Что будет делать наш трекер? Мы должны иметь возможность описать привычку и отмечать прогресс ее соблюдения (или отказа от нее) каждый день в течение заданного срока. По сути, ставить каждый день галочки.

Значит нам нужен объект *Habit*, описывающий привычку: ее заголовок (*Title*) и мотивацию поддержания трекинга (*Motivation*). Для привычки нужно будет вести прогресс: его можно задать коллекцией объектов *HabitCheck* для каждого из планируемых дней трекинга, которые будут определяться свойствами *Date* и *IsChecked*. Как только все дни прошли (вне зависимости, были ли отмечены все из них), трекинг завершен и это можно отметить в свойстве *IsFinished*.

Теперь у нас есть простая модель.

Условимся, что в любой момент времени мы можем трекать одну привычку. Это упрощение поможет нам и в разработке, и в отслеживании этой привычки. Значит если неоконченного трекинга в системе не существует, мы должны иметь возможность создать его, заполнив заголовок привычки, мотивацию, дату старта и количество дней трекинга.

Если какая-то привычка трекается, мы должны иметь возможность отметить ее соблюдение или отказ от нее в каждый из дней (поставить *IsChecked* для конкретного объекта *HabitCheck).* Отметку можно ставить за любую дату.

Если отмечена последняя дата или прошел последний день трекинга, свойство *IsFinished* объекта *Habit* должно стать true.

Перейдем к реализации.

# 01. Интерфейс

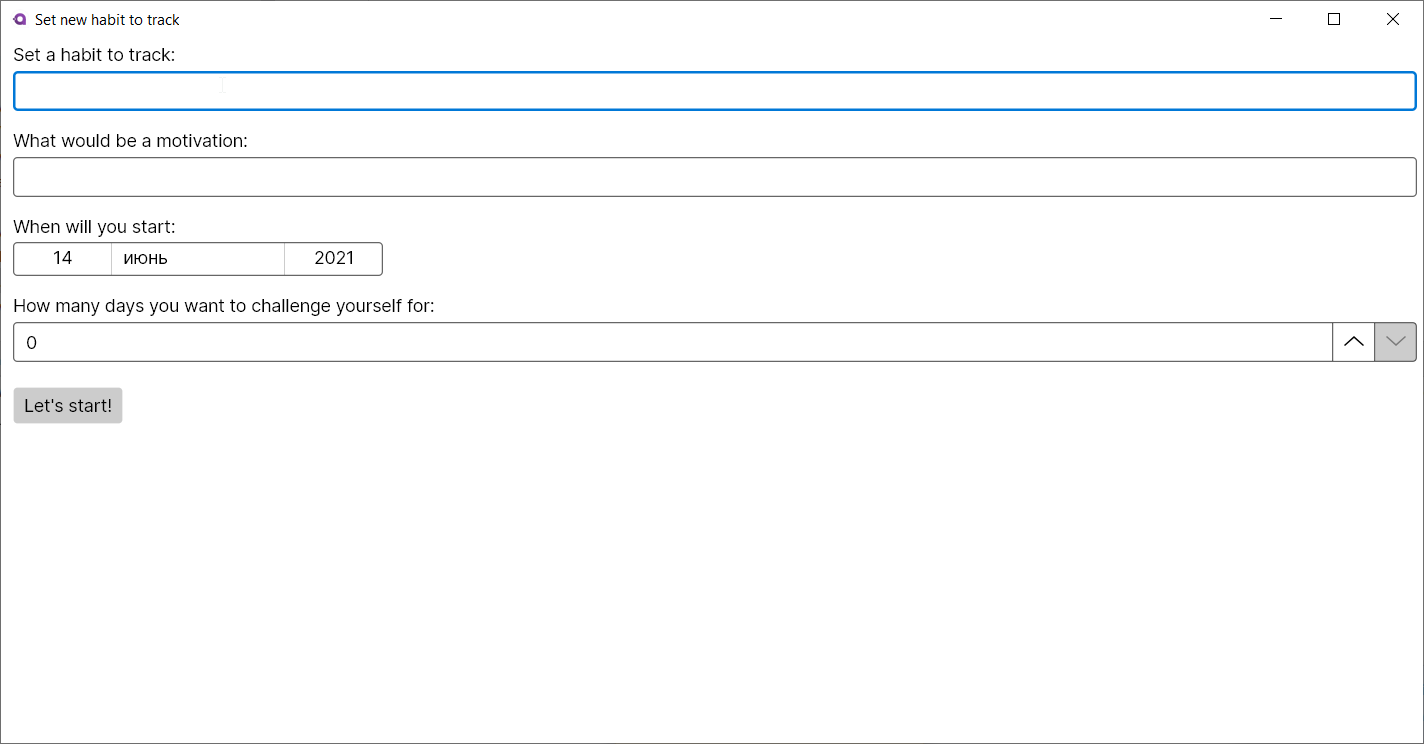
Реализуем несложный интерфейс для нашего трекера: пусть это будет **desktop приложение**. Для разработки .NET приложений для десктопа огромную популярность снискали **WPF** и **UWP** приложения, которые и являются одними из самых распространенных платформ. К сожалению, они очень ограничены тем, что позволяют разрабатывать только под Windows.

Мы же взглянем в сторону [Avalonia UI](https://docs.avaloniaui.net/). Это молодой, но активно развивающийся кросс-платформенный **UI фреймворк**, который сами разработчики называют духовным преемником технологии **WPF**. Он основан на **XAML разметке** и позволяет создавать desktop приложения для разных операционных систем. В разработке чаще всего используется шаблон проектирования **MVVM**.

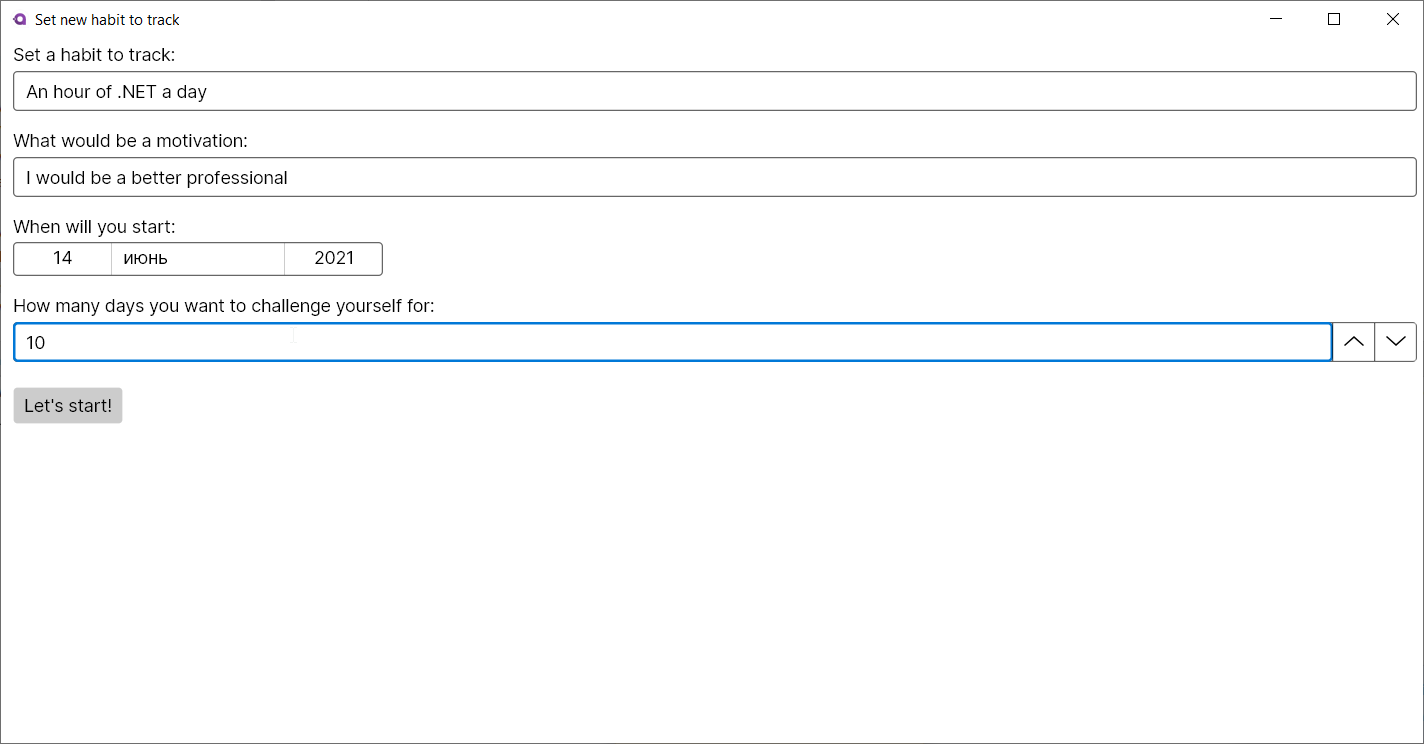
Нам потребуется всего несколько несложных экранов.

## Задание привычки

Если в системе не существует ни одной привычки, по которой ведется активный трекинг, приложение должно вывести экран ее задания.



Пользователь заполняет необходимые поля, чтобы начать трекинг.



Обратите внимание на входные данные: все поля обязательны, количество дней не должно быть отрицательным. Кнопка “старт” должна вести на следующий экран.

## Трекинг

Если существует заданная для трекинга привычка (или она была заполнена на шаге ранее), пользователь должен увидеть экран трекинга. Тут он имеет возможность отметить каждый из дней.



Здесь важно обрабатывать каждую отметку (событие *Checked* каждого объекта *CheckBox*), чтобы проставлять и сохранять ее в модели. Не забудьте проверить, если трекинг закончен!

Приведенные выше скрины работы программы помогут вам сориентироваться, что именно нужно реализовать. Обязательно воспользуйтесь [официальной документацией](https://docs.avaloniaui.net/) AvaloniaUI - там вы найдете описание всех необходимых компонентов, [основы](https://docs.avaloniaui.net/guides/basics) технологии и [туториалы](https://docs.avaloniaui.net/tutorials/todo-list-app) двух простых проектов, которые помогут вам понять и усвоить базовые принципы разработки. Вдобавок почитайте о том, что есть этот [DataContext](https://rachel53461.wordpress.com/2012/07/14/what-is-this-datacontext-you-speak-of/).

# 02. Хранение данных

Осталось всего ничего: сделать так, чтобы данные сохранялись не только в памяти в процессе работы приложения. Пусть каждый запуск приложения работает с актуальным и сохраненным набором данных. Для этого нам понадобится **база данных** - возьмем простейший вариант и будем хранить все в одном файле с помощью **SQLite**.

Дни прямого обращения к базам данных и прописывания непосредственно SQL-запросов из приложения прошли. Разработка становится все более абстрактной и высокоуровневой, и для доступа к данным чаще всего используются различные **ORM**. Это технология, позволяющая общаться с базами данных в контексте объектов ООП, а не таблиц данных и SQL-запросов к ним, по сути позволяя проецировать данные из базы прямо в объекты приложения (и наоборот).

.NET приложения используют **Entity Framework** и самый распространенный подход **Code First**. Что это значит?

Это значит, что имея в приложении описание нашей предметной модели в виде классов *Habit* и *HabitCheck,* мы можем настроить его так, чтобы база данных и соответствующие таблицы сами создавались на основе нашего кода. И при любом изменении кода в них, **схема БД** будет обновляться с помощью [механизма миграций](https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli), встроенного в технологию **Entity Framework**.

Далее мы можем рассматривать данные из базы данных как коллекции соответствующих им объектов и обращаться к ним при помощи LINQ запросов.

Начнем с того, что разделим код на два проекта: *rush00.App* и *rush00.Data.* Первый будет содержать всю логику отображения нашего приложения, второй - предметные модели. Чтобы проверить себя, задайтесь вопросом: что будет, если я решу использовать другой способ ввода и вывода информации, и вместо desktop приложения это будет веб-приложение или консольное приложение?

При таком разделении вы легко сможете переиспользовать проект, реализующий основную логику, и просто подключить к нему другое приложение с новой точкой входа.

## Рекомендованная структура проекта

rush00.App

Assets/

...

Models/

...

ViewModels/

...

Views/

...

App.axaml

Program.cs

ViewLocator.cs

rush00.Data

Migrations/

...

Models/

Habit.cs

HabitCheck.cs

HabitDbContext.cs

Подключите с помощью **nuget-менеджера** пакеты **Microsoft.EntityFrameworkCore.Tools** и **Microsoft.EntityFrameworkCore.Sqlite**.

Для работы с данными создайте класс *HabitDbContext* и унаследуйтесь от Microsoft.EntityFrameworkCore.DbContext. Это будет [основным сервисом](https://www.entityframeworktutorial.net/efcore/entity-framework-core-dbcontext.aspx) для работы с объектами из базы. В *HabitDbContext* будут две наших коллекции:

*DbSet<Habit> Habits*

*DbSet<HabitCheck> HabitChecks*

Чтобы использовать контекст доступа к данным с базой SQLite, добавьте в *HabitDbContext*:

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlite("Filename=habits.db");

}

Теперь [настройте миграции](https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli), для этого достаточно воспользоваться командой:

dotnet ef migrations add {MigrationName}

В вашем проекте должна появиться папка с миграциями, проверьте, какой код был сгенерирован и какие таблицы будут созданы с его помощью. Не забудьте добавить объектам *Habit* и *HabitCheck* поля *Id!*

Далее созданные миграции нужно “применить” к нашей базе с помощью команды:

dotnet ef database update

Или же добавить в App.axaml.cs код, который будет проверять и накатывать миграции нашего контекста при запуске приложения:

using (var context = new HabitDbContext())

{

context.Database.EnsureCreated();

}

Обратите внимание на конструкцию **using**.

Теперь, когда база существует, нам достаточно использовать экземпляр класса *HabitDbContext* и обращаться к коллекциям его объектов *Habits* и *HabitChecks* с помощью методов LINQ.

К примеру получение актуального трекинга привычки будет выглядеть как:

using (var context = new HabitDbContext())

{

var habit = \_dbContext.Habits

.Include(x => x.HabitChecks)

.FirstOrDefault(x => !x.IsFinished);

}

Обратите внимание на [метод Include](https://docs.microsoft.com/en-us/ef/ef6/querying/related-data) - он позволяет загрузить связанные данные по внешнему ключу.

Если вы выполняете сохранение в базу данных с помощью методов Add или редактируя поля объектов из базы, не забудьте сохранить изменения с помощью [метода SaveChanges()](https://docs.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.dbcontext.savechanges?view=efcore-5.0).

Подключите БД в ваш код так, чтобы:

* При запуске приложения проходила проверка, есть ли в БД привычка с актуальным трекингом
* Если таковой нет, привычка создавалась в БД, для нее создавалась коллекция *HabitChecks* для всех дат трекинга.
* Если таковая существует (или была только создана), экран с датами и отметками отображался для актуальной информации из БД.
* Отметка для даты в трекинге записывалась в БД.
* Отметка о завершенности записывалась в БД.

# Бонус. Ты молодец!

Бонусным заданием сделайте так, чтобы при окончании трекинга привычки (IsFinished == true) выводилось поздравительное сообщение с количеством выполненных дней и текстом мотивации.



## Самопроверка

* При первом запуске приложение предлагает заполнить информацию для создания трекинга привычки.
* При повторном запуске приложение отображает список дат для созданной привычки.
* Для созданной привычки с отмеченными днями повторный запуск отображает их же.