Темы урока

[**Организация слоя данных чат-бота как отдельного сервиса Web API**](#_r8dwlw946bhp) **1**

[Постановка задачи](#_rx8aay36p0sw) 1

[Что мы хотим сделать?](#_pmdoin3hoia5) 1

[Как мы будем это делать?](#_tgqfprb95as) 1

[Подготовка среды](#_tpdi10gk1whu) 2

[Оставляем в IReminderStorage только то, что нужно](#_7qcdo6iqj1ry) 2

[Меняем структуру папок](#_v8sxzbhygy67) 2

[Добавляем работу с конфигурацией](#_44iwc4vnxvuo) 3

[Добавляем возможность работы бота через прокси](#_hhg23e409y7x) 3

[Пример использования Moq для тестов доменной логики](#_2zhzlzwkdzdy) 3

[**Разработка**](#_x996uckzk9uw) **4**

[Reminder.Storage.WebApi](#_qvef9nbefnnh) 4

[Reminder.Storage.WebApi.Client](#_3kliqkwpr42g) 5

[Reminder.Storage.WebApi.Core](#_d4stfex29qcc) 5

[**Возможности для обсуждения архитектуры**](#_7sjefor46r64) **5**

[**Домашнее задание**](#_xliovyhyekrq) **5**

Вообще всё нужно будет делать максимально быстро — времени будет очень-очень “в обрез”.

# Организация слоя данных чат-бота как отдельного сервиса Web API

## Постановка задачи

### Что мы хотим сделать?

Мы хотим обеспечить сервисный доступ к списку напоминаний через Web API. Т.е., иметь возможность смотреть список существующих напоминаний, добавлять новые напоминания, удалять ещё не завершенные напоминания.

### Как мы будем это делать?

Если бы у нас была реальная база данных (т.е. уже был бы выделен единственный сервис с абсолютной адресацией), мы могли бы написать стороннее Web API приложение, которое бы ссылалось на Reminder.Storage.Core и Reminder.Storage.Sql, например. Иными словами, мы смогли бы себе позволить иметь сборку Reminder.Storage.Sql в двух независимых проектах, так как она просто обеспечивает доступ к базе данных. База при этом единственна.

Однако, у нас есть просто Reminder.Storage.InMemory, данные хранятся непосредственно в сборке Reminder.Storage.InMemory, а значит, если мы её продублируем для Web API, у него будет просто свой собственный набор напоминаний, никак не связанных с данными реального приложения чат-бота.

Поэтому нам придется обеспечить единственность Reminder.Storage.InMemory и выставлять её через Web API для как для пользователей Web API, так и для приложения.

## Подготовка среды

Теперь сделаем несколько подготовительных шагов.

### Оставляем в IReminderStorage только то, что нужно

Закомментируем в интерфейсе IReminderStorage (сборка Reminder.Storage.Core) всё, что не требуется непосредственно для работы нашего приложения. Поскольку сегодня работы у нас много, мы сконцентрируемся на функционально необходимых вещах, остальные члены можно будет реализовать и самостоятельно дома.   
  
В итоге, в интерфейсе у нас останутся только следующие методы:

* ReminderItem Get(Guid id);  
  *\* он не столько нужен функционально, однако он потребуется для Web API как часть ответа 201 - Created (там возвращается заголовок Location, который имеет значение URL, по которому доступен вновь созданный ресурс).*
* void Add(ReminderItem reminder);
* List<ReminderItem> Get(ReminderItemStatus status);
* void UpdateStatus(IEnumerable<Guid> ids, ReminderItemStatus status);
* void UpdateStatus(Guid id, ReminderItemStatus status);

Компилируем, убеждаемся, что мы не “отрезали” ничего лишнего :)

### Меняем структуру папок

Создадим дополнительный уровень структуры для ещё одного солюшена.

Дело в том, что мы в какой-то момент захотим запустить в отладке и наше приложение и наш Web API. А в солюшене может быть только один запущенный проект в один момент времени.

Таким образом, нам необходимо иметь 2 файла солюшена. И чтобы они красиво располагались — каждый в своей папке *(ну… на самом деле, будет “почти красиво”: будет небольшое пересечение проектов, которого, впрочем в реальной жизни бы не было так как мы пользовались бы NuGet-пакетами вместо проектных зависимостей; я расскажу подробнее, когда мы до этого доберёмся)* — мы разделим проекты по двум солюшенам — каждый набор будет лежать в отдельной папке:

* Папка Reminder.App — для проектов солюшена Reminder.App.sln основного консольного приложения
* Папке Reminder.Storage — для проектов солюшена Web API, но поскольку у нас там только Reminder.Storage.\* проекты, солюшн назовём Reminder.Storage.sln.

### Добавляем работу с конфигурацией

Для этого добавляем три NuGet-пакета:

* Microsoft.Extensions.Configuration
* Microsoft.Extensions.Configuration.FileExtensions
* Microsoft.Extensions.Configuration.Json

И немного кода самое начала метода Main класса Program сборки.

### Добавляем возможность работы бота через прокси

Добавляем в конструкторы классов TelegramReminderSender и TelegramReminderReceiver вторым параметром IWebProxy. В сборку ReminderApp добавляется в зависимости NuGet-пакет HttpToSocks5Proxy, в котором присутствует реализация IWebProxy с функциональностью SOCKS5 прокси, необходимой телеграм-боту.

### Пример использования Moq для тестов доменной логики

Убираем из зависимостей Reminder.Domain ссылку на Reminder.Storage.InMemory проект.

Восстанавливаем наш единственный единственный тест с использованием мока IReminderStiorage.

Можно показать, как работает Setup и Callback/Returns, но лучше уже в самом конце.

# 

# Разработка

Движемся итеративно метод за методом:

* Cначала GetReminder в Web API
* Затем Get метод в библиотеке клиента Web API
* Смотрим на результат в наскоро набросанном консольном приложении для тестирования клиента (и только в самом конце мигрируем в приложении телеграм-бота)
* Затем переходим к методу Add и далее по списку в порядке, указанном ниже

Стремиться нужно к результату тут: <https://github.com/ago-cs/cs-course-q3/tree/master/Lessons/25/ClassWork/Final>.

Вся работа у меня заняла плотных 2 урока (8 часов) в режиме “я пишу и объясняю, они пишут у себя”.

На первом уроке мы остановились на реализации двух методов получения и одного метода создания напоминаний в клиентской библиотеке. Проверяли всё на тестовом консольном приложении.

## Reminder.Storage.WebApi

Эта сборка — Web API к нашему хранилищу.

Здесь всё должно идти гладко, так как именно это мы и делали на прошлом занятии.

Рекомендуемая последовательность реализации методов:

* [HttpGet("{id}", Name = "GetReminder")]  
  public IActionResult GetReminder(Guid id)  
  *\* он не столько нужен функционально, однако он потребуется для Web API как часть ответа 201 - Created (там возвращается заголовок Location, который имеет значение URL, по которому доступен вновь созданный ресурс).*
* [HttpPost]  
  public IActionResult CreateReminder(  
   [FromBody] ReminderItemCreateModel reminder);
* [HttpGet]  
  List<ReminderItem> GetReminder(  
   [FromQuery(Name = "[filter]status")] int status = -1,);
* [HttpPatch("{id}")]  
  void UpdateReminderStatus(  
   Guid id,  
   [FromBody] JsonPatchDocument<ReminderItemUpdateModel> patchDocument);
* [HttpPatch]  
  void UpdateRemindesrStatus(  
   [FromBody] ReminderItemsUpdateModel reminderItemsUpdateModel);

## Reminder.Storage.WebApi.Client

Эта сборка — клиентская библиотека упрощающая работу с нашим Web API.

## Reminder.Storage.WebApi.Core

Сюда в процессе написания перенесётся общий код между первыми двумя сборками.

# Возможности для обсуждения архитектуры

На самом деле, вопрос спорный, нужно ли выделять модели контроллера нашего Web API (Reminder.Storage.WebApi) в отдельную сборку.

В данном конкретном случае, библиотека клиента Reminder.Storage.WebApi.Client ссылается 2 библиотеки:

* Reminder.Storage.WebApi.Core
* Reminder.Storage.Core

Если разместить классы моделей контроллера Reminder.Storage.WebApi в библиотеке Reminder.Storage.Core. Это уменьшило бы количество библиотек, которые необходимо “тащить за собой” клиенту. Однако, самому приложению Reminder.App классы моделей контроллера Reminder.Storage.WebApi не нужны.

В связи с этим, я решил вынести классы моделей контроллера Reminder.Storage.WebApi в библиотеку Reminder.Storage.WebApi.Core. Однако, надо понимать, что “правильного” ответа здесь нет, и большой проблемы не будет, если все служебные классы, которые нужны более, чем одной внешней сборке, объединить в Reminder.Storage.Core.

# Домашнее задание

Постараться доделать методы API и поддержать их в клиентской библиотеке.

*Я давал его как “Д/з без оценки”, чтобы попытались сами доделать. На следующем уроке мы будем всё это доделывать вместе.*