Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра комплексной и информационной безопасности
электронно-вычислительных систем (КИБЭВС)

# ОСНОВЫ ПРОГРАММИРОВАНИЯ

**Составители: Б.С. Лодонова, С.С. Харченко**

*Учебно-методическое пособие по курсовой работе*

В-Спектр
Томск, 2026

**УДК 004.42**
**ББК 32.973**
**Л 70**

Л 70 **Лодонова Б.С.** Основы программирования: учеб.-метод. пособие по курсовой работе / Б.С. Лодонова, С.С. Харченко. – Томск: ТУСУР, В-Спектр (ИП В.М. Бочкарева), 2026. – 90 с.
ISBN 978-5-902958-54-3

Данное методическое руководство содержит описание курсовой работы по дисциплине «Основы программирования» для студентов, обучающихся по направлению информационной безопасности. Руководство содержит теоретические выкладки и методические указания по выполнению работы.

УДК 004.42
ББК 32.973

*Одобрено на заседании кафедры КИБЭВС*
*протокол № 6 от 17 февраля 2026 г.*

**ISBN 978-5-902958-54-3**

© Б.С. Лодонова,
С.С. Харченко, 2026
© ТУСУР, 2026

## Содержание

ВВЕДЕНИЕ........................................................................................................ 4
1. ТЕОРЕТИЧЕСКИЕ ВЫКЛАДКИ ............................................................. 5
1.1. REST API ........................................................................................... 5
1.2. Реализация REST. Клиент-серверное взаимодействие...................... 8
1.3. JSON.....................................................................................................14
1.4. cURL.....................................................................................................15
1.5. NuGet....................................................................................................21
1.6. SQLite...................................................................................................24
1.7. Варианты авторизации на сервере.....................................................26
2. ХОД РАБОТЫ ..........................................................................................34
2.1. Установка начальных пакетов. Ubuntu ...............................................34
2.2. Установка начальных пакетов. Astra Linux ........................................39
2.3. Установка VS Code ..............................................................................44
2.4. Установка расширений C# для VSCode .............................................47
2.5. Создание первого серверного WEBApplication..................................48
2.6. Postman...................................................................................................51
2.7 NuGET.....................................................................................................51
2.8. Работа с SQLite......................................................................................55
2.9. Модификация WEBApplication.............................................................59
2.10. Реализация авторизации. Логин и пароль..........................................63
2.11. Реализация авторизации. Токен.........................................................66
2.12. Работа с SQLite....................................................................................70
2.13. Создание консольного приложения клиента.....................................72
2.14. Пример реализации клиент-серверного приложения ......................74
Литература ......................................................................................................89

## ВВЕДЕНИЕ

Цель дисциплины «Основы программирования» – научить студентов строить алгоритмы и реализовывать их на компьютере в виде программ. Решать различные задачи по обработке информации и моделированию, применяемые при разработке программных и программно-аппаратных компонентов защищенных автоматизированных систем.

В результате изучения дисциплины обучающийся должен:

– знать язык программирования высокого уровня.

– уметь проектировать и кодировать алгоритмы с соблюдением требований к качественному стилю программирования; реализовывать основные структуры данных и базовые алгоритмы средствами языков программирования.

– владеть навыками разработки, документирования, тестирования и отладки программного обеспечения в соответствии с современными технологиями и методами программирования; навыками разработки программной документации; навыками программирования с использованием эффективных реализаций структур данных и алгоритмов.

Указанные задачи частично могут быть решены с помощью данного учебного пособия.

# 1. ТЕОРЕТИЧЕСКИЕ ВЫКЛАДКИ

**API** (Application Programming Interface) – это набор правил и протоколов, который позволяет различным программам взаимодействовать друг с другом. API определяет способы, как различные компоненты программного обеспечения могут общаться друг с другом, обмениваться данными и выполнять определенные функции. API может быть предоставлен разработчиками программного обеспечения для использования другими приложениями или сервисами для интеграции функциональности без необходимости знать внутреннюю реализацию.

Несколько примеров API, вероятно, знакомых вам:

1. Google Maps API: Позволяет разработчикам интегрировать картографические данные и функциональность Google Maps в свои приложения.

2. Twitter API: Предоставляет доступ к данным Twitter, позволяя разработчикам создавать приложения, интегрированные с твиттером.

3. GitHub API: Предоставляет доступ к репозиториям, пользователям, организациям и другим данным на GitHub.

4. OpenWeatherMap API: Предоставляет доступ к погодным данным и прогнозам погоды для различных географических местоположений.

5. YouTube Data API: Позволяет получить доступ к данным о видео, каналах, комментариях и другой информации на YouTube.

6. VK API (ВКонтакте API): это набор программных интерфейсов и методов, предоставляемых социальной сетью ВКонтакте для разработчиков. Этот API позволяет разработчикам создавать приложения, интегрированные с ВКонтакте, и взаимодействовать с различными функциями и данными платформы.

Использование сторонних API позволяет существенно снизить время разработки за счет использования готовых решений. Например, интегрированные в новостных лентах видеоролики с сервиса Youtube – один из вариантов использования сторонними разработчиками YouTube Data API – позволяет расширить тип контента и увеличить охваты аудитории без необходимости разработки своего видеохостинга.

В работе вам надо будет использовать REST API для осуществления клиент-серверного взаимодействия. REST API представляет собой архитектурный стиль разработки или, если говорить точнее, рекомендацию по стандартизации программных интерфейсов клиент-серверной архитектуры ПО.

## 1.1. REST API

Слово **REST** [1] – акроним от Representational State Transfer, что переводится на русский как «передача состояния представления», «передача

репрезентативного состояния» или «передача самоописываемого состояния». Таким образом, можно сформировать основные требования к проектированию согласно REST:

1. Клиент-серверная модель (client-server model);
2. Отсутствие состояния (statelessness);
3. Кэширование (cacheability);
4. Единообразие интерфейса (uniform interface);
5. Многоуровневая система (layered system);
6. Код по требованию (code on demand) – необязательно.

Подробнее опишем каждое из требований.

**Клиент-серверная модель** (рис. 1.1). Первое требование ожидает от разработчика архитектуры системы такого типа, что готовое ПО делится на 2 основные части – часть с работой клиентской части ПО и ПО сервера. Причем взаимодействие между этими частями происходит на основе запросов клиента и ответов на эти запросы сервера.

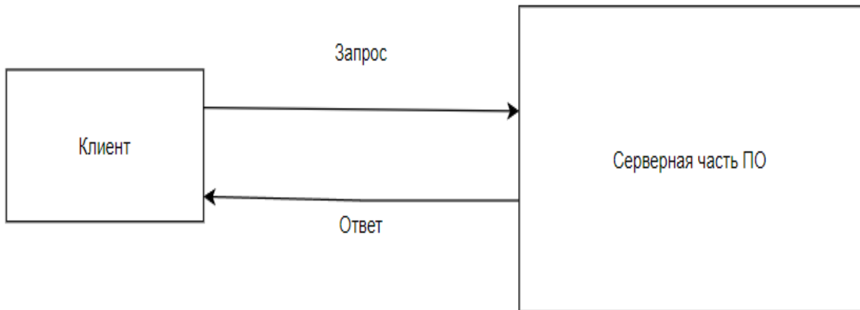


Рис. 1.1. Общий вид архитектуры REST API

При этом серверная часть может быть поделена на отдельные модули, например, модуль взаимодействия с БД, модуль работы с сетевым взаимодействием, ПО логики работы и т.д. Главное в этом всем то, что клиентской части абсолютно все равно, какие изменения произошли с ПО сервера, ответ будет получен в том виде, который клиентская часть ПО способна обработать. Причем абсолютно не важно, какие именно ресурсы получит в ответе клиент – простой ответ, файлы или код ошибки.

Для вас наиболее показательным примером могут послужить мессенджеры и соцсети. Например, ВКонтакте, WhatApp, Telegram и пр. В ВКонтакте используют серверную часть для хранения данных ваших аккаунтов, видеохостинга и прочих своих возможностей, при этом абсолютно не важно, с какого устройства будет осуществлено подключение, с мобильной версии ВК или браузерной, доступ ко всем ресурсам ВК будет у поль-

зователя одинаков. При этом, при внедрении новых возможностей (из самых явных можно назвать донаты, платные подписки, VKpay) функционал серверной части был в равной степени доступен на всех платформах. Это стало возможно как раз-таки за счет общего API этих сервисов.

**Отсутствие состояния.** Согласно REST, данные ваших сессий на серверной части ПО сохраняться не будут, потому что это слишком ресурсозатратно по памяти. Клиентов может быть много, а сервер один. Потому каждый запрос от клиента – новый ответ от сервера без сохранения результата этого запроса на стороне сервера. И серверу надо просто отвечать на запросы по мере их поступления вне зависимости от того, от кого они поступили.

**Кеширование.** Для объяснения этого необходимо вспомнить, что такое кеш/кэш (далее кеш). **Кеш** – это сохраненные на вашем устройстве файлы браузера, позволяющие быстро воспроизвести последний сеанс посещения той или иной страницы. При очистке истории браузера вы можете удалить и эти данные вместе с паролями и прочими данными.

Как вы уже знаете, многие сервисы и сайты используют сохранение кеша для быстроты взаимодействия клиента с сервером. Каждый раз пересобирать ресурсы в ответ на запрос – дело неблагодарное и затратное, особенно если на клиенте сохранено кое-что. Так что мешает запросить только новую часть данных, а, например, информацию о верстке html-страницы, взять из кеша? В этом случае нагрузка на систему будет снижена за счет уменьшения «размера» ответа. И предыдущий принцип также выполняется, поскольку даже при сохранении кеша все данные сессии сохраняются только на клиентском устройстве, и после завершения сеанса с серверной части невозможно понять, что делал клиент.

Тем не менее, в данном вопросе следует быть осторожнее и без фанатизма относиться к кешированию. Все подряд сохранять на устройстве клиента смысла зачастую не имеет, поскольку не вся информация может быть актуальна к следующему сеансу. Например, для соцсетей это может быть данные о переписке. Новое сообщение в чате должно быть подгружено во время нового сеанса, потому кешировать сообщения не стоит.

Стоит отметить, что кеширование осуществляется не только на клиенте, но и на сервере. Например, кешируются результаты однотипных запросов к БД, чтобы ускорить быстродействие системы.

**Единообразие интерфейса.** Этот принцип требует следующего: не важно, что именно вы запрашиваете и когда, ответ должен быть единообразным в любой момент времени. Даже при расширении функционала со стороны сервера.

Например, те же самые мессенджеры. В браузерной версии клиент в ответ на запрос получит html-страницу, даже если в мессенджере интегри-

руют возможность видеохостинга. Просто новые ресурсы можно будет реализовать через гиперссылки.

**Многоуровневая система** (рис. 1.2). Этот принцип больше относится к серверной части ПО, но также может быть реализован на клиенте. Суть его проста: каждый уровень взаимодействует только с соседями, т.е. использует их API, при этом он не должен взаимодействовать с модулями, находящимися через уровень. Это позволяет обеспечить определенный уровень безопасности взаимодействия, поскольку инкапсулирует функциональность на одном уровне, а также за счет относительной независимости уровней друг от друга дает возможность добавлять или удалять слои из системы при условии сохранения связей между уровнями.

Никто из участников цепочки не знает всего пути, который проходит запрос, – только своих «соседей» справа и слева. Ни клиент, ни один из прокси-серверов не знает, к кому он обращается – к основному сервису или к другому прокси. В REST API это работает в обе стороны: никакие серверы (ни основные, ни прокси) не знают, кому отправляют ответ и уходит ли он куда-то дальше.

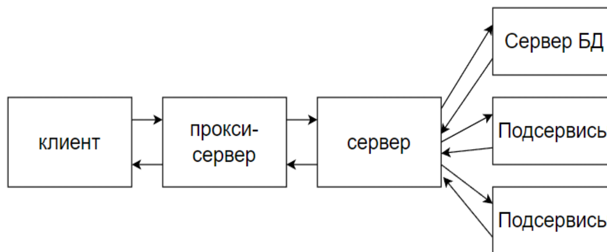


Рис. 1.2. Общая иллюстрация принципа многоуровневой системы

**Код по требованию (необязательный принцип).** Сервер отправляет код частями по мере необходимости, когда клиентскому приложению требуется дополнительная функциональность. Например, для работы в офлайн-режиме Google Docs пользователь получит дополнительный код, необходимый для работы без интернета. Этот принцип считается необязательным, особенно если такой функциональности для клиента не требуется.

## 1.2. Реализация REST. Клиент-серверное взаимодействие

REST API тесно связан с протоколом http. Как правило, архитектуру на основе REST реализуют с использованием методов http, поскольку протокол http предоставляет достаточную функциональность для этого.

**HTTP** [2] – это протокол, позволяющий получать различные ресурсы, например, HTML-документы. Протокол HTTP лежит в основе обмена данными в Интернете. HTTP является протоколом клиент-серверного

взаимодействия, что означает инициирование запросов к серверу самим получателем, обычно веб-браузером (web-browser). Данные между клиентом и сервером передаются с помощью HTTP-сообщений, содержащих команды к серверу. Сообщения, отправляемые между сервером и клиентом, бывают двух видов:

1) Запросы (HTTP Requests) – сообщения, которые отправляются клиентами на сервер. В них указывается нужный метод и URL ресурса, с которым они хотят взаимодействовать.

2) Ответы (HTTP Responses) – сообщения, которые сервер отправляет в ответ на клиентский запрос.

Наиболее распространенными методами, используемыми в REST API, являются GET, POST, PUT, PATCH и DELETE. Вот краткий обзор каждого метода:

**GET** – получение данных с сервера. Этот метод используется для получения ресурса с сервера.

**POST** создает новый ресурс на сервере. Этот метод используется для создания нового ресурса на сервере.

**PUT** – обновление существующего ресурса на сервере. Этот метод используется для обновления существующего ресурса на сервере.

**PATCH** – частичное обновление существующего ресурса на сервере. Этот метод используется для обновления части существующего ресурса на сервере.

**DELETE** удаляет ресурс с сервера. Этот метод используется для удаления ресурса с сервера.

Пример реального запроса был получен с помощью утилиты WireShark (рис. 1.3).

Синим выделена стартовая строка запроса. Наш HTTP-запрос выполняется методом GET, URI (Uniform Resource Identifier, унифицированный идентификатор ресурса) [3] указывает на необходимый ресурс, а поле «версия» HTTP содержит значение 1.1. Ниже WireShark представляет структуру запроса: метод запроса, URL запроса и версия http. Ответ сервера также был зафиксирован (рис. 1.4).

Также синим выделена строка ответа, и представление структуры ответа в интерпретации WireShark: версия http, код статуса и статус ответа, коды http-ответов.

В запрос могут быть добавлены дополнительные заголовки (опционально). На рис. 1.3 как раз вы можете увидеть несколько вариантов заголовков, а именно Accept, Connection, Host и User-Agent. Заголовки зачастую используются для передачи дополнительной информации. Например, заголовок Connection определяет статус соединения после отправки запроса, нужно ли оставлять соединение действующим или стоит разорвать соединение. Connection: Keep-Alive на рис. 1.3 означает, что соединение остаётся и не завершается, позволяя выполнять последующие запросы на

тот же сервер, в то время как Connection: close указывает на потребность в разрыве соединения.

```
871 13.269458    192.168.3.1      192.168.3.12     DNS    213 Standard query response 0xda24 A mobile.events.data.microsoft.com CNAME
813 12.811279    192.168.3.12     192.229.221.95   HTTP   290 GET /MFEwTzBNMEswSTAJBgUrDgMCGgUABBSAUQYBMq2awn1Rh6Doh%2FsBYgFV7gQUA95Q
780 12.556970    192.168.3.50     224.0.0.251      MDNS   500 Standard query response 0x0000 TXT, cache flush PTR _yandexio._tcp.loca
781 12.557106    fe80::82b6:55ff:fe6… ff02::fb     MDNS   520 Standard query response 0x0000 TXT, cache flush PTR _yandexio._tcp.loca
815 12.900070    192.229.221.95   192.168.3.12     OCSP   791 Response

> Frame 813: 290 bytes on wire (2320 bits), 290 bytes captured (2320 bits) on interface \Device\NPF_{D10F9781-1A55-4726-BE64-A051344CFFA8}, id 0
v Ethernet II, Src: GigaByteTech_a7:bc:f7 (d8:5e:d3:a7:bc:f7), Dst: HuaweiDevice_e0:59:8f (78:c5:f8:e0:59:8f)
  > Destination: HuaweiDevice_e0:59:8f (78:c5:f8:e0:59:8f)
  > Source: GigaByteTech_a7:bc:f7 (d8:5e:d3:a7:bc:f7)
    Type: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 192.168.3.12, Dst: 192.229.221.95
> Transmission Control Protocol, Src Port: 62342, Dst Port: 80, Seq: 1, Ack: 1, Len: 236
v Hypertext Transfer Protocol
  v GET /MFEwTzBNMEswSTAJBgUrDgMCGgUABBSAUQYBMq2awn1Rh6Doh%2FsBYgFV7gQUA95QNVbRTLtm8KPiGxvDl7I90VUCEAJ0LqoXyo4hxxe7H%2Fz9DKA%3D HTTP/1.1\r\n
    > [Expert Info (Chat/Sequence): GET /MFEwTzBNMEswSTAJBgUrDgMCGgUABBSAUQYBMq2awn1Rh6Doh%2FsBYgFV7gQUA95QNVbRTLtm8KPiGxvDl7I90VUCEAJ0LqoXyo4hxxe7
      Request Method: GET
      Request URI: /MFEwTzBNMEswSTAJBgUrDgMCGgUABBSAUQYBMq2awn1Rh6Doh%2FsBYgFV7gQUA95QNVbRTLtm8KPiGxvDl7I90VUCEAJ0LqoXyo4hxxe7H%2Fz9DKA%3D
      Request Version: HTTP/1.1
    Connection: Keep-Alive\r\n
    Accept: */*\r\n
    User-Agent: Microsoft-CryptoAPI/10.0\r\n
    Host: ocsp.digicert.com\r\n
    \r\n
    [Full request URI: http://ocsp.digicert.com/MFEwTzBNMEswSTAJBgUrDgMCGgUABBSAUQYBMq2awn1Rh6Doh%2FsBYgFV7gQUA95QNVbRTLtm8KPiGxvDl7I90VUCEAJ0LqoXyo
    [HTTP request 1/1]
    [Response in frame: 815]
```

Рис. 1.3. Пример пакета с http-запросом

Рис. 1.4. Пример ответа на http-запрос

Весь список заголовков достаточно обширен, и в рамках этого курса рассматриваться не будет. Стоит отметить, что http-протокол – это стандарт, и в разработке с использованием http-запросов следует строго придерживаться правил этого протокола.

Тело HTTP-запроса (Request Body) используется для передачи данных от клиента к серверу. Оно обычно присутствует в запросах, которые создают или обновляют ресурсы на сервере, таких как POST, PUT, PATCH и DELETE. Данные в теле запроса могут быть представлены в разных форматах в зависимости от типа содержимого и цели запроса.

Основные форматы данных в теле HTTP-запроса:

**Форма URL-кодированная** (application/x-www-form-urlencoded). Данные отправляются в формате ключ-значение и кодируются таким образом, что специальные символы (например, пробелы и символы &, =) заменяются соответствующими кодами. Чаще всего используется при отправке HTML-форм. Пример данных на рис. 1.5.

```
username=johndoe&password=secure123
```

Рис. 1.5. Пример URL-кодированной формы

Многочастная форма данных (multipart/form-data). Используется для отправки файлов и бинарных данных вместе с текстовыми полями. Обычно применяется при загрузке файлов через формы, например, при добавлении аватара или документа. Каждая часть данных отделена границей (boundary), и каждый блок имеет собственные заголовки. Пример данных на рис. 1.6.

```
--boundary
Content-Disposition: form-data; name="username"

johndoe
--boundary
Content-Disposition: form-data; name="file"; filename="example.jpg"
Content-Type: image/jpeg

(бинарные данные изображения)
--boundary--
```

Рис. 1.6. Пример многочастной формы данных

**JSON** (application/json). Один из самых популярных форматов для передачи данных между клиентом и сервером в веб-приложениях, особенно при работе с API. Данные представляются в виде ключ-значение, подобно объектам в JavaScript. Пример данных на рис. 1.7.

```
{
  "username": "johndoe",
  "email": "john.doe@example.com",
  "age": 25
}
```

Рис. 1.7. Пример данных формата JSON

**XML** (application/xml или text/xml). Формат, часто используемый в старых системах и некоторых API, особенно тех, которые требуют сложных структур данных. Данные описываются с использованием элементов и атрибутов в XML-дереве. Пример данных на рис. 1.8.

```
<user>
  <username>johndoe</username>
  <email>john.doe@example.com</email>
  <age>25</age>
</user>
```

Рис. 1.8. Пример данных формата XML

**Текстовые данные** (text/plain). Используется для отправки простого текста без какой-либо структуры. Может применяться для отправки небольших заметок или сообщений.

**Двоичные данные** (application/octet-stream). Используется для передачи двоичных файлов в «сырых» данных, таких как файлы изображений, видео или исполняемые файлы. Чаще всего используется в API для загрузки файлов или передачи нестандартных данных.

Как выбирается тип данных для тела запроса?

Тип данных в теле запроса указывается в заголовке Content-Type. Например:

- для JSON – Content-Type: application/json;
- для формы – Content-Type: application/x-www-form-urlencoded;
- для загрузки файлов – Content-Type: multipart/form-data.

Выбор формата зависит от типа данных, которые передаются, и требований сервера, обрабатывающего запрос. JSON стал стандартом де-факто для современных веб-приложений благодаря его легкости и удобству работы.

Поговорим о той части ответа на HTTP-запрос, который вы часто видели в браузере, а именно – код состояния HTTP. Это коды, по которым можно бы быстро определить каким был результат вашего запроса и не лезть в тело http-ответа. Самые распространенные из них:

– 200 OK – запрос успешно выполнен, а результат можно найти в теле ответа;

– 301 Moved Permanently – запрошенный ресурс перенесен по новому URL. Новый адрес документа указан в поле Location в заголовке страницы;

– 304 Not Modified – ресурс, запрошенный клиентом, остался без изменений и может быть использован из кеша клиента. В этом случае сервер не будет высылать его снова;

– 400 Bad request – сообщение о синтаксической ошибке, обнаруженной сервером;

– 403 Forbidden – запрашиваемый ресурс не может быть передан из-за ограничений в доступе (отсутствие прав доступа). Либо клиент не авторизовался на сервере, либо доступ к этому ресурсу ему передавать нельзя из-за политики безопасности на сервере;

– 404 Not Found – запрашиваемый ресурс не найден;

– 502 Bad Gateway – сервер-шлюз не получил некорректный ответ (или не получил вообще);

– 504 Gateway Timeout – нет ответа от сервера за время ожидания.

## 1.3. JSON

Далее опишем контейнер, который будет содержаться в http-запросах и который будет передаваться между клиентом и сервером. Разумеется, вы вправе придумать свой вариант отправки, хоть файлы формата txt. Однако для стандартизации и простой интеграции были придуманы форматы файлов, работа с которыми уже интегрированы в стандарты большинства языков программирования: JSON, YAML, XML и пр. В рамках курсовой работы мы сосредоточимся на первом из вышеперечисленных.

**JSON** [4] (англ. JavaScript Object Notation) – текстовый формат обмена данными, основанный на JavaScript. Но при этом формат независим от JS и может использоваться в любом языке программирования. В REST и http используется как JSON, так и XML, однако первый более удобоваримый для начинающих программистов. К тому же на данный момент он является стандартом для обмена данными в веб и мобильных приложениях.

Синтаксис JSON состоит из пар вида ключ – значение, ключ1 – значение1 и т.д.

```
{
  «Имя»: «Студент1»,
  «Курс»: 2
}
```

Выше представлен объект JSON. При работе с ним вы можете обратиться к конкретному значению не по индексу, а по ключу и получить тот же самый ответ. То есть если вы перепутаете и напишите структуру в другом порядке:

```
{
  «Курс»: 2,
  «Имя»: «Студент1»
}
```

И в коде будете работать с данными через их ключи, то ничего страшного не произойдет, вы будете получать те же результаты. В этой связи следует упомянуть и массивы JSON:

```
[
«Apple»,
«Strawberry»,
«Raspberry»
]
```

К этой структуре вам придется обращаться по индексам, и перепутанный порядок в этой связке данных будет влиять на результат. Объекты и массивы могут быть вложены друг в друга, а количество уровней вложенности не ограничено.

На рис. 1.9 вы можете заметить как просто объекты (галочки), так и массивы (квадратные скобки). Массив members содержит в себе отдельные структуры с объектами (name, age, secretIdentity) и массивами (powers).

Стоит отметить: абсолютно не важно, какого типа данные вы помещаете в JSON, их обработка целиком и полностью ложится на плечи разработчика. Например, если в представленном на рисунке 5 файле вместо Super Hero Squad написать какое-то число, то корректная обработка этого варианта должна проходить на стороне сервера.

## 1.4. cURL

В рамках курсовой работы предлагается использование cURL для взаимодействия с сервером. Для начала разберем, что такое URL.

**Uniform Resource Locator (URL)** [3]– это механизм присвоения уникального адреса ресурсу для взаимодействия с ним в сети или, собственно, сам адрес этого ресурса. Его состав следующий:

- Протокол (http, https, ftp, mailto, file).
- Имя хоста (доменное имя или IP).
- Порт (необязательно).
- Путь к странице или файлу.
- Параметры запроса (необязательно).
- Якорь (необязательно).

Основные части: протокол, домен и путь к ресурсу в рамках домена представлены на рис. 1.10.

Протокол указывает на метод передачи данных: http – протокол передачи гипертекста, ftp – протокол передачи транспорта файлов, file – протокол открытия в браузере файла с локального устройства.

Имя хоста – адрес сервера в сети. Может быть представлен в виде домена или IP-адреса. Крайне редко дополняется портом – //example.com:8080.

```
JSON

{
  "squadName": "Super hero squad",
  "homeTown": "Metro City",
  "formed": 2016,
  "secretBase": "Super tower",
  "active": true,
  "members": [
    {
      "name": "Molecule Man",
      "age": 29,
      "secretIdentity": "Dan Jukes",
      "powers": ["Radiation resistance", "Turning tiny",
"Radiation blast"]
    },
    {
      "name": "Madame Uppercut",
      "age": 39,
      "secretIdentity": "Jane Wilson",
      "powers": [
        "Million tonne punch",
        "Damage resistance",
        "Superhuman reflexes"
      ]
    },
    {
      "name": "Eternal Flame",
      "age": 1000000,
      "secretIdentity": "Unknown",
      "powers": [
        "Immortality",
        "Heat Immunity",
        "Inferno",
        "Teleportation",
        "Interdimensional travel"
      ]
    }
  ]
}
```

Рис. 1.9. Пример JSON-файла с вложенными объектами и массивами

Рис. 1.10. Простая схема URL

*Заметка:* доменное имя заменяется IP-адрес с помощью системы доменных имен.

Третья часть URL– это, собственно, путь к ресурсу, со всеми каталогами и подкаталогами, плюс параметры запроса и якорь. Пример представлен на рис. 1.11. Все, что идет после первого спецсимвола (см. рис. 1.11, решетка, отмечена галочкой) – тело запроса.

Рис. 1.11. Пример запроса с каталогом и

В URL разрешено использование буквенно-цифровых символов (A–Z, a–z, 0–9), безопасных (-, _, ., ~), специальных (!, *, ', (, )) и зарезервированных (;, /, ?, :, @, &, =, +, $, ,). Неподдерживаемые символы кодируются в процентах по их шестнадцатеричному ASCII-коду – например, %20 представляет пробел.

*Заметка*: Более подробно с правилами URL можно ознакомиться в стандарте [3].

Используются символы управления:

: (двоеточие) – для разделения схемы (например, http) от остальной части URL.

/ (косая черта) – для разделения различных частей URL, таких как путь.

? (вопросительный знак) – для начала строки запроса.

# (решётка) – для указания фрагмента (якоря) в документе.

Могут использоваться зарезервированные символы:

! * ' ( ) – в некоторых контекстах, но часто требуют кодирования.

; (точка с запятой) – в строках запроса.

@ (собачка) – для указания информации о пользователе.

& (амперсанд) – позволяет передавать несколько параметров в одном запросе.

Если вы хотите использовать эти символы в запросе, то их следует кодировать с помощью percent-encoding. Например:

Пробел кодируется как %20 или +;  : кодируется как %3A; / кодируется как %2F.

**cURL** – (client URL) – это инструмент командной строки на основе библиотеки libcurl для передачи данных с сервера и на сервер при помощи различных протоколов, в том числе HTTP, HTTPS, FTP, FTPS, IMAP,

IMAPS, POP3, POP3S, SMTP и SMTPS. Он очень популярен в сфере автоматизации и скриптов благодаря широкому диапазону функций и поддерживаемых протоколов.

Начиная с Windows 10, он является встроенным в командную строку инструментом. В Linux системах он также предустановлен, но для проверки можете в терминале прописать команду «curl –version» и проверить. Если в ответ команда не выдаст версию утилиты, то установите ее с помощью команды: sudo apt install curl.

Просмотреть весь функционал cURL можно командой «curl --help all».

Команда запроса имеет следующий вид:

curl [options] [URL].

По умолчанию формируется запрос GET (рис. 1.12).

Запрос POST имеет следующую структуру:

curl -X POST -H «[тип содержимого]» -d «[post data]» [options] [URL].

Для реализованного в ходе работы сервера запрос представлен на рис. 1.13. Запрос GET после аутентификации представлен на рис. 1.14.

Параметры, использованные в запросе:

-X – определение метода http запроса (POST, GET, DELETE и пр., по умолчанию – GET)/ не путать с -x;

-H – наполнение запроса указанными заголовками;

--location – необязательный – позволяет curl выполнять перенаправления;

--header – то же самое, что -H.

Второй запрос был взят из автоматического заполнения запросов cURL от Postman. Модифицируем его под исходный шаблон (рис. 1.15).

Ответ получен. Далее приведем примеры работы с сайтом с помощью cURL (рис. 1.16). Также можно сделать запрос на получение логин-токена (рис. 1.17, 1.18).

Следующий запрос: авторизация с вашим логином и токеном (рис. 1.19). Если токен некорректен, старый или нет файла cookies.txt, то будет получен результат, как на рис. 1.20. Если все прошло хорошо, вы получите токен (рис. 1.21) и после сможете редактировать страницы (рис. 1.22). Надо получить еще csrf-токен (рис. 1.23), но это особенность API wiki.

Отправим «Тестовая страница редактирования данных ОР» на страницу пользователя AdaraCross/SandBox, чтобы «не мусорить» в основной части Википедии (рис. 1.24). Придет ответ (рис. 1.25), и будет получен результат (рис. 1.26).

```
C:\Windows\System32>curl https://ru.wikipedia.org/wiki/%D0%A1%D1%8B%D0%BD%D0%BE%D0%B2%D1%8C%D1%8F_%D0%BC%D1%83%D0%BB%D0%
BB%D1%8B
<!DOCTYPE html>
<html class="client-nojs" lang="ru" dir="ltr">
<head>
<meta charset="UTF-8">
<title>Сыновья муллы — Википедия</title>
<script>(function(){var className="client-js";var cookie=document.cookie.match(/(?:^|; )ruwikimwclientpreferences=([^;]+
)/);if(cookie){cookie[1].split('%2C').forEach(function(pref){className=className.replace(new RegExp('(^| )'+pref.replace
(/-clientpref-\w+$|[^\w-]+/g,'')+'-clientpref-\\w+( |$)'),'$1'+pref+'$2');});}document.documentElement.className=classNa
me;}());RLCONF={"wgBreakFrames":false,"wgSeparatorTransformTable":[",\t.","  \t,"],"wgDigitTransformTable":["",""],"wgDef
```

Рис. 1.12. Запрос GET

```
    ^
balzhit@balzhit-main:~$ curl -X POST http://localhost:3000/login -H "Content-Type: application/json"
 -d '{"email":"user","password": "1234"}'
{"access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMD
A1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1lIjoidXNlciIsImV4cCI6MTcyNjU4Njg5NiwiaXNzIjoiTXlBdXRoU2VydmVyIbalba
```

Рис. 1.13. Запрос для реализованного сервера

```
balzhit@balzhit-main:~$ curl --location 'http://localhost:3000/data' --header 'Authorization: Bearer ey
JhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2Ns
YWltcy9uYW1lIjoidXNlciIsImV4cCI6MTcyNjU4NzAwNywiaXNzIjoiTXlBdXRoU2VydmVyIiwiYXVkIjoiTXlBdXRoQ2xpZW50In0
.RKfaF7rBZYT3PVR4VJt066o2xf8aA9mVZ-pYcerQvHQ'
{"Date":"2024-09-17T00:00:00+07:00","TemperatureCelsius":"Autumn, fog, 15C"}balzhit@balzhit-main:~$
```

Рис. 1.14. Запрос GET после аутентификации GET

```
balzhit@balzhit-main:~$ 0:00+http://localhost:3000/data -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsI
nR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1lIjoidXN
lciIsImV4cCI6MTcyNjU4NzAwNywiaXNzIjoiTXlBdXRoU2VydmVyIiwiYXVkIjoiTXlBdXRoQ2xpZW50In0.RKfaF7rBZYT3PVR4VJ
t066o2xf8aA9mVZ-pYcerQvHQ'
{"Date":"2024-09-17T00:00:00+07:00","TemperatureCelsius":"Autumn, rain, -8C"}balzhit@balzhit-main:~$
```

Рис. 1.15. Модифицированный запрос

```
C:\Users\Balzhit>curl -X POST "https://en.wikipedia.org/w/api.php"
 -H "Content-Type: application/x-www-form-urlencoded"
-d "action=query&list=search&srsearch=Python&format=json"
```

Рис. 1.16. Запрос POST к Википедии, которая выводит содержимое https://en.wikipedia.org/w/api.php страницы

```
C:\Users\Balzhit>curl -c cookies.txt -X POST "https://en.wikipedia.org/w/api.php"
rm-urlencoded" -d "action=query&meta=tokens&type=login&format=json"
 -H "Content-Type: application/x-www-fo
rm-urlencoded" -d "action=query&meta=tokens&type=login&format=json"
```

Рис. 1.17. Запрос на получение логин-токена

```
rm-urlencoded" -d "action=query&meta=tokens&type=login&format=json"
{"batchcomplete":"","query":{"tokens":{"logintoken":"795297...768b5274b+\\"}}}
C:\Users\Balzhit>
```

Рис. 1.18. Полученный токен

```
C:\Users\Balzhit>curl -b cookies.txt -c cookies.txt
-X POST "https://en.wikipedia.org/w/api.php" -H "Content-Type: appli
cation/x-www-form-urlencoded"
 "action=login&lgname=ВАШ_ЛОГИН&lgpassword=ВАШ_ПАРОЛЬ&lgtoken=
=ВАШ_LOGIN_TOKEN&format=json"
```

Рис. 1.19. Авторизация с логином и токеном

```
79a2996cde6f868b52a5b+\"&format=json"
{"login":{"result":"WrongToken"}}
```

Рис. 1.20. «WrongToken»

```
[1] PџPsP»CѓC‡P°PµPј login token...
 Login token PїPsP»CѓC‡PµPS: 0557f58966a7db7348f918acc64aec8268b5321d+\
```

Рис. 1.21. Полученный токен

```
C:\Users\Balzhit>curl -b cookies.txt -X POST "https://en.wikipedia.org/w/api.php"
-H "Content-Type: application/x-www-fo
rm-urlencoded" -d "action=query&meta=tokens&format=json"
```

Рис. 1.22. Использование токена

```
CSRF token: 64c1e584b4773ee23d9a242c0c56653f68b5321e+\
```

Рис. 1.23. Получение csrf-токен

```
C:\Users\Balzhit>curl -b cookies.txt -X POST "https://en.wikipedia.org/w/api.php"
-H "Content-Type: application/x-www-fo
rm-urlencoded" -d "action=edit&title=User:AdaraCross/sandbox&format=json&token=
=ВАШ_CSRF_TOKEN&text=Тестовая страница редактирования данных ОР"
```

Рис. 1.24. Отправка на страницу пользователя

```
        },
"login": {
        "result": "Success",
        "lguserid": 50199443,
        "lgusername": "AdaraCross"
    }
```

Рис. 1.25. Полученный ответ

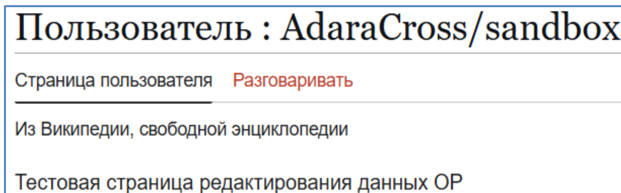


Рис. 1.26. Результат на странице пользователя

## 1.5. NuGet

Почти у каждого языка программирования есть свой (официальный) пакетный менеджер. Его задача – *регистрировать* в своем архиве *новые пакеты*, автоматически *подтягивать* выбранные разработчиком *пакеты расширений языка вместе с их зависимостями* на его рабочий инструмент, *распаковывать и устанавливать*, и, самое важное, *интегрировать* его *в проект*, модифицируя файлы настроек проекта.

Для C# разработан официальный пакетный менеджер – NuGet. С ним вы будете работать, когда будете устанавливать расширения для базы данных. NuGet для своей работы использует dotnet – виртуальная машина C#. Задачи dotnet – устанавливать, удалять и обновлять пакеты как NuGet, так и любые другие.

**Postman**. Для работы с этим расширением вам потребуется зарегистрироваться на сайте разработчика (рис. 1.27, 1.28).

После этого требуется установить расширение на VSCode и авторизоваться в вашей учетной записи. Для тестирования API вашего сервера вы будете использовать Postman. Расширение Postman в VSCode можно будет найти в левой части меню, его иконка – круговая диаграмма.

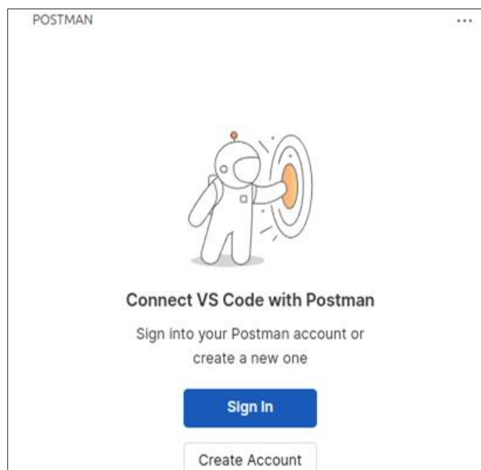


Рис. 1.27. Форма для перехода к регистрации

Create Postman account

Why sign up?

- Organize all your API development within Postman Workspaces
- Sync your Postman data across devices
- Back up your data to the Postman cloud
- It's free!

Work email

Username

Password Show

Receive product updates, news, and othe
marketing communications

Stay signed in

Verify you are human

Create Free Account

Рис. 1.28. Создание аккаунта

Сам интерфейс делится на 3 части (рис. 1.29):

1) с сохраненными коллекциями запросов, историей запросов и т.д.;
2) с запросами и их параметрами;
3) с Postman-консолью VSCode.

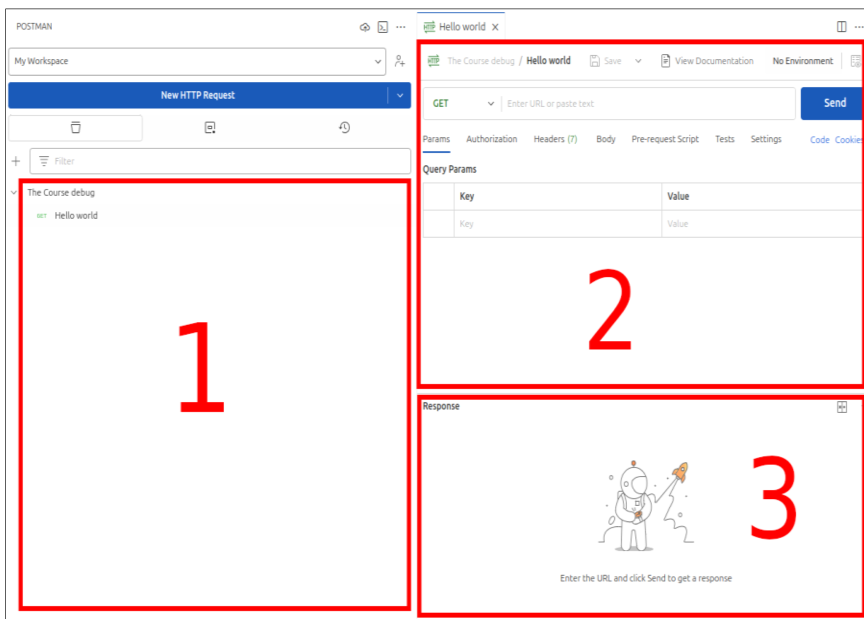

Рис. 1.29. Интерфейс Postman

Часть с формированием запроса (рис. 1.30) – ваш основной инструмент. В нем вы можете составлять запросы к API сервера, используя различные параметры и модифицируя их:

**Params** – формирует список параметров в URL, прописываемые через &;

**Authorization** – формирует запрос на авторизацию с использованием различных технологий (базовая авторизация, по токену, сертификату и пр., будет использовано для авторизации по токену);

**Headers** – формирует запрос с определенными пользователем заголовками;

**Body** – формирует запрос с определенными пользователем телом запроса (будет использовано, чтобы передать json в запросе);

**Tests** – написание скриптов на JavaScript для автоматизированного тестирования бэкэнда

**Settings** – различные настройки Postman.

В правой верхней части окна параметров есть 2 дополнительных параметра – Code и Cookies. Они показывают cURL-код и куки сформированного запроса соответственно. В данной версии Postman не может показать полный код авторизации для cURL, это функционал расширенной версии.

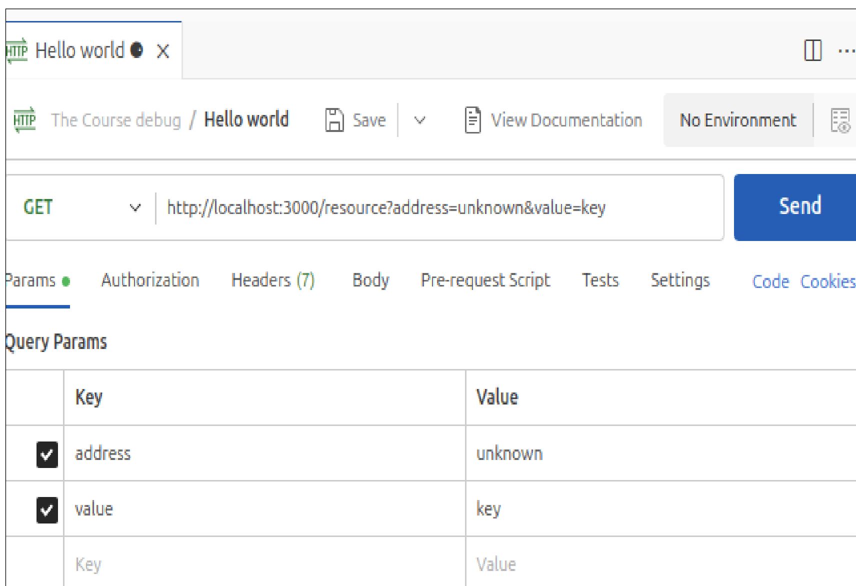


Рис. 1.30. Часть с формированием запроса

## 1.6. SQLite

База данных – неотъемлемая часть сервисов с авторизацией. В базе пользователей зачастую хранится информация об их идентификаторах – преобразованная (защищенная) связка пароля и логина. Для хранения этой информации необходим контейнер, который бы позволил бы быстро находить нужную запись и упорядочивал бы эти записи для удобства администрирования. Одним из вариантов таких БД являются реляционные БД.

Реляционные базы данных представляют собой базы данных, которые используются для хранения и предоставления доступа к взаимосвязанным элементам информации. Реляционные базы данных основаны на реляционной модели – интуитивно понятном, наглядном табличном способе представления данных. Таблицы оказываются связанными между собой за счет ключевой информации – первичный и вторичный ключ. В особенности реализации вдаваться не будем, но для дальнейшего понимания следует знать, что БД бывают не только в том виде, что будет использоваться в курсовой.

**SQLite** – реляционная встраиваемая БД. Это означает, что помимо табличного представления данных она отличается от иных БД тем, что не требует развертывания сервера, а ей достаточно 1 файла, в которой пропи-

сывается вся информация БД и библиотеки API SQLite, которая компилируется с программой. Эта версия БД очень проста в использовании и отлично подходит для ознакомления.

Но в целом не суть важно, какая именно реляционная БД будет использована. Если в названии есть SQL, значит, надо знать, что такое SQL.

**SQL** (Structured Query Language, или язык структурированных запросов) – это декларативный язык программирования, используемый ТОЛЬКО для взаимодействия с поддерживающими его БД. Используя его, не получится написать программу, но поработать с данными в базе он помогает. Как и у любого языка, у него есть свой синтаксис, с основами которого вам понадобиться ознакомиться для выполнения курсовой.

Представим, что у нас есть база данных пользователей. Для этой БД требуется создать таблицу в БД. Запрос SQL на создание таблицы будет выглядеть следующим образом:

```
CREATE TABLE user_DB(
id INT PRIMARY KEY,
email TEXT NOT NULL,
password TEXT NOT NULL).
```

В запросе указаны:

- CREATE TABLE – команда на создание таблицы;
- user_DB – название таблицы;
- id, email, password – названия столбцов;
- INT, TEXT – типы данных (TEXT – аналог string);
- NOT NULL – означает, что значение этого столбца должно быть установлено;
- PRIMARY KEY – означает, что значение этого столбца является первичным ключом.

Далее таблицу следует заполнить. Для этого выполним запрос на добавление записи в таблицу:

```
INSERT INTO user_DB (id, email, password ) VALUES (5, 'user', '1234')
```

В данном запросе:

- INSERT INTO…VALUES – запрос на внедрение данных в таблицу;
- user_DB (id, email, password ) – название таблицы и столбцов;
- (1, pasha@mail.ru', '1234') – значения столбцов в той же последовательности.

По аналогии выполняются и другие запросы. Таким образом, в таблице оказываются и другие записи (рис. 1.31).

| id | email | password |
|---|---|---|
| 1 | pasha@mail.ru | 1234 |
| 2 | ANANAS@mail.ru | jhdugazs |
| 3 | SUPERHERO2077@mail.ru | batman555 |
| 4 | tusur@mail.ru | /tuhhw4_09-oga |

Рис. 1.31. Таблица в БД

И как бы мы нашли пароль пользователя с почтой tusur@mail.ru на отличном от SQL языке? Считали бы базу user_DB, распарсили бы ее в контейнер (список, словарь – не важно) и прошлись бы по ключу email и нашли бы с почтой tusur@mail.ru и его пароль /tuhhw4_09-oga. SQL для этого нужен один запрос:

```
SELECT password FROM user_DB WHERE email=tusur@mail.ru
```

Здесь мы видим самую простую структуру запроса обработки данных:

• SELECT – выбирает указанное значение (password);

• FROM – указывает множество(а), по которому будет осуществляться выборка;

• WHERE – уточняет, по какому признаку осуществляется выбор.

Этот запрос позволяет некоторые модификации:

```
SELECT значение1, значение2, … FROM таблица1, таблица2, … WHERE (признак1 AND признак2) OR признак3
```

## 1.7. Варианты авторизации на сервере

На данный момент существует множество протоколов авторизации, часть из которых вы, не обращая на это внимания, использовали. Наверняка часть из вас использовала авторизацию через сервисы VK, Google и пр. на сайтах библиотек, модов для игр, YouTube и им подобных. Например, как вам известно, YouTube использует Indexing API от Google для авторизации пользователей через учетные записи Google.

Разумеется, в рамках курсовой работы использовать данный способ мы не будем, поскольку он требует регистрации приложения в Google API Console, что вряд ли можно сделать за выделенное на выполнение курсовой время (проверка кода, источника происхождения кода, проверка соответствия кода правилам Google, правки в соответствии с правилами Google и т.д.). Начнем ознакомление с простой авторизации по паре логин-пароль.

**Базовый способ авторизации**. Самый простой способ авторизации – передача логина и пароля серверу в запросе серверу в чистом виде. Этот способ является крайне уязвимым к атакам, потому не применяется в таком виде. Для подобной аутентификации необходимо, чтобы сервер знал логин и соотнесенный с ним пароль, и при нахождении вышеуказанной пары проводил авторизацию.

Примерный алгоритм использования такого способа авторизации показан на схеме на рис. 1.32, 1.33.

Во второй половине схемы представлен механизм авторизации после записи логина-пароля в БД сервера. В этом случае запись в БД будет выглядеть как на рис. 1.34.

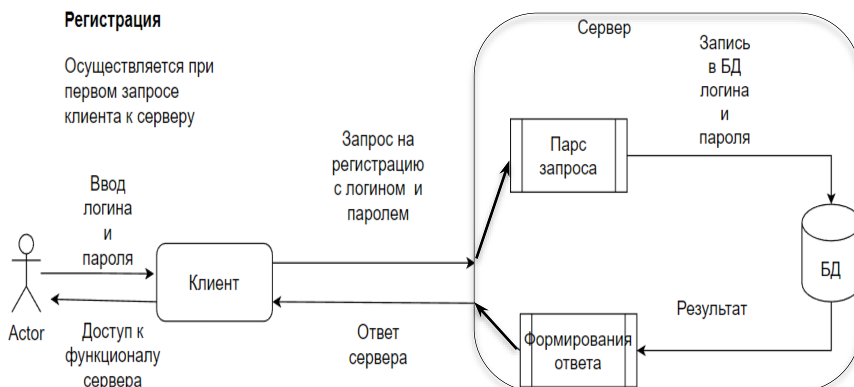


Рис. 1.32. Примерная схема регистрации по паре логин-пароль

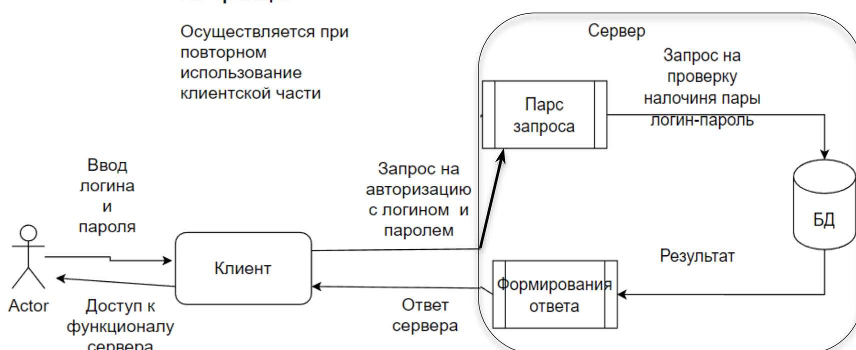


Рис. 1.33. Примерная схема авторизации по паре логин-пароль

| userid | user_name | password |
|---|---|---|
| 1 | ekim | ekim |
| 2 | lena | loveyou |

Рис. 1.34. Пример БД паролей пользователей в открытом виде

**Базовый способ авторизации с хешированием и статичной солью**. Передача логина и пароля в запросе в открытом виде является крайне нежелательным способом авторизации, поскольку любой может перехватить пакет и получить к ним доступ. Как видно на рис. 1.3, http-запросы в открытом виде передают команду и ресурс (в нашем случае – логин и пароль).

*Заметка*. В данный момент протокол http практически не используется, в основном эксплуатируется его защищенная шифрованием версия

https, но тем не менее, в рамках обучения требуется ознакомиться с исходной технологией для дальнейшего понимания.

Логика авторизации с хешированием схожа с предыдущим вариантом, только пара логин-пароль передаются в защищённом виде – в виде хэша. Вариаций передачи такого формата может быть множество: логин и хэш от пароля с солью и без; зашифрованные логин с солью и без и хэш пароля с солью и без; хэш от логина с солью и без и хэш от пароля с солью и без и т.д.

Следует разобраться, что такое хэш(хэширование) и соль.

**Соль** – это псевдослучайная последовательность данных, которую добавляют к криптоключу (паролю, логину и др.) для предотвращения декодирования информации методом перебора.

**Хэширование** – это преобразование информации с помощью однопроходных математических функций. Результатом этого преобразования является хеш. Однопроходность функции преобразования гарантирует невозможность восстановления исходных данных по полученному результату. Таким образом, защищенный хэшированием пароль невозможно восстановить по его хэшу.

Как правило, на серверных БД хранятся не пароль в чистом виде, а его хэш. В таком случае запись и в БД будут выглядеть как на рис. 1.35.

| userid | user_name | password |
|---|---|---|
| 1 | ekim | 2aee1c40199c7754da766e61452612cc |
| 2 | lena | f74a10e1d6b2f32a47b8bcb53dac5345 |

Рис. 1.35. Пример БД паролей пользователей в виде хэша

Стоит понимать, что данный тип хранения ничем не отличается от предыдущего, поскольку пароль все так же может быть перехвачен, хоть и в виде хэша. С солью дела обстоят также. Для аутентификации пользователя в БД соль должна храниться на сервере, чтобы можно было отделить ее от пароля (рис. 1.36).

| userid | user_name | password | salt |
|---|---|---|---|
| 1 | ekim | e32375ec978651577c52db4a1f46dbcb | 5&d!hg |
| 2 | lena | c9f78dcc5606a4ccaefea1e25ea90a2e | fh^4$j |

Рис. 1.36. Пример БД паролей пользователей в виде хэша

В таком виде перебор пароля затянется, поскольку нужно вычленить соль из хэша в поле пароля. Но тем не менее, сегодня такой тип хранения используется значительно реже.

Как использовать соль? Как пожелаете! Можно конкатенировать ваш пароль с солью и взять хэш с полученной последовательности. Можно усложнять данный алгоритм, используя для хэширования следующее: $\frac{1}{2}$соли+пароль+ $\frac{1}{2}$соли и т.д. Соль можно полностью смешать с паролем, попеременно записывая символы из пароля и соли , а можно использовать несколько последовательностей солей и смешивать их с паролем.

*Заметка*: Соль также является хорошим механизмом защиты в том случае, если на сервере зарегистрировано два пользователя с одинаковыми логинами и паролями, что совсем не является редкостью.

Примерные алгоритмы генерации хэша по соли и паролю:

хэш(пароль+соль);

хэш($\frac{1}{2}$ соли+пароль+$\frac{1}{2}$ соли);

хэш(хэш($\frac{1}{2}$ соли+пароль+$\frac{1}{2}$ соли)+$\frac{1}{2}$ соли);

хэш(пароль)+хэш(соль);

хэш(хэш(пароль+соль1)+соль2);

хэш(хэш(пароль)+хэш(соль)) и т.д.

В переборе алгоритмов вас ограничивает только ваше воображение.

**Базовый способ авторизации с хешированием и изменяющейся солью.** Статическая соль, или неизменяемая, как в предыдущем пункте, является все же уязвимостью системы, поскольку она все также хранится в БД в открытом виде, и все равно может быть восстановлена перебором (хоть и значительно увеличивает время перебора вариантов). Это связано с тем, что мы, люди, все еще не способны запоминать действительно случайные последовательности, а используем в качестве паролей знакомые нам слова и числа: фамилии, имена, клички питомцев, названия любимых песен, даты дней рождения и т.д. То есть перебор пароля становится значительно проще как задача перебора по словарю хэшей, что существенно сокращает время его вскрытия, даже при условии использования специальных символов(а их тоже не очень много).

*Заметка*: можно также почитать о радужных таблицах и коллизиях.

В этом случае усложнить задачу перебора пароля позволит динамическая соль, т.е. изменяемая для каждой итерации аутентификации. Логика проста: каждый раз при подключении к серверу, генерируется новая соль и добавляется к паролю, после чего посылается на сервер. Важно, чтобы на сервере была та же самая соль (потому, как правило, этим и занимается сервер и пересылает клиенту). Чаще всего, как сид генерации используется соль с предыдущей итерации.

*Заметка*: стоит отметить, что передача пароля и логина по сети может быть защищена и другими механизмами: TLS (что лежит в основе https и является одним из самых распространенных способов для веб-сервисов, но его трогать не будем), scrypt, challenge, SRP и другие. Однако их вы скорее всего будете изучать в рамках курса по криптографии, где подробно будет описан весь механизм их работы.

**Авторизация по сертификатам.** Цифровой сертификат (далее сертификат) представляет собой набор атрибутов, идентифицирующих владельца, подписанный certificate authority (CA). CA выступает в роли посредника, который гарантирует подлинность сертификатов (по аналогии с ФМС, выпускающей паспорта). Также сертификат криптографически связан с закрытым ключом, который хранится у владельца сертификата и позволяет однозначно подтвердить факт владения сертификатом.

Если говорить простыми словами, сертификат – это файл, в котором записана информация о том, кому выдан сертификат, кем выдан, сколько он действителен, алгоритм подписи и открытый ключ. Помимо этих данных, могут быть прописана и другая информация: серийный номер, контрольная сумма сертификата, версия и т.д. Ознакомиться с сертификатом сервиса вы можете, открыв сведения о сайте. В разделе о защищенности подключения можно ознакомиться с составом сертификата (рис. 1.37).

Экспортируйте его, и можете просмотреть его состав во встроенном в ОС средстве (рис. 1.38). Вдаваться в подробности о механизме действия сертификатов, их выдаче, сроков хранения, контроле состояния, перекрестной сертификации и пр. не будем, поскольку это материал курса по криптографическим средствам защиты информации.

Отметим только несколько основных моментов, которые будут важны для вас в рамках выполнения курсовой работы:

• один сертификат может принадлежать только одному владельцу;

• сертификат генерирует СЕРВЕР сертификации и передает клиенту;

ВСЯ информация о сертификате хранится на сервере, выпустившем его, и при необходимости подтверждения подлинности сертификата этот сервер сверяет данные из него со своими записями;

сам сертификат должен храниться на клиенте и при необходимости предоставляться серверу для аутентификации клиента.

Алгоритм сертификации в рамках курсовой должен выглядеть примерно так:

• клиент запрашивает регистрацию на сервере;

• сервер запрашивает информацию о клиенте: это могут быть логины и пароли для регистрации клиента в БД;

• клиент передает запрошенные данные;

• сервер пушит данные в БД, а затем формирует сертификат для клиента и передает ему;

• клиент принимает сертификат и сохраняет на устройстве;

• при следующей сессии для аутентификации сервер запрашивает сертификат, проверяет его с имеющимися записями в БД и авторизирует клиента в соответствии с данными БД.

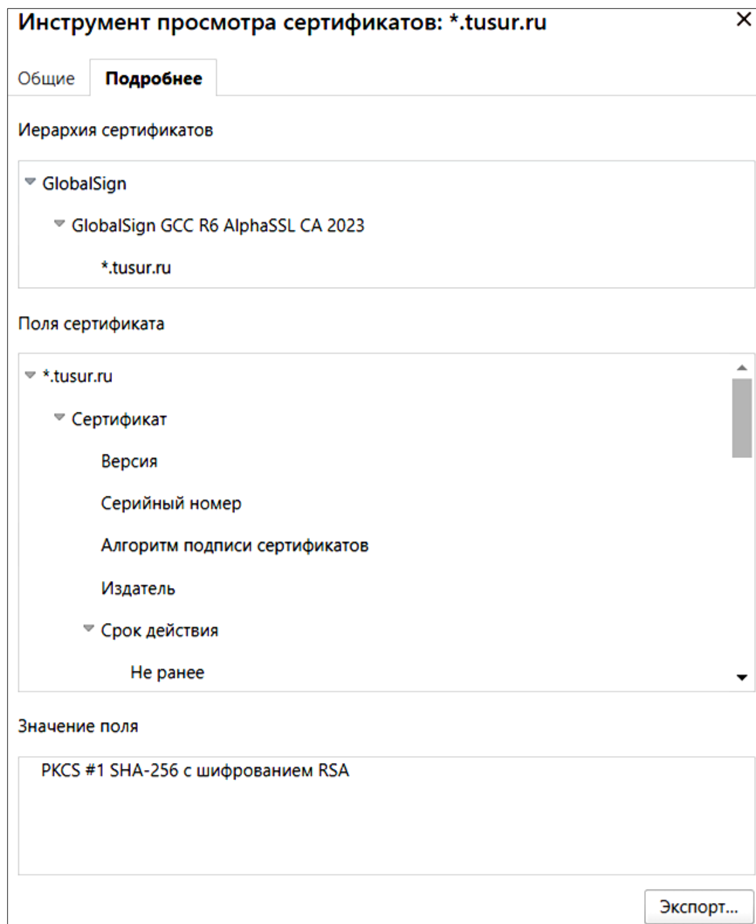


Рис. 1.37. Пример сертификата

Кажется, что в данной схеме имеется явная уязвимость: украденный сертификат с клиента может помочь авторизоваться на сервере злоумышленнику. Потому стоит еще раз упомянуть, что в сертификате хранится только ОТКРЫТЫЙ ключ, а закрытый – отдельно от сертификата. В основе этого метода защиты лежит ассиметричное шифрование. Именно связка

открытого и закрытого ключа позволяет провести аутентификацию клиента. Как это работает:

- используем закрытый ключ, чтобы зашифровать сообщение;
- высылаем зашифрованные данные и свой сертификат;
- сервер проверяет сертификат на соответствие своим данным в БД, если совпадает – расшифровывает сообщение открытым ключом из сертификата;
- если сервер смог расшифровать сообщение – значит, тот закрытый ключ, что хранится на клиенте, соответствует открытому в сертификате, а значит клиент – тот, за кого себя выдает, если нет – сервер отмечает, что сертификат был украден и может аннулировать этот сертификат.

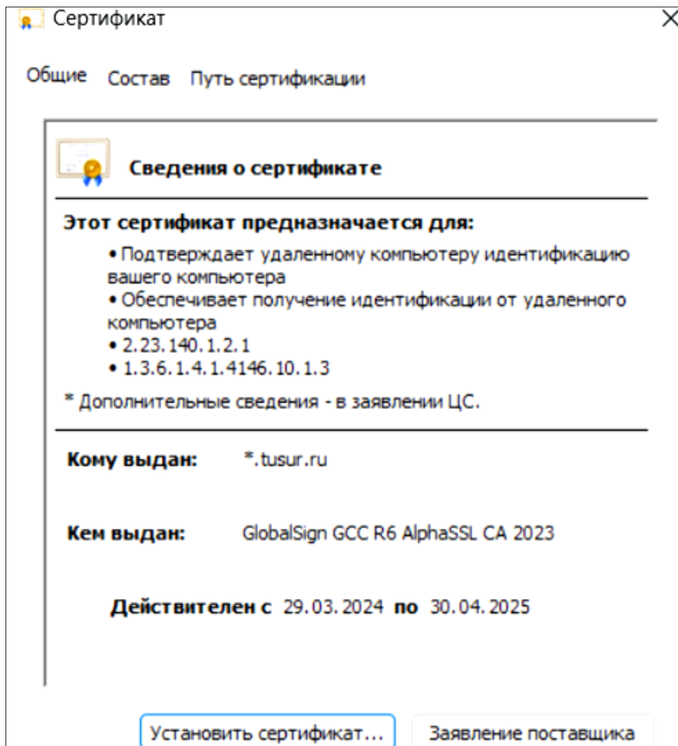


Рис. 1.38. Экспортированный сертификат

Существует множество вариаций протоколов использования сертификатов: на основе SSL, PGP и различных типов шифрования. В работе используйте встроенные в C# форматы сертификатов (например, X509).

**Авторизация по токенам.** Токен – это, если говорить кратко, ключ, или идентификатор, который используется для представления доступа к ресурсам или выполнения определенных операций. Как мы рассматривали чуть ранее, основными идентификаторами пользователей в системе являются связка логина и пароля. Но, авторизоваться с их помощью каждый раз, совершая какое-то действие – небезопасно, поскольку это кратно повышает шанс на их перехват. Потому придумали альтернативу, а именно – токен.

*Заметка*: не путайте этот токен с понятием из сферы криптовалют, там используют немного другой механизм, и они несут более широкий спектр функций, чем в простой авторизации.

Аутентификация на основе токенов обычно состоит из четырех этапов:

**Первоначальный запрос** – пользователь запрашивает доступ к защищенному ресурсу. Изначально пользователь должен идентифицировать себя способом, не требующим токена, например, при помощи имени пользователя или пароля.

**Верификация** – аутентификация определяет, что идентификационные данные пользователя верны, и проверяет, какие полномочия он имеет в запрошенной системе.

**Токены** – система выпускает токен и передает его пользователю. В случае аппаратного токена это подразумевает физическую передачу токенов пользователю. В случае программных токенов это происходит в фоновом режиме, пока фоновые процессы пользователя обмениваются данными с сервером.

**Сохранение** – токен удерживается пользователем, или физически, или в браузере/мобильном телефоне. Это позволяет ему выполнять аутентификацию без указания идентификационных данных.

Теперь, для следующего действия, пользователь должен вместе с запросом к серверу передать не логин и пароль, а токен.

Действующие токены хранятся на сервере, однако их не привязывают к определенному пользователю. Сервер при получении запроса просто проверяет токен в списке действующих, и если он там есть, высылает ответ. При этом серверу не важно, кому именно он отвечает.

# 2. ХОД РАБОТЫ

## 2.1. Установка начальных пакетов. Ubuntu

Установите ОС Ubuntu (рис. 2.1). Данное пособие выполнено на Ubuntu 24.04 LTS. Данная версия не является обязательной, можете найти и другие образы. Затем требуется установить Visual Studio Code с официального сайта (рис. 2.2).

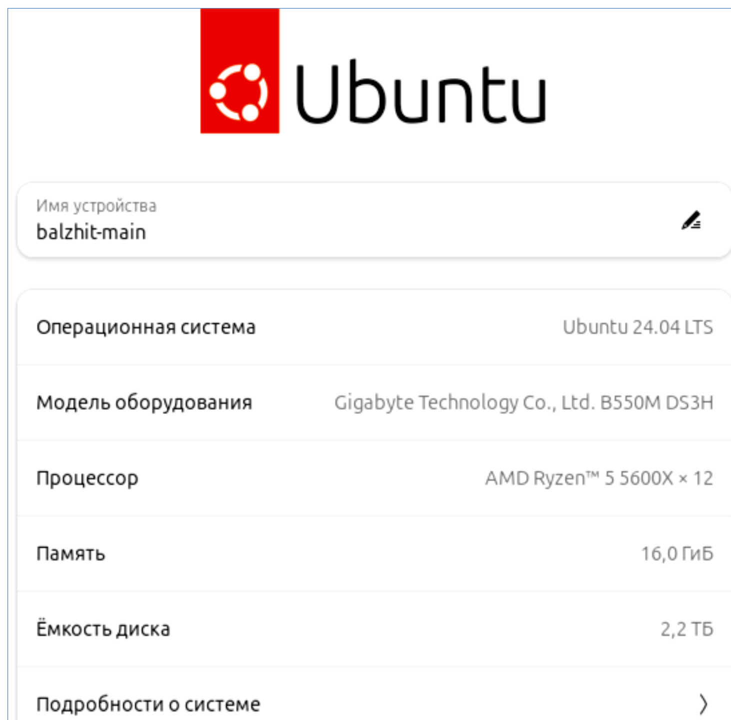


Рис. 2.1. Установка ОС Ubuntu

При необходимости можете ознакомиться с интерфейсом и функционалом VS Code  на официальном сайте Microsoft. После ознакомления с интерфейсом и прочим можете начать устанавливать требуемые расширения : C# Dev Kit (рис. 2.3), .NET Install Tool, C# (Base language support for C#). Далее требуется установить SDK .NET (рис. 2.4, 2.5).

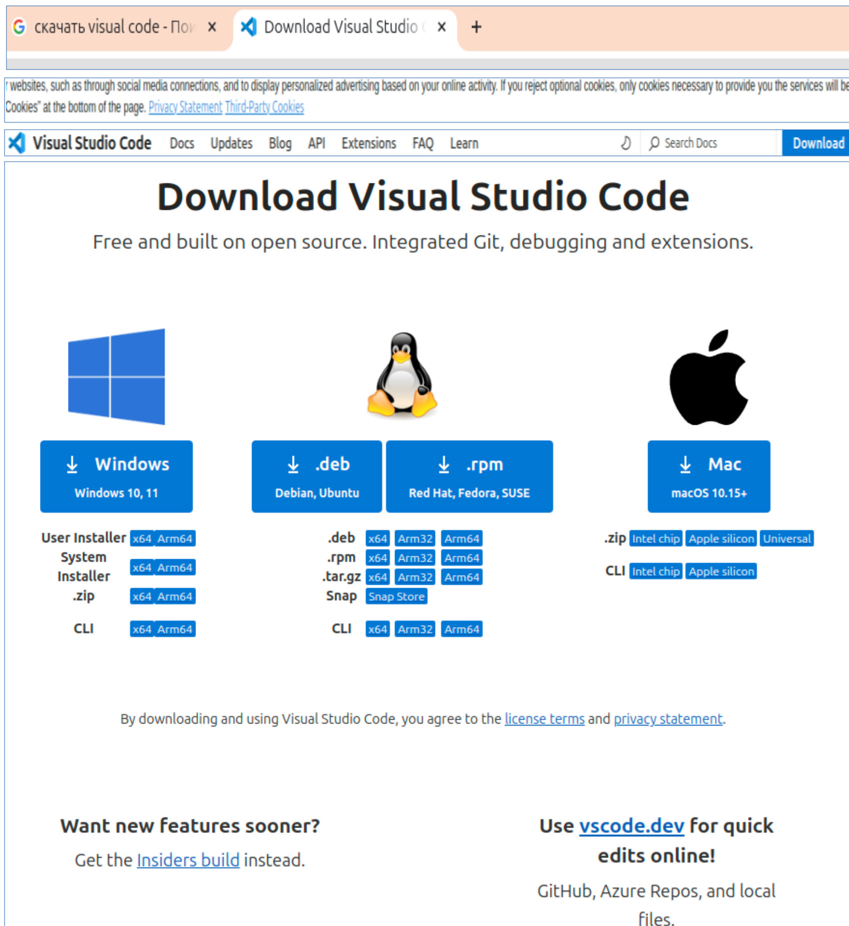


Рис. 2.2. Официальный сайт Visual Studio Code

Можно установить dotnet через скрипт и с помощью SDK. Пользуйтесь удобным вариантом, а далее будет показан вариант с SDK. Выбираем пакет для Ubuntu (рис. 2.6), на странице с пакетами Ubuntu выбирайте нужную версию. В нашем случае – 24.04.

Далее запускаем терминал и запускаем три bash-скрипта по очереди (рис. 2.7):

```
sudo apt-get update && \  sudo apt-get install -y dotnet-sdk-8.0
sudo apt-get update && \  sudo apt-get install -y aspnetcore-runtime-8.0
sudo apt-get install -y dotnet-runtime-8.0
```

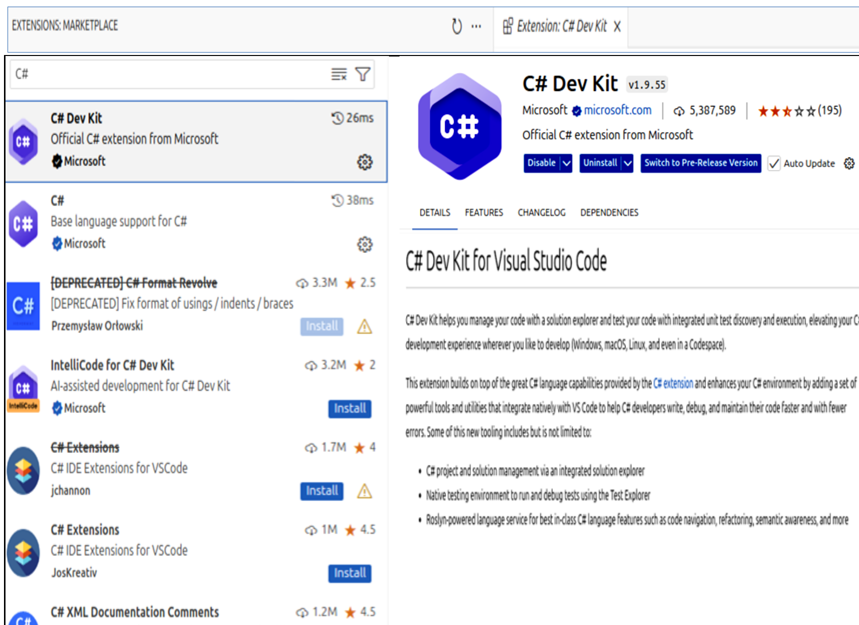


Рис. 2.3. Расширение C# Dev Kit

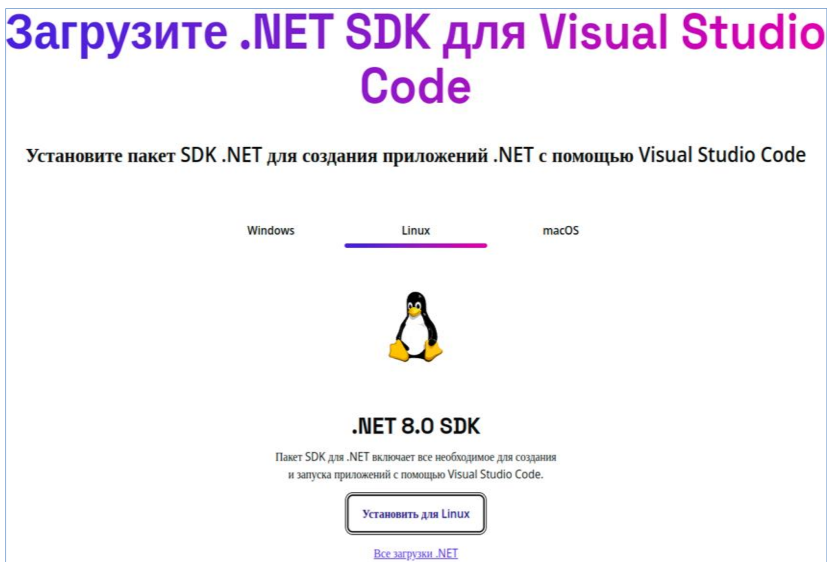


Рис. 2.4. SDK .NET

Learn / .NET /

# Установка .NET в Linux

Статья • 26.05.2024 • Участники: 8

Обратная связь

**В этой статье**

- Пакеты
- Прикрепление
- Установка вручную

Установка на Linux

В этой статье описывается, как .NET доступен в различных дистрибутивах Linux. .NET можно установить диспетчером пакетов, оснасткой или вручную. .NET также доступен как образ контейнера.

## Пакеты

.NET доступен в официальных архивах пакетов для различных дистрибутивов Linux и packages.microsoft.com.

- Альпийский
- Debian
- Fedora
- openSUSE
- SLES
- Ubuntu

.NET поддерживается корпорацией Майкрософт при скачивании из источника Майкрософт. Лучшая поддержка предоставляется от Майкрософт при скачивании из другого места. При возникновении проблем можно открывать проблемы в dotnet/core.

## Прикрепление

Пакеты оснастки пакета SDK для .NET предоставляются и поддерживаются каноническим пакетом. Пакеты Snap — это отличная альтернатива диспетчеру пакетов, встроенному в дистрибутив Linux.

- Установка среды выполнения .NET с помощью Snap
- Установка пакета SDK для .NET с помощью Snap

## Установка вручную

Вы можете установить .NET вручную следующим образом:

- Установка вручную
- Установка скриптов

Рис. 2.5. Статья «Установка .Net в Linux»

- Мой дистрибутив Ubuntu не включает нужную версию .NET, или мне нужна не поддерживается версия .NET.
- Я хочу установить предварительную версию
- Я не хочу использовать APT
- Я использую ЦП на основе Arm
- Я использую платформу IBM System Z

## Я использую Ubuntu 22.04 или более поздней версии, и мне нужен только .NET

Установите .NET через веб-канал Ubuntu. Дополнительные сведения см. на следующих страницах:

- Установите .NET в Ubuntu 24.04.
- Установите .NET в Ubuntu 23.10.
- Установите .NET в Ubuntu 23.04.
- Установите .NET в Ubuntu 22.04.

ⓘ **Важно!**

Версии пакета SDK для .NET, предлагаемые Каноническим, всегда находятся в группе функций .1xx. Если вы хотите использовать более новый выпуск группы компонентов, используйте веб-канал Майкрософт для установки пакета SDK. Убедитесь, что вы просматриваете сведения в пакете .NET в статье Linux , чтобы понять последствия переключения между веб-каналами репозитория.

Если вы собираетесь установить репозиторий Майкрософт для использования других пакетов Майкрософт, таких как `powershell` `mdatp`, или `mssql`, необходимо отменять использование пакетов .NET, предоставляемых репозиторием Майкрософт. Инструкции по отмене использования пакетов см. в разделе "Мой дистрибутив Linux" предоставляет пакеты .NET, и я хочу их использовать.

Рис. 2.6. Выбор пакета

```
science@science-MS-7C37:~/Project/WebApp$ sudo apt-get update && sudo apt-get install -y dotnet-sdk-8.0
Сущ:1 http://ru.archive.ubuntu.com/ubuntu noble InRelease
Сущ:2 http://security.ubuntu.com/ubuntu noble-security InRelease
Сущ:3 http://ru.archive.ubuntu.com/ubuntu noble-updates InRelease
Сущ:4 https://dl.google.com/linux/chrome/deb stable InRelease
Сущ:5 http://ru.archive.ubuntu.com/ubuntu noble-backports InRelease
Чтение списков пакетов… Готово
Чтение списков пакетов… Готово
Построение дерева зависимостей… Готово
Чтение информации о состоянии… Готово
Уже установлен пакет dotnet-sdk-8.0 самой новой версии (8.0.108-0ubuntu1~24.04.1).
• science@science-MS-7C37:~/Project/WebApp$ sudo apt-get install -y aspnetcore-runtime-8.0
  Чтение списков пакетов… Готово
  Построение дерева зависимостей… Готово
  Чтение информации о состоянии… Готово
• science@science-MS-7C37:~/Project/WebApp$ sudo apt-get install -y dotnet-runtime-8.0
  Чтение списков пакетов… Готово
  Построение дерева зависимостей… Готово
  Чтение информации о состоянии… Готово
```

Рис. 2.7. Установка пакетов через терминал

Для проверки установки пакетов можете запустить команду проверки версии dotnet. (рис. 2.8)

```
science@science-MS-7C37:~/Project/WebApp$ dotnet --version
8.0.108
```

Рис. 2.8. Проверка версии dotnet

## 2.2. Установка начальных пакетов. Astra Linux

Для установки Astra Linux скачайте образ с указанного на sdo репозитория (рис. 2.9, 2.10).

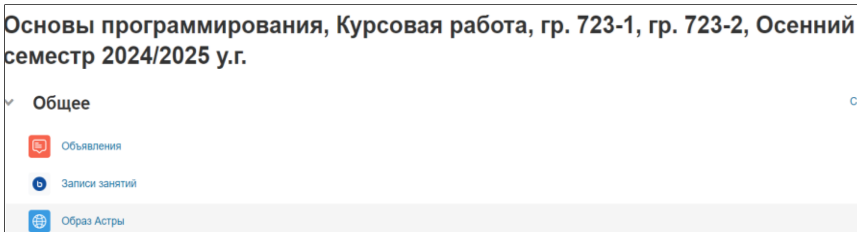


Рис. 2.9. Ссылка на образ Astra Linux

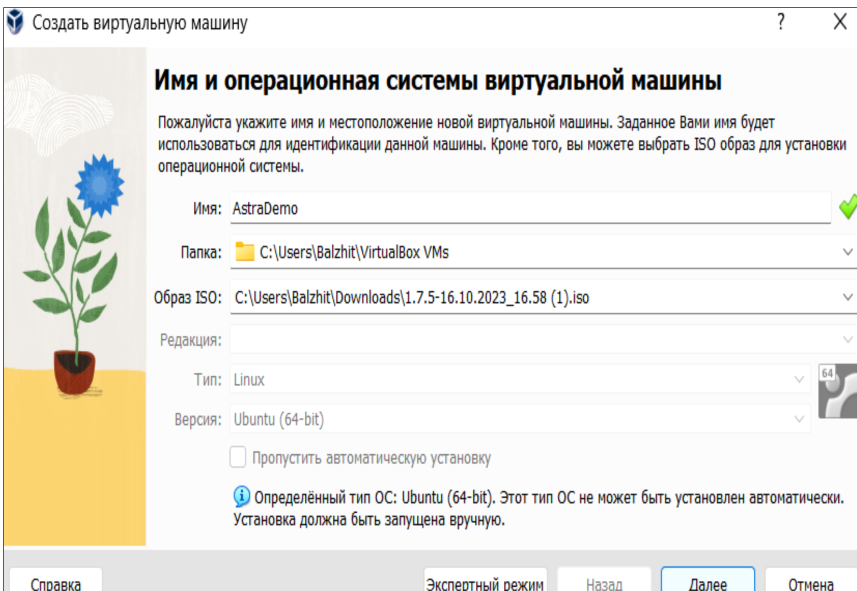


Рис. 2.10. Создание виртуальной машины

Запустите машину и продолжите установку. На этапе выбора клавиш смены раскладки выберите удобный для вас вариант и продолжайте установку. Установите имя системы на любую удобную вам (рис. 2.11). Не забудьте указать имя администратора системы (рис. 2.12). Его пароль сделайте запоминающимся вам, он может пригодиться в дальнейшем (8 символов). Затем выбирается метод разметки диска (рис. 2.13, 2.14). Далее выберите ядро для установки (рис. 2.15).

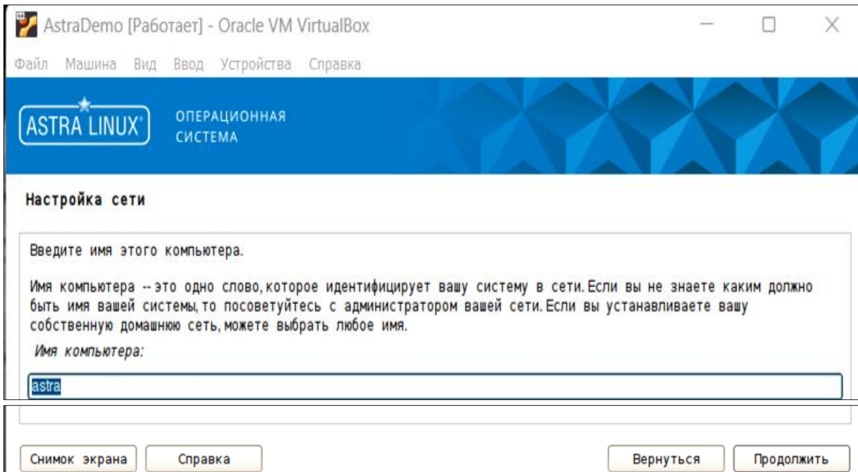

Рис. 2.11. Ввод имени компьютера

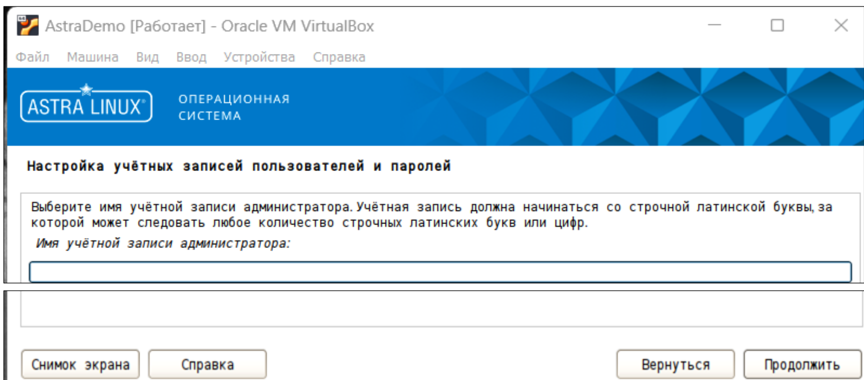

Рис. 2.12. Выбор имени учетной записи администратора

Дальнейшие инструкции будут актуальны для этой версии ядра системы. Во избежание проблем в работе рекомендуется использовать базовую версию Astra (рис. 2.16). После перезагрузки системы войдите в создан-

ную запись администратора. Иногда после загрузки системы вы можете увидеть экран загрузки без графической оболочки: в этом случае нужно подождать, пока она прогрузится (рис. 2.17). Для установки иных разрешений экрана потребуется установка дополнений (рис. 2.18).

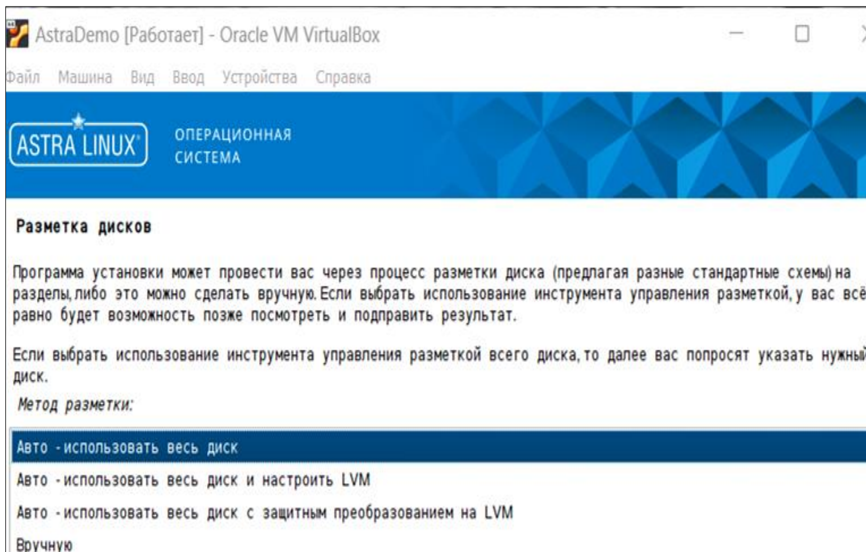

Рис. 2.13. Разметка дисков

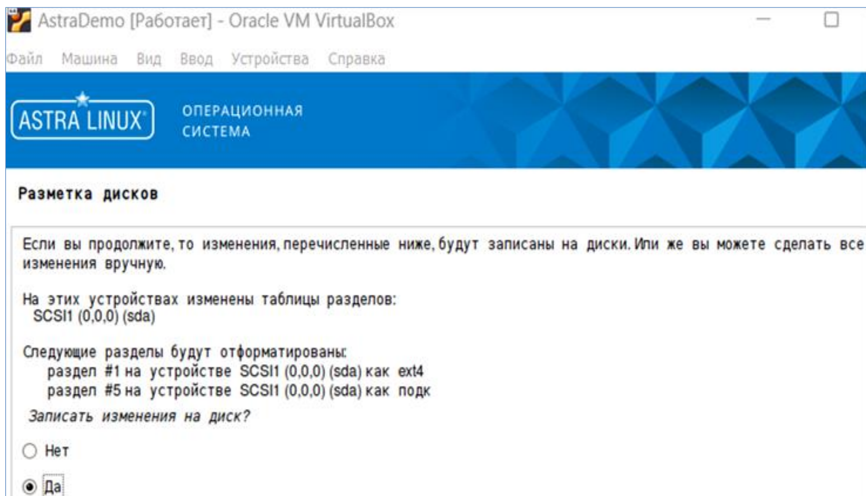

Рис. 2.14. Предупреждение о записи на диск

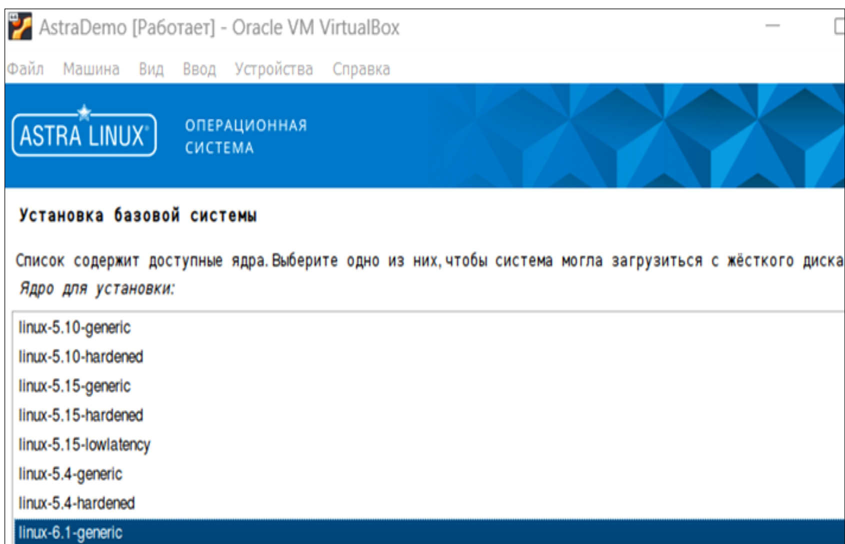


Рис. 2.15. Установка базовой системы

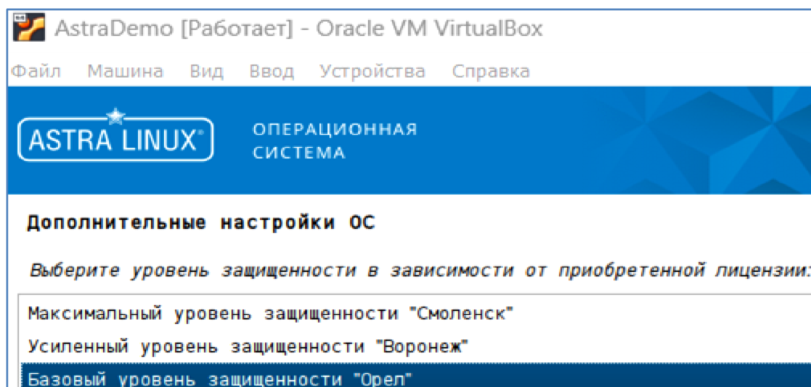


Рис. 2.16. Выбор уровня защищенности

```
Astra Linux 1.7.5 astra tty1
astra login: _
```

Рис. 2.17. Экран входа в систему без графической оболочки

Напрямую запустить установку пакета через sudo не выйдет из-за настроек безопасности. Однако обойти запрет позволит запуск пакета дополнений как bash-скрипта (рис. 2.19, 2.20). Напрямую установить dotnet и aspnetcore не получится (рис. 2.20).

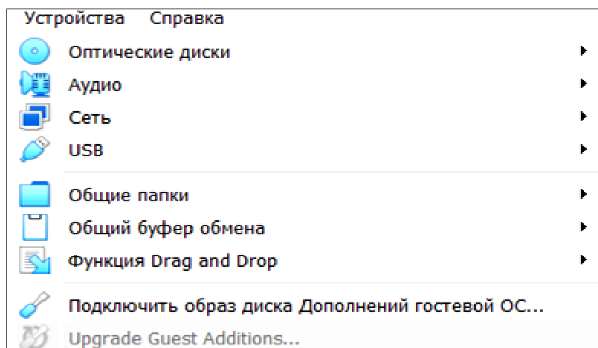


Рис. 2.18. Устройства виртуальной машины

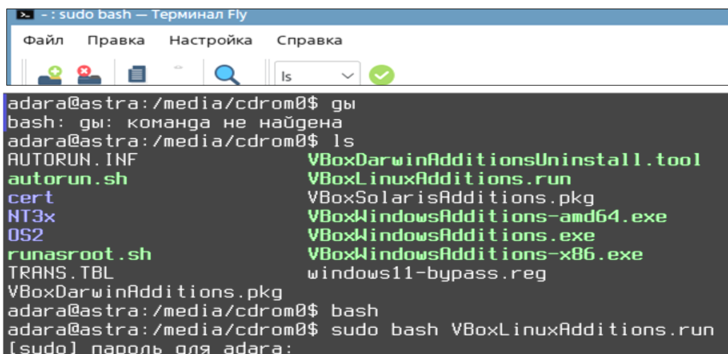


Рис. 2.19. Запуск пакета дополнений как bash-скрипта

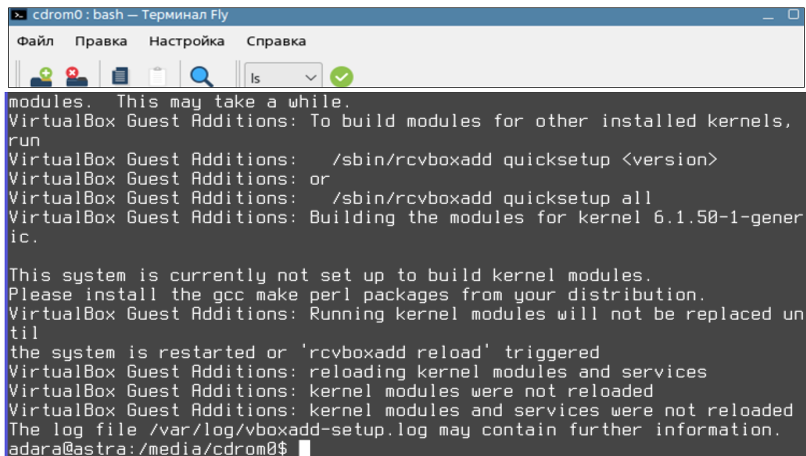


Рис. 2.20. Bash-скрипт

```
adara@astra:~$ sudo apt install aspnetcore-* dotnet-*
[sudo] пароль для adara:
Попробуйте ещё раз.
[sudo] пароль для adara:
Чтение списков пакетов… Готово
Построение дерева зависимостей
Чтение информации о состоянии… Готово
E: Невозможно найти пакет aspnetcore-*
E: Не удалось найти ни один пакет с помощью шаблона «aspnetcore-*»
E: Не удалось найти ни один пакет с помощью регулярного выражения «aspnetc
ore-*»
E: Невозможно найти пакет dotnet-*
E: Не удалось найти ни один пакет с помощью шаблона «dotnet-*»
E: Не удалось найти ни один пакет с помощью регулярного выражения «dotnet-
*»
adara@astra:~$
```

Рис. 2.21. Попытка установить dotnet и aspnetcore напрямую

### 2.3. Установка VS Code

Для загрузки VSCode потребуется несколько шагов.

1. Открыть терминал Fly и ввести команду подгрузки пакета libxkbfile1 до версии не ниже 1:1.1.0 из репозиториев Debian: wget http://archive.ubuntu.com/ubuntu/pool/main/libx/libxkbfile/libxkbfile1_1.1.0-1_amd64.deb (рис. 2.22).

```
adara@astra:~$ wget http://archive.ubuntu.com/ubuntu/pool/main/libx/libxkb
file/libxkbfile1_1.1.0-1_amd64.deb
--2024-09-12 14:37:38--  http://archive.ubuntu.com/ubuntu/pool/main/libx/l
ibxkbfile/libxkbfile1_1.1.0-1_amd64.deb
Распознаётся archive.ubuntu.com (archive.ubuntu.com)… 185.125.190.82, 91.1
89.91.82, 185.125.190.81, ...
Подключение к archive.ubuntu.com (archive.ubuntu.com)|185.125.190.82|:80..
. соединение установлено.
HTTP-запрос отправлен. Ожидание ответа… 200 OK
Длина: 65300 (64K) [application/vnd.debian.binary-package]
Сохранение в: «libxkbfile1_1.1.0-1_amd64.deb»

libxkbfile1_1.1.0- 100%[=============>]  63,77K   264KB/s    за 0,2s

2024-09-12 14:37:38 (264 KB/s) - «libxkbfile1_1.1.0-1_amd64.deb» сохранён
[65300/65300]
```

Рис. 2.22. Загрузка пакета libxkbfile1

2. Установите скачанный пакет: sudo apt install ./libxkbfile1_1.1.0-1_amd64.deb (рис. 2.23).

3. Следующая команда необязательная, но на всякий случай выполните: sudo apt install ca-certificates apt-transport-https (рис. 2.24).

4. Далее требуется скачать список продуктов от Microsoft (это нам позже понадобится не раз): sudo wget https://packages.microsoft.com/config/debian/10/prod.list-O/etc/apt/sources.list.d/microsoft-prod.list

```
adara@astra:~$ sudo apt install ./libxkbfile1_1.1.0-1_amd64.deb
Чтение списков пакетов… Готово
Построение дерева зависимостей
Чтение информации о состоянии… Готово
Заметьте, вместо «./libxkbfile1_1.1.0-1_amd64.deb» выбирается «libxkbfile1
»
Следующие пакеты будут обновлены:
  libxkbfile1
Обновлено 1 пакетов, установлено 0 новых пакетов, для удаления отмечено 0
пакетов, и 0 пакетов не обновлено.
Необходимо скачать 0 B/65,3 kB архивов.
После данной операции объём занятого дискового пространства уменьшится на
14,3 kB.
Пол:1 /home/adara/libxkbfile1_1.1.0-1_amd64.deb libxkbfile1 amd64 1:1.1.0-
```

Рис. 2.23. Установка скачанного пакета

```
adara@astra:~$ sudo apt install ca-certificates apt-transport-https
Чтение списков пакетов… Готово
Построение дерева зависимостей
Чтение информации о состоянии… Готово
Уже установлен пакет ca-certificates самой новой версии (20230311ubuntu0.2
0.04.1).
Уже установлен пакет apt-transport-https самой новой версии (1.8.2.3+ci202
309131156+astra26).
Обновлено 0 пакетов, установлено 0 новых пакетов, для удаления отмечено 0
пакетов, и 0 пакетов не обновлено.
```

Рис. 2.24. Установка дополнительных пакетов

```
adara@astra:~$ wget -O - https://packages.microsoft.com/keys/microsoft.asc
 | gpg --dearmor | sudo tee /etc/apt/trusted.gpg.d/microsoft.asc.gpg > /de
v/null
--2024-09-12 14:41:50--  https://packages.microsoft.com/keys/microsoft.asc
Распознаётся packages.microsoft.com (packages.microsoft.com)… 13.107.246.5
3, 2620:1ec:29:1::72
Подключение к packages.microsoft.com (packages.microsoft.com)|13.107.246.5
3|:443... соединение установлено.
HTTP-запрос отправлен. Ожидание ответа… 200 OK
Длина: 983 [application/octet-stream]
Сохранение в: «STDOUT»
-                   100%[=============>]     983  --.-KB/s    за 0s
```

Рис. 2.25. Скачивание продуктов от Microsoft

5. Обновите кэш: sudo apt update.

6. Установите пакеты от разработчиков: sudo apt install dotnet-sdk-8.* затем sudo apt install aspnetcore-runtime-8.* (это необязательно, но рекомендуется перестраховаться).

7. Скачайте установщик VSCode с официального сайта (рис. 2.26).

8. В директиве загрузок откройте терминал (рис. 2.27).

9. Запустите установку (рис. 2.28).

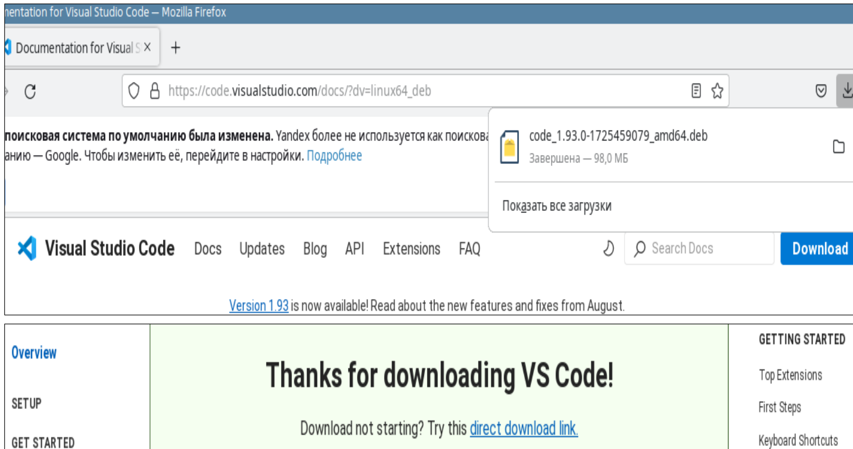


Рис. 2.26. Скачивание установщика VS Code

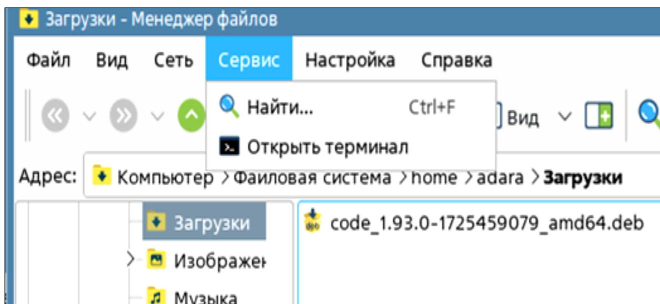


Рис. 2.27. Открытие терминала из папки «Загрузки»

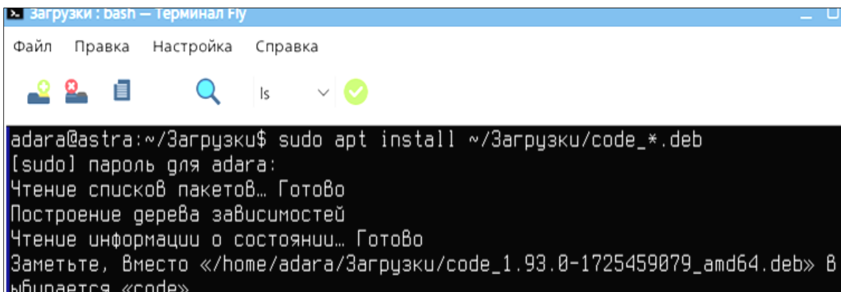


Рис. 2.28. Запуск установки

10. В процессе загрузки потребуется выдать разрешение на установку. VSCode установлен (рис. 2.29).

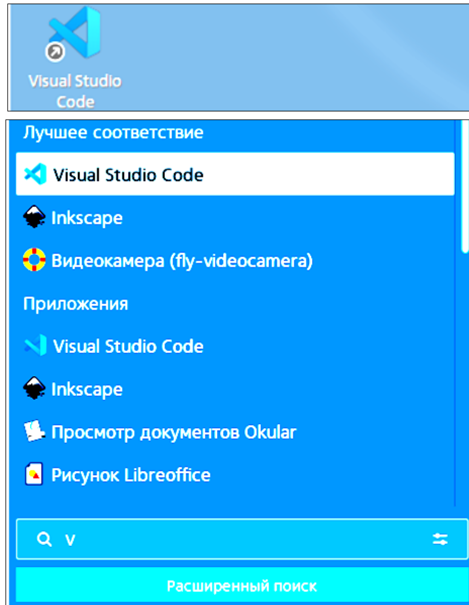


Рис. 2.29. Проверка успешной установки

## 2.4. Установка расширений C# для VSCode

VSCode позволяет нативно устанавливать расширения для различных языков(при условии, что требуемые для языков пакеты установлены в ОС). Установите C# Dev Kit, требуемые ему пакеты он подтянет автоматически (рис. 2.30).

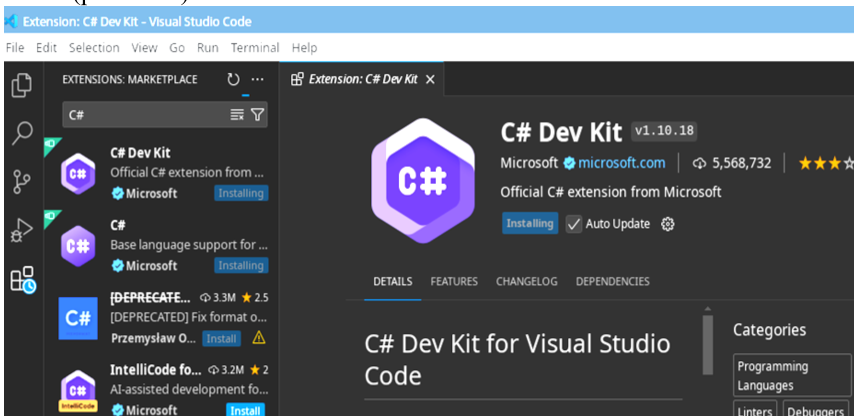


Рис. 2.30. Установка C# Dev Kit

## 2.5. Создание первого серверного WEBApplication

Теперь вы можете в VS Code создать проект на .Net (рис. 2.31–2.34).

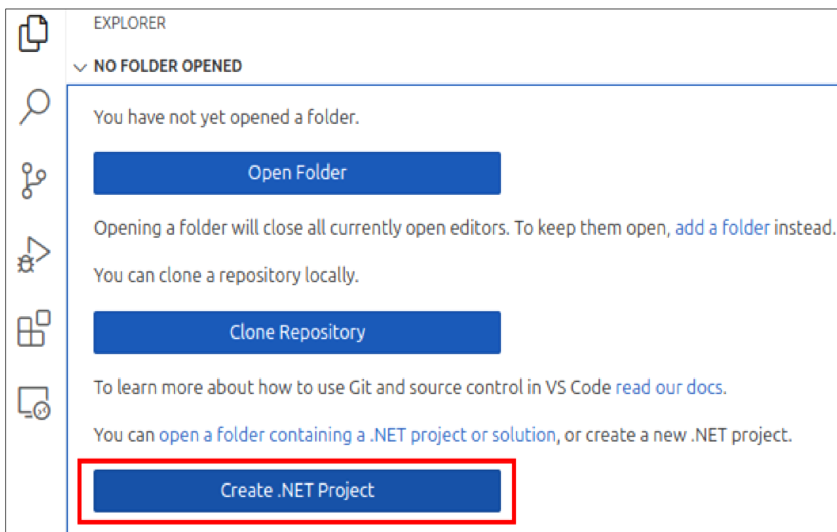


Рис. 2.31. Создание проекта на .Net

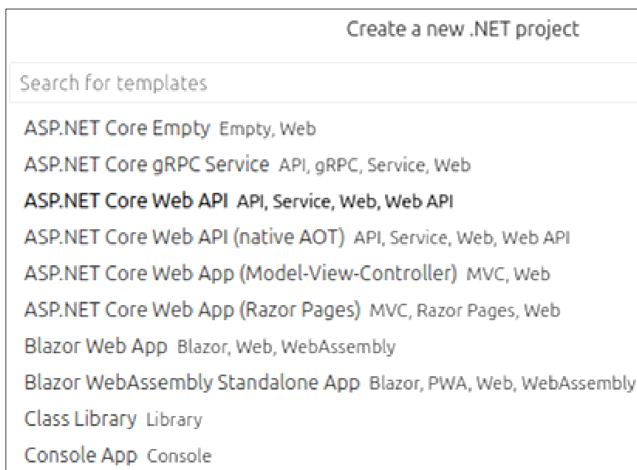


Рис. 2.32. Выбор типа проекта

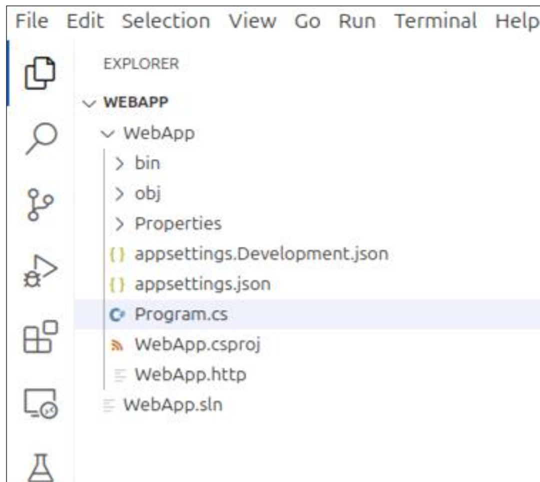


Рис. 2.33. Структура созданного проекта

```
Program.cs ×
WebApp > Program.cs > ...
17  app.UseHttpsRedirection();
18
19  var summaries = new[]
20  {
21      "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
22  };
23
24  app.MapGet("/weatherforecast", () =>
25  {
26      var forecast =  Enumerable.Range(1, 5).Select(index =>
27          new WeatherForecast
28          (
29              DateOnly.FromDateTime(DateTime.Now.AddDays(index)),
30              Random.Shared.Next(-20, 55),
31              summaries[Random.Shared.Next(summaries.Length)]
32          ))
33          .ToArray();
34      return forecast;
35  })
36  .WithName("GetWeatherForecast")
37  .WithOpenApi();
38
39  app.Run();
40
    1 reference
41  record WeatherForecast(DateOnly Date, int TemperatureC, string? Summary)
42  {
        0 references
43      public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);
44  }
```

Рис. 2.34. Содержимое файла Program.cs

Начинаем работу с такого старта. Тут вы видите демонстрационный вариант WebApplication. Запустите его. Первый шаг сделан, у вас есть веб-сервис погоды (рис. 2.35).

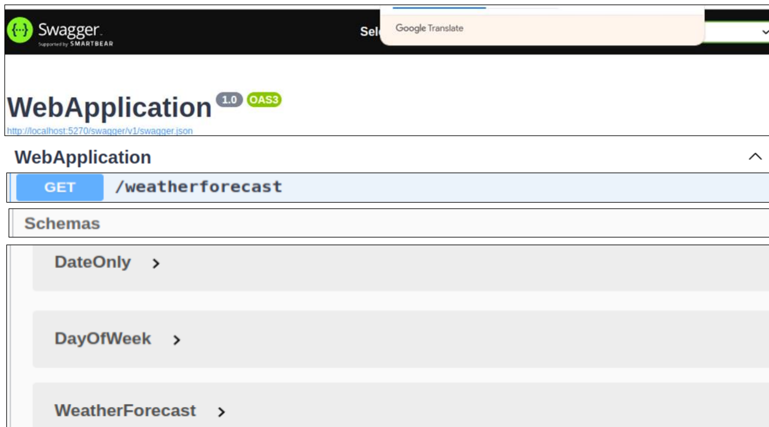


Рис 2.35. Запущенный веб-сервис

Разумеется, нам этот формат не подойдет, но насладитесь моментом, дальше такой красоты вы не увидите. Закройте страницу браузера и удостоверьтесь, что серверная часть ПО все еще работает (рис. 2.36).

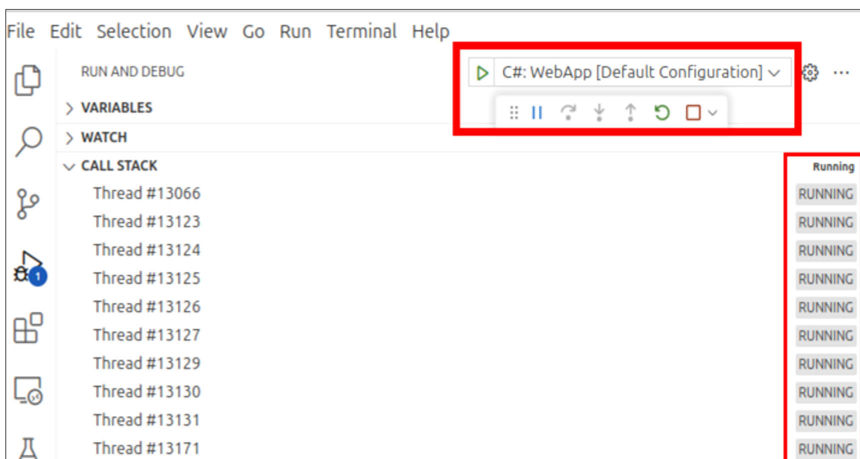


Рис. 2.36. Серверная часть запущена

Для нативной проверки работы сервера можно отправить запрос из терминала (рис. 2.37).

```
science@science-MS-7C37:~/Project/WebApp$ curl http://localhost:5254/swagger/index.html
<!-- HTML for static distribution bundle build -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Swagger UI</title>
    <link rel="stylesheet" type="text/css" href="./swagger-ui.css">
    <link rel="icon" type="image/png" href="./favicon-32x32.png" sizes="32x32" />
    <link rel="icon" type="image/png" href="./favicon-16x16.png" sizes="16x16" />
    <style>
```

Рис. 2.37. Проверка

Обратите внимание на запрос. О нем рассказано в разд. 1.4 cURL. После запроса выведен ответ (html-страница), которую формирует сервер в ответ на ваш запрос. Браузер умеет красиво отрисовывать эту верстку, терминал – нет, но нам важно понимать, что запрос получает ответ. URL, к которому вы обратились, можно получить из браузера, удалив спецсимволы. Но, если вы хотите получить более удобочитаемый вариант ответа (в json-формате), обратитесь к файлу WebApplication.http (рис. 2.38). В нем прописаны хост адрес, обрабатываемый запрос и маршрут.

```
science@science-MS-7C37:~/Project/WebApp$ curl http://localhost:5254/weatherForecast
[{"date":"2024-09-02","temperatureC":25,"summary":"Sweltering","temperatureF":76},{"date":"2024-09-03
","temperatureC":2,"summary":"Sweltering","temperatureF":35},{"date":"2024-09-04","temperatureC":7,"s
ummary":"Freezing","temperatureF":44},{"date":"2024-09-05","temperatureC":10,"summary":"Chilly","temp
eratureF":49},{"date":"2024-09-06","temperatureC":53,"summary":"Cool","temperatureF":127}]science@sci
ence-MS-7C37:~/Project/WebApp$
```

Рис. 2.38. Просмотр содержимого файла WebApplication.http

## 2.6. Postman

В дальнейшем роль клиента будет выполнять расширение Postman. для его установки перейдите в раздел Extension и найдите его в поиске. Установите встроенными средствами VSCode (рис. 2.39).

После установки у вас появится дополнительный раздел в меню слева (2.40).

Перейдите в него и выберете создайте новый запрос http. В выпадающем меню можно выбрать тип запроса, а в соседнем окне прописать URL, по которому запрос будет передаваться. Результат запроса появится ниже.

На первом этапе этого будет достаточно для ознакомления с http-запросами.

## 2.7 NuGET

Внимательно прочитайте данный раздел полностью, прежде чем приступать к настройке.

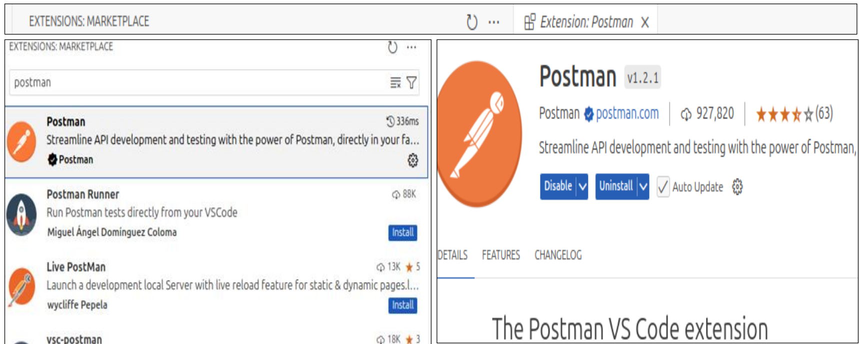


Рис. 2.39. Расширение Postman

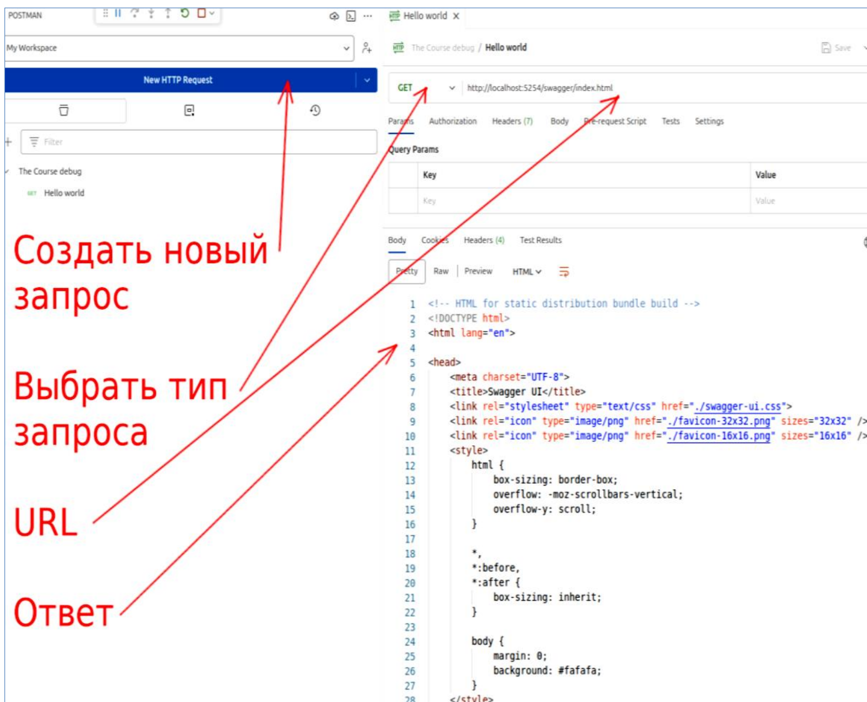


Рис. 2.40. Дополнительный раздел

В VSCode можно установить пакетный менеджер NuGET посредством установки расширений. В списке расширений найдите два расширения: NuGet Reverse Package и NuGet Reverse Package Search («Add Package» support). Установите оба (рис. 2.41).

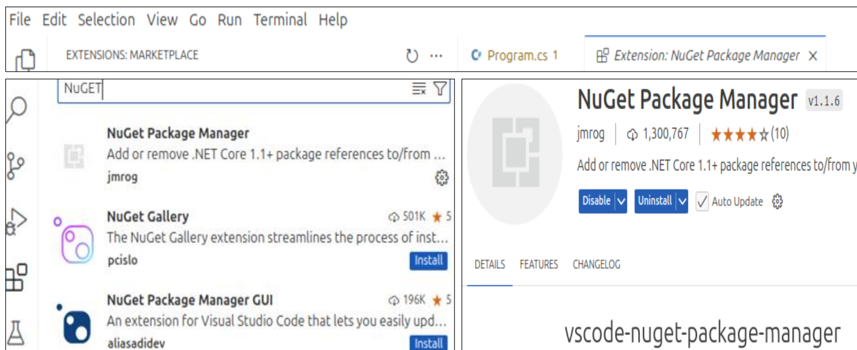


Рис. 2.41. Установка расширений

*Заметка 0:* Может потребоваться решение одной проблемы этих пакетных менеджеров. Если при установке иных расширений с помощью NuGET у вас всплывет окно ошибки: Versioning information could not be retrieved from the NuGet package repository, сделайте следующее.

Откройте терминал и выполните команду: cd.vscode/extensions/jmrog.vscode-nuget-package-manager-1.1.6/out/src/actions/add-methods/.

Она перенесет вас в директиву NuGET менеджера, а конкретно – туда, где расположен скрипт добавления пакетов (рис. 2.42).

Вам нужен файл fetchPackageVersions.js. Откройте его в редакторе. В примере используется nano (его тоже может потребоваться установить). Для этого выполните команду: nano fetchPackageVersions.js (рис. 2.43).

В строке

```
node_fetch_1.default(`${versionsUrl}${selectedPackageName}/index.json`,
```

надо дописать функцию приведения к нижнему регистру:

```
node_fetch_1.default(`${versionsUrl}${selectedPackageName.toLowerCase()}/index.json`, utils_1.getFetchOptions(vscode.workspace.getConfiguration('http')))
```

как сделано в примере на рис. 2.43.

*Заметка 1:* На самом деле все предыдущие шаги были не столь необходимы. Существует более проработанная альтернатива предыдущим двум расширениям. Речь идет о Visual NuGet (рис. 2.44).

Данное расширение предоставляет интерфейс взаимодействия с пакетным менеджером NuGet для VS Code идентичный тому, что представлен в Visual Studio для Windows. Предположительно данное расширение более стабильно, однако оно находится в раннем превью доступе и может грешить ошибками. Все перечисленные расширения – неофициальные и разработаны сообществом. Выбирайте на свой вкус.

```
balzhit@balzhit-main:~/vs code project/WebApplication1$ cd
balzhit@balzhit-main:~$ cd .vscode/extensions/jmrog.vscode-nuget-package-manager-1.1.6/out/src/actions/add-methods/
balzhit@balzhit-main:~/.vscode/extensions/jmrog.vscode-nuget-package-manager-1.1.6/out/src/actions/add-methods$ ls -a
 .  ..  fetchPackages.js  fetchPackageVersions.js  getNuGetSearchUrl.js  handleSearchResponse.js  handleVersionsQuickPick.js  handleV
balzhit@balzhit-main:~/.vscode/extensions/jmrog.vscode-nuget-package-manager-1.1.6/out/src/actions/add-methods$
```

Рис. 2.42. Переход в директиву NuGET-менеджера

```
GNU nano 7.2                                  fetchPackageVersions.js
"use strict";
Object.defineProperty(exports, "__esModule", { value: true });
const vscode = require("vscode");
const node_fetch_1 = require("node-fetch");
const shared_1 = require("../shared");
const constants_1 = require("../../constants");
const utils_1 = require("../../utils");
function fetchPackageVersions(selectedPackageName, versionsUrl = constants_1.NUGET_VERSIONS_URL) {
    if (!selectedPackageName) {
        // User has canceled the process.
        return Promise.reject(constants_1.CANCEL);
    }
    vscode.window.setStatusBarMessage('Loading package versions...');
    return new Promise((resolve) => {
        node_fetch_1.default(`${versionsUrl}${selectedPackageName.toLowerCase()}/index.json`, utils_1.getFetchOptions(vscode.workspace.getConfiguration('http')))
            .then((response) => {
            shared_1.clearStatusBar();
            resolve({ response, selectedPackageName });
```

Рис. 2.43. Файл fetchPackageVersions.js в редакторе nano

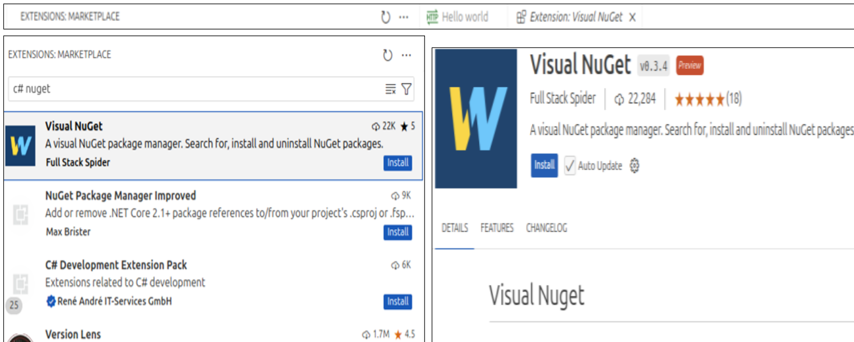


Рис. 2.44. Расширение Visual NuGet

## 2.8. Работа с SQLite

Сначала установите SQLite (рис. 2.45).

```
balzhit@balzhit-main:~$ sudo apt install sqlite3
Чтение списков пакетов… Готово
Построение дерева зависимостей… Готово
Чтение информации о состоянии… Готово
Уже установлен пакет sqlite3 самой новой версии (3.45.1-1ubuntu2).
```

Рис. 2. 45. Установка SQLite

*Заметка*: Astra для установки пакетов требует подключения оптического диска с образом ОС и списком разрешенных репозиториев от разработчиков ОС (рис 2.46).

```
adara@astra:~$ sudo apt install sqlite3
Чтение списков пакетов… Готово
Построение дерева зависимостей
Чтение информации о состоянии… Готово
Предлагаемые пакеты:
  sqlite3-doc
Следующие НОВЫЕ пакеты будут установлены:
  sqlite3
Обновлено 0 пакетов, установлено 1 новых пакетов, для удаления отмечено 0
пакетов, и 0 пакетов не обновлено.
Необходимо скачать 0 B/1 157 kB архивов.
После данной операции объём занятого дискового пространства возрастёт на 2
 858 kB.
Смена носителя: вставьте диск с меткой
 «OS Astra Linux 1.7.5 1.7_x86-64 DVD»
в устройство «/media/cdrom/» и нажмите [Enter]
```

Рис. 2.46. Подключение оптического диска

Верните оптический диск (если он по каким-то причинам отключился) и продолжите установку (рис. 2.47, 2.48).

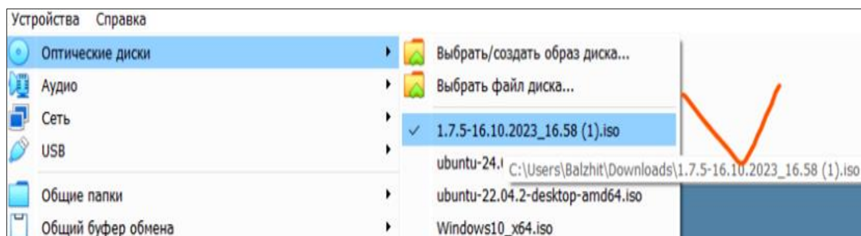


Рис. 2.47. Выбор оптического диска

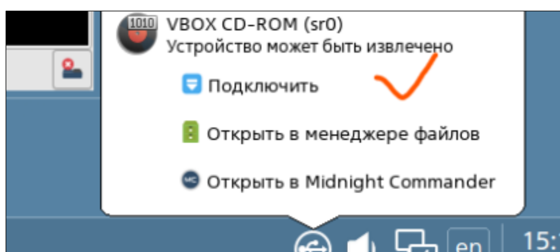


Рис. 2.48. Подключение оптического диска

Создайте новую БД в удобном вам месте (в примере – в директории проекта). В примере создана БД с названием user_DB.db (рис. 2.49).

Затем откройте проект в VSCode и нажмите сочетание клавиш Ctrl + Shift + P. Таким образом вы откроете терминал команд оболочки VSCode (рис. 2.50). Если вы установили все правильно, то в списке вы увидите SQLite: Open database. Выберите эту опцию и укажите созданную вами БД.

После этого появится SQLite Explorer(где он будет, зависит от того, где у вас расположена рабочая область Explorer). Выберете свою БД и откроется файл SQLite (рис. 2.51, 2.52).

На данном этапе стоит снова вернуться к разделу 1.6 SQLite в теоретических выкладках и посмотреть на структуру запросов к БД, чтобы освежить память. Далее работа будет строиться на предположении, что указанные запросы вы знаете.

Для выполнения запроса в среде расширения SQLite необходимо следующее (рис. 2.53):

1) создать новый запрос к БД;
2) выделить запрос, выполнение которого вам требуется;
3) нажать клавиши Ctrl + Shift + P;
4) в открывшемся терминале ввести – SQLite:
5) Run selected query(после первого выполнения запрос будет в списке последних использованных).

```
balzhit@balzhit-main:~$ cd 'vs code project'/
balzhit@balzhit-main:~/vs code project$ ls
Program_Balzhit_shit.cs  'vs code project.sln'   WebApplication   WebApplication1
balzhit@balzhit-main:~/vs code project$ cd WebApplication1
balzhit@balzhit-main:~/vs code project/WebApplication1$ ls
appsettings.Development.json   appsettings.json   bin   database   obj   Program.cs   Properties   '-- SQLite.sql'   WebApplication1.csproj
balzhit@balzhit-main:~/vs code project/WebApplication1$ cd database/
balzhit@balzhit-main:~/vs code project/WebApplication1/database$ ls
user_DB.db
balzhit@balzhit-main:~/vs code project/WebApplication1/database$ sqlite3 user_DB.db
SQLite version 3.45.1 2024-01-30 16:01:20
Enter ".help" for usage hints.
sqlite> databases
   ...>
```

Рис. 2.49. Создание базы данных

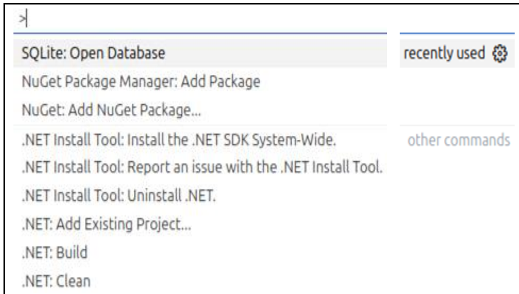


Рис. 2.50. Терминал команд оболочки VSCode

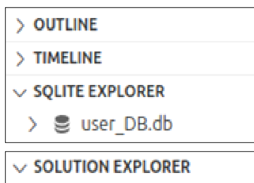


Рис. 2.51. Доступные базы данных

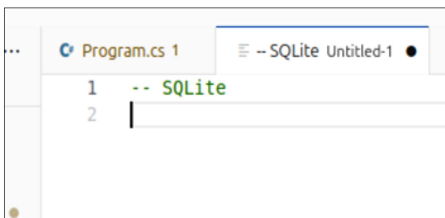


Рис. 2.52. Файл SQLite

После выполнения запроса откроется оболочка SQLite и выведет результат запроса. Учтите, что не все запросы имеют явно выраженный результат. Например, из указанных трех запросов только последний выведет таблицу пользователей (рис. 2.54).

БД готова, продолжаем работу. Теперь необходимо подключить SQLite к нашему проекту. Для этого снова открываем терминал (Ctrl + Shift + P) и выбираем установку пакетов менеджером NuGet (рис. 2.55), если возникает ошибка, вернитесь к разделу с NuGet и проверьте заметку.

```
Program.cs 1    -- SQLite Untitled-1
1   -- SQLite
2   CREATE TABLE users(
3   id INTEGER PRIMARY KEY,
4   email TEXT NOT NULL,
5   password TEXT NOT NULL
6   );
7
8   INSERT INTO users (id, email, password) VALUES (1, 'user', '1234')
9
10  SELECT * FROM users
```

Рис. 2.53. Запросы к базе данных

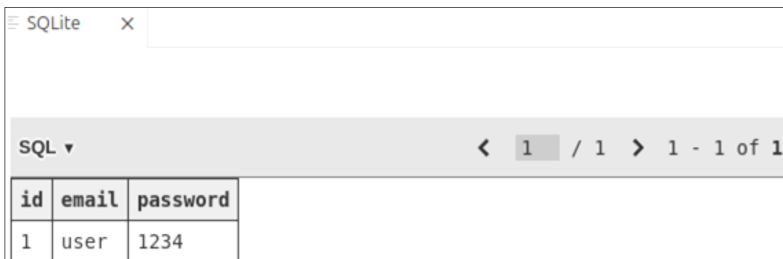


Рис. 2.54. Вывод таблицы пользователей

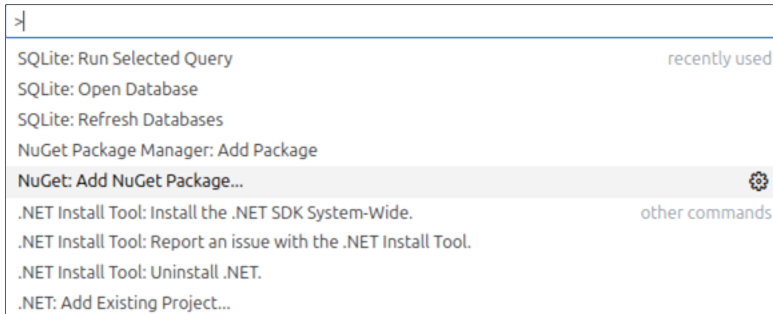


Рис. 2.55. Выбор установки пакетов менеджером NuGet в терминале

В следующем окне вводим SQLite и ждем результат запроса. Среди результатов надо найти Microsoft.Data.Sqlite и установить его. Сразу после установки необходимо подключить его к проекту. Если вы все сделали правильно, то в файле НАЗВАНИЕ_ПРОЕКТА.csproj (файл-проект) появится запись о подключенном пакете (рис. 2.56).

```
WebApplication1 > WebApplication1.csproj
1  <Project Sdk="Microsoft.NET.Sdk.Web">
2    <PropertyGroup>
3      <TargetFramework>net8.0</TargetFramework>
4      <Nullable>enable</Nullable>
5      <ImplicitUsings>enable</ImplicitUsings>
6    </PropertyGroup>
7    <ItemGroup>
8      <PackageReference Include="Microsoft.AspNetCore.Authentication.JwtBearer" Version="8.0.8"/>
9      <PackageReference Include="Microsoft.Data.Sqlite" Version="8.0.8"/>
10   </ItemGroup>
11 </Project>
```

Рис. 2.56. Запись о подключенном пакете в файл-проекте

*Заметка*: для удобства работы с БД можете установить расширение на sqlite – sqlite browser. Поскольку в БД предполагается хранение только авторизационных данных, авторы пособия посчитали эту утилиту избыточной.

## 2.9. Модификация WEBApplication

Для дальнейшего удобства вернем старый шаблон (рис. 2.57) и уберем все лишнее, оставив только скелет (рис. 2.58).

Далее модифицируем обработку запроса MapGet: пропишем маршрут запроса, входные параметры и лямбда-функцию или то, что запрос GET сгенерирует и передаст в ответ (рис. 2.59).

```
   0 references
1  internal class Program
2  {
       0 references
3      private static void Main(string[] args)
4      {
5          var builder = WebApplication.CreateBuilder(args);
6
7          // Add services to the container.
8          // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
9          builder.Services.AddEndpointsApiExplorer();
10         builder.Services.AddSwaggerGen();
11
12         var app = builder.Build();
13
14         // Configure the HTTP request pipeline.
15         if (app.Environment.IsDevelopment())
16         {
17             app.UseSwagger();
18             app.UseSwaggerUI();
19         }
20
21         app.UseHttpsRedirection();
```

Рис. 2.57. Старый шаблон

```
Hello world    Program.cs ×
ebApp > Program.cs > Program > Main
    0 references
1   internal class Program
2   {
        0 references
3       private static void Main(string[] args)
4       {
5           var builder = WebApplication.CreateBuilder(args);
6           var app = builder.Build();
7
8           var summaries = new[]
9           {
10              "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
11          };
12
13          app.MapGet("/weatherforecast", () =>
14          {
15              var forecast =  Enumerable.Range(1, 5).Select(index =>
16                  new WeatherForecast
17                  (
18                      DateOnly.FromDateTime(DateTime.Now.AddDays(index)),
19                      Random.Shared.Next(-20, 55),
20                      summaries[Random.Shared.Next(summaries.Length)]
21                  ))
22                  .ToArray();
23              return forecast;
24          });
25
26          app.Run();
27      }
28  }
29
    1 reference
30  record WeatherForecast(DateOnly Date, int TemperatureC, string? Summary)
31  {
        0 references
32      public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);
33  }
```

Рис. 2.58. Изменение старого шаблона

```
app.MapGet("/weatherforecast", () =>
{
    var forecast =  Enumerable.Range(1, 5).Select(index =>
        new WeatherForecast
        (
            DateOnly.FromDateTime(DateTime.Now.AddDays(index)),
            Random.Shared.Next(-20, 55),
            summaries[Random.Shared.Next(summaries.Length)]
        ))
        .ToArray();
    return forecast;
});
```

Рис. 2.59. MapGet

Аналогичным способом можно модифицировать обработки всех запросов: app.MapPost, app.MapPatch и прочих. Пример передачи параметров на рис. 2.60.

```
app.MapGet("/weatherforecast", (int start, int end) =>
{
    var forecast =  Enumerable.Range(1, 5).Select(index =>
        new WeatherForecast
        (
            DateOnly.FromDateTime(DateTime.Now.AddDays(index)),
            Random.Shared.Next(start, end),
            summaries[Random.Shared.Next(summaries.Length)]
        ))
        .ToArray();
    return forecast;
});
```

Рис. 2.60. Передача параметров

Теперь в http-запросе надо передать 2 параметра – start и end, а температура будет генерироваться в промежутке от start до end градусов. Представление в Postman на рис. 2.61.

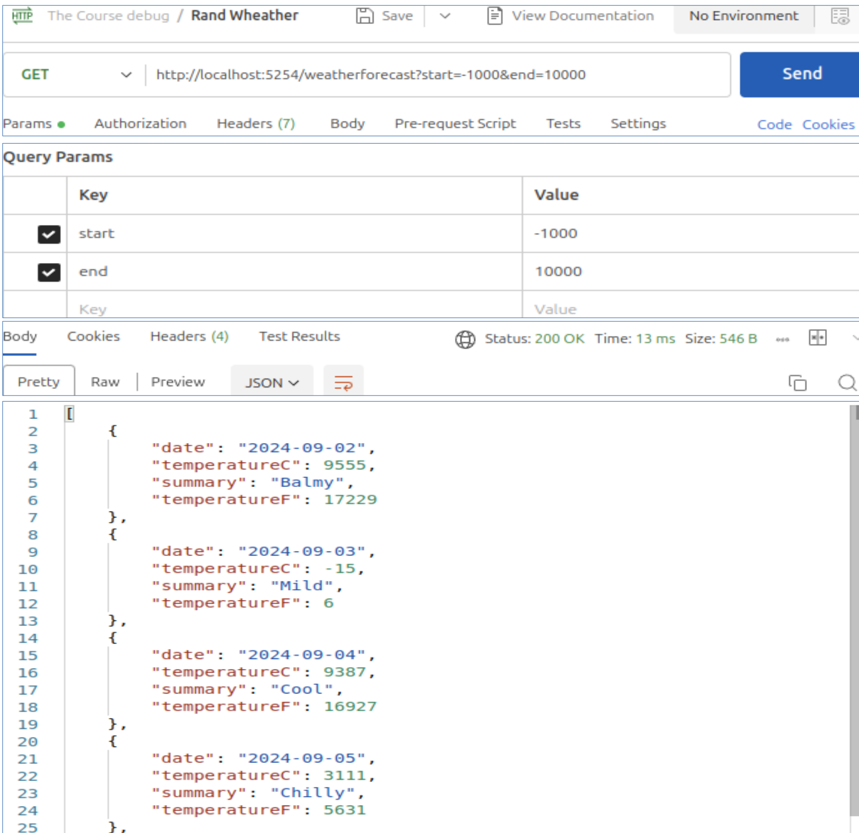


Рис. 2.61. Отображение в Postman

В примере прогноз генерируется в теле main. Согласно REST API это неприемлемо, поскольку один код не может отвечать за весь функционал. Следует вывести генерацию прогноза за пределы класса Program (лучше, конечно, отдельными подпроектами, но поскольку для начала подойдет отдельный класс).

Предположим, что есть некий код бэкэнда, который в ответ на запрос генерирует случайный прогноз погоды (рис. 2.62).

Сама реализация нам не важна, главное, что есть функция, которая возвращает какое-то значение. Теперь можно запросить результат функции через GET (рис. 2.63, 2.64).

```
2 references
class ForecastMaker
{
    2 references
    public bool is_auth = false;
    5 references
    private DateTime dateNow  = DateTime.Now;
    0 references
    public DateTime ChangeDateNow()…
    2 references
    public DateTime GetDate()…
    2 references
    public string GetForecast()…
    0 references
    public DateTime PatchDateNow(string date)…
    1 reference
    public string MakeJson()…
    1 reference
    private string GetRandomForecast()…
    2 references
    private int GetSeason()…

}
```

Рис. 2.62. Код с генерация случайного прогноза погоды

```
var app = builder.Build();
ForecastMaker forecast = new ForecastMaker();
var summaries = new[]
{
    "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
};

app.MapGet("/weatherforecast", () =>forecast.GetForecast().ToString());

app.MapPost("/input", async (string? returnUrl, HttpContext context) =>…

app.MapGet("/", async (string? returnUrl, HttpContext context) => {…
);

app.Run();
```

Рис. 2.63. Запрос результата функции через GET

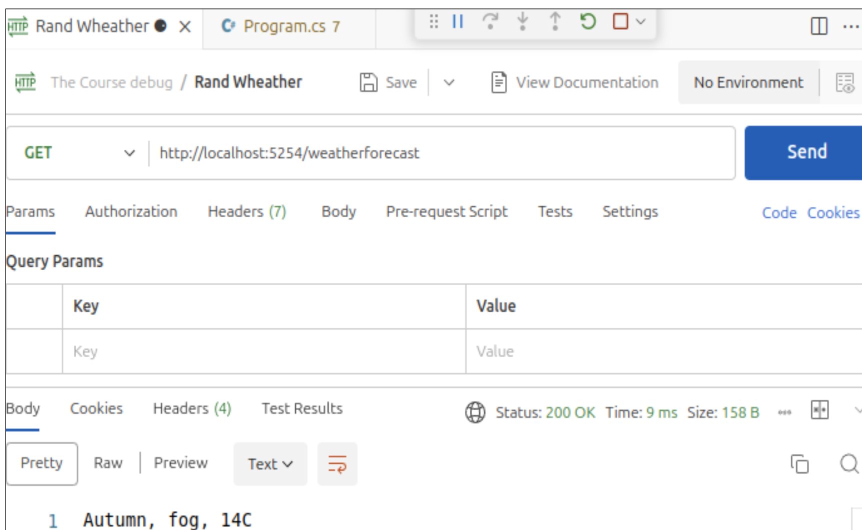


Рис. 2.64. Результат запроса в Postman

## 2.10. Реализация авторизации. Логин и пароль

Первый вариант реализации – авторизация по паролю и логину. Как уже говорилось ранее, неудачный способ и имеет существенные проблемы с безопасностью, рассмотрим его для ознакомления с классической авторизацией. Общепринято передавать логины и пароли через запрос POST. Заменяем наш POST так, как на рис. 2.65.

```
app.MapPost("/input", async (string? returnUrl, HttpContext context) =>
{
    // получаем из формы email и пароль
    var form = context.Request.Form;
    // если email и/или пароль не установлены, посылаем статусный код ошибки 400
    if (!form.ContainsKey("email") || !form.ContainsKey("password"))
        return Results.BadRequest("Email и/или пароль не установлены");

    string email = form["email"];
    string password = form["password"];

    if (email == "user" && password == "password") {
        forecast.is_auth = true;
        return Results.Ok();
    }

    return Results.StatusCode(403);
});
```

Рис. 2.65. Замена POST

В данном коде сделано все по-простому, пароль и логин сравниваются с конкретным значением – user и password. Но, считается, что сравнение двух строчек из какого-нибудь списка вы осуществить сможете.

Обратите внимание на код, и на то, как мы обращаемся к e-mail и password. form.ContainsKey означает, что мы работаем с ключами. Теперь к запросу от Postman. Запрос в случае передачи данных в ключах будет выглядеть так, как на рис. 2.66.

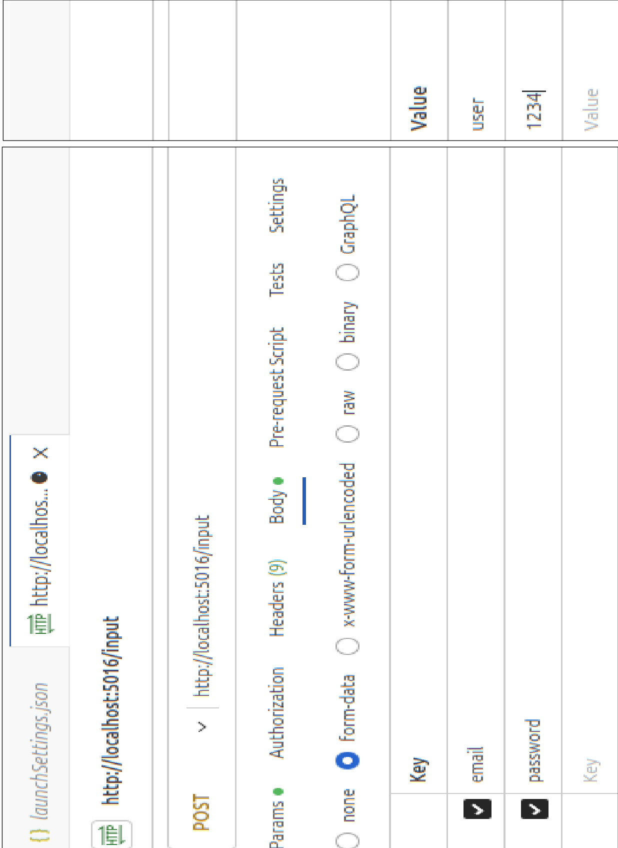


Рис. 2.66. Отображение в Postman

Ключи – это данные в теле запроса. Теперь посмотрим на ответ в формате json. На новый запрос GET будет выдаваться ответ в формате json (рис. 2.67, 2.68).

```
app.MapGet("/", async (string? returnUrl, HttpContext context) => {
    // получаем из формы email и пароль
    if (!context.Request.HasFormContentType)
            return Results.BadRequest();

    var form = context.Request.Form;
    // если email и/или пароль не установлены, посылаем статусный код ошибки 400
    if (!form.ContainsKey("email") || !form.ContainsKey("password"))
        return Results.BadRequest("Email и/или пароль не установлены");

    string email = form["email"];
    string password = form["password"];

    if (email == "user" && password == "password") {
        forecast.is_auth = true;
        return Results.Content(forecast.MakeJson());
    }

    return Results.Unauthorized();
}
);
```

Рис. 2.67. Замена POST

The Course debug / Rand Wheather    Save    View Documentation    No Environment

GET    http://localhost:5254/    Send

Params    Authorization    Headers (9)    Body    Pre-request Script    Tests    Settings    Code    Cookies

none    form-data    x-www-form-urlencoded    raw    binary    GraphQL

| | Key | Value |
|---|---|---|
| ☑ | email | user |
| ☑ | password | password |
| | Key | Value |

Body    Cookies    Headers (4)    Test Results    Status: 200 OK    Time: 2 ms    Size: 210 B

Pretty    Raw    Preview    JSON

```
1 {
2     "Date": "2024-09-01T00:00:00+07:00",
3     "TemperatureCelsius": "Autumn, hail, 8C"
4 }
```

Рис. 2.68. Ответ в формате json

Код создания JSON-объекта представлен на рис. 2.69. Не забудьте подключить библиотеку System.Text.Json.

```
public string MakeJson()
{
    var weatherForecast = new WeatherForecast
    {
        Date = dateNow.Date,
        TemperatureCelsius = GetForecast(),
    };

    string jsonString = JsonSerializer.Serialize(weatherForecast);

    return jsonString;
}
```

Рис. 2.69. Код создания JSON-объекта

## 2.11. Реализация авторизации. Токен

Для авторизации токеном был выбран токен JWT. Предварительно надо его установить: через пакетный менеджер (Ctrl+Shift+P -> NuGET:add package ->Microsoft.AspNetCore.Authentication.JwtBearer) или через терминал командой (dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer --version 8.0.8), затем подключением его к проекту. Если все правильно подключено, то он появится в файл-проекте (рис. 2.70).

```
1 <Project Sdk="Microsoft.NET.Sdk.Web">
2   <PropertyGroup>
3     <TargetFramework>net8.0</TargetFramework>
4     <Nullable>enable</Nullable>
5     <ImplicitUsings>enable</ImplicitUsings>
6   </PropertyGroup>
7   <ItemGroup>
8     <PackageReference Include="Microsoft.AspNetCore.Authentication.JwtBearer" Version="8.0.8"/>
9     <PackageReference Include="Microsoft.Data.SqlClient" Version="5.2.2"/>
0     <PackageReference Include="Microsoft.Data.Sqlite" Version="8.0.8"/>
1   </ItemGroup>
2 </Project>
```

Рис. 2.70. JWT в файл-проекте

В предыдущей части мы сравнивали логины и пароли напрямую. Сейчас сделаем подводку к БД, а именно – сделаем импровизированную БД (рис. 2.71).

```
var people = new List<Person>
{
    new Person("user", "1234"),
    new Person("non_user", "4321")
};
var builder = WebApplication.CreateBuilder(args);
```

Рис. 2.71. База данных в коде

Для работы с JWT вам понадобится примерно такой список библиотек:

1) using Microsoft.AspNetCore.Authentication.JwtBearer;
2) using Microsoft.AspNetCore.Authorization;
3) using Microsoft.IdentityModel.Tokens;
4) using System.IdentityModel.Tokens.Jwt;
5) using System.Security.Claims;
6) using System.Text.

К нашему коду надо добавить авторизацию и тип аутентификации (рис. 2.72).

```
builder.Services.AddAuthorization();
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            // указывает, будет ли валидироваться издатель при валидации токена
            ValidateIssuer = true,
            // строка, представляющая издателя
            ValidIssuer = AuthOptions.ISSUER,
            // будет ли валидироваться потребитель токена
            ValidateAudience = true,
            // установка потребителя токена
            ValidAudience = AuthOptions.AUDIENCE,
            // будет ли валидироваться время существования
            ValidateLifetime = true,
            // установка ключа безопасности
            IssuerSigningKey = AuthOptions.GetSymmetricSecurityKey(),
            // валидация ключа безопасности
            ValidateIssuerSigningKey = true,
        };
    });
```

Рис. 2.72. Добавление авторизации и типа аутентификации

Параметры токена представлены отдельный классом (рис. 2.73).

```
6 references
public class AuthOptions
{
    2 references
    public const string ISSUER = "MyAuthServer"; // издатель токена
    2 references
    public const string AUDIENCE = "MyAuthClient"; // потребитель токена
    1 reference
    const string KEY = "mysupersecret_secretsecretsecretkey!123";   // ключ для шифрации
    2 references
    public static SymmetricSecurityKey GetSymmetricSecurityKey() =>
        new SymmetricSecurityKey(Encoding.UTF8.GetBytes(KEY));
}
```

Рис. 2.73. Класс для параметров токена

Таким образом у генератора нашего токена на сервере есть следующие параметры:

– издатель;
– владелец токена;
– сид генерации случайной величины;
– функция генерации случайной величины.

Два последних пункта должны храниться в секрете. Затем зададим серверу настройки авторизации и аутентификации (рис. 2.74).

```
app.UseDefaultFiles();
app.UseStaticFiles();

app.UseAuthentication();
app.UseAuthorization();
```

Рис. 2.74. Настройки авторизации и аутентификации для сервера

Следующим шагом требуется переопределить результат запроса POST (рис. 2.75).

В ответ на запрос сервер будет посылать json-файл, в котором хранится токен и почта пользователя (рис. 2.76).

После переопределим запрос GET (рис. 2.77).

Особенно интересен в этом запросе параметр [Authorization]. Этот параметр относится к разделу заголовков запросов. В нем прописывается способ авторизации и токен (рис. 2.78).

```
// находим пользователя
Person? person = people.FirstOrDefault(p => p.Email == loginData.Email && p.Password == loginData.Password);
// если пользователь не найден, отправляем статусный код 401
//if(!auth) return Results.Unauthorized();
if(person is null) return Results.Unauthorized();
var claims = new List<Claim> {new Claim(ClaimTypes.Name, loginData.Email) };
// создаем JWT-токен
var jwt = new JwtSecurityToken(
        issuer: AuthOptions.ISSUER,
        audience: AuthOptions.AUDIENCE,
        claims: claims,
        expires: DateTime.UtcNow.Add(TimeSpan.FromMinutes(2)),
        signingCredentials: new SigningCredentials(AuthOptions.GetSymmetricSecurityKey(), SecurityAlgorithms.HmacSha256));
var encodedJwt = new JwtSecurityTokenHandler().WriteToken(jwt);

// формируем ответ
var response = new
{
    access_token = encodedJwt,
    username = loginData.Email
};

return Results.Json(response);
```

Рис. 2.75. Переопределение результата запроса POST

New Collection / http://localhost:3000/

POST http://localhost:3000/login

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2     "email" : "user",
3     "password" : "1234"
4 }
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview JSON

```
1 {
2     "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
          eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1lIjoidXNlciIsImV4cC
3     "username": "user"
4 }
```

Рис. 2.76. Ответ сервера в виде json-файла

```
70        app.MapGet("/data", [Authorize] () => new { message= "Hello World!" });
71 >      {…
```

Рис. 2.77. Переопределение запроса GET

## 2.12. Работа с SQLite

SQLite уж подключен, данные в базу записаны. Теперь найдем пользователя в базе и проведем авторизацию по данным из нее (рис. 2.79).

.ExecuteReader() используется, если ожидается ответ в виде нескольких строк, т.е. с запросом SELECT. В иных случаях используйте ExecuteNonQuery() или ExecuteScalar().

Передача запроса серверу и ответ его аналогичен предыдущему пункту.

New Collection / http://localhost:3000/

GET http://localhost:3000/data

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Type Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about authorization ↗

Token eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9....

Рис. 2.78. Параметр [Authorization]

```
app.MapPost("/login", (Person loginData) =>
{
    bool auth = false;
    string sqlExpression = "SELECT email,password FROM users WHERE email = '"+loginData.Email.ToString() + "' AND password = '"+ loginData.Password.ToString()+"'";
    using (var connection = new SqliteConnection("Data Source=database/user_DB.db"))
    {
        connection.Open();

        SqliteCommand command = new SqliteCommand(sqlExpression, connection);
        using (SqliteDataReader reader = command.ExecuteReader())
        {
            if (reader.HasRows) // если есть данные
            {
                auth = true;
            }
        }
    }
```

Рис. 2.79. Авторизация по данным из базы данных

## 2.13. Создание консольного приложения клиента

В качестве клиента должно быть реализовано консольное приложение. Создадим новый проект C# (или подпроект в текущем). На сервере уже реализована авторизация, потому первый запрос должен быть запросом на авторизацию (рис. 2.80).

```
static Token? LoginOnServer (string path, string email, string password)
{
    Token? token = null;// создание нулевого токена

    var LoginData = new Person//шаблон под данные для json-контейнера
    {
        Email = email,
        Password = password,
    };

    HttpResponseMessage response = client.PostAsJsonAsync<Person>(path, LoginData).Result;
    //отправка запроса - автоматическая серелиализация var LoginData->JSON, создание пакета
    //http с контейнером json. Ответ помещается в response
    if (response.IsSuccessStatusCode)// проверка статус-кода
    {
        token = response.Content.ReadFromJsonAsync<Token>().Result;
        //считывание токена из ответа в нулевой токен
    }

    return token;
}
```

Рис. 2.80. Запрос на авторизацию

Для этого запроса уже должны быть реализованы классы Token и Person (рис. 2.81). Вызов метода представлен на рис. 2.82.

Для работы с этими функциями требуется подключить следующие библиотеки:

1) using System.Net.Http.Headers;

2)using System.Text.Json;

3) using System.Net.Http.Json.

На сервере реализован маршрут /weatherforecast с GET запросом. Теперь сформируем GET запрос на клиенте (рис. 2.83).

Сам запрос в теле main будет выглядеть примерно так, как представлено на рис. 2.84.

```
namespace HttpClientSample
{
    public class Token
    {
        public required string username {get; set; }
        public required string access_token {get; set; }
    }
    public class Person
    {
        public required string Email {get; set; }
        public required string Password {get; set; }
    }
    class Program
    {
        static HttpClient client = new HttpClient();
```

Рис. 2.81. Классы Token и Person

```
static void Main()
{
    // Update port # in the following line.
    client.BaseAddress = new Uri("http://localhost:5254/");
    client.DefaultRequestHeaders.Accept.Clear();
    //очищение значений заголовков Accept по умолчанию
    client.DefaultRequestHeaders.Accept.Add(
        new MediaTypeWithQualityHeaderValue("application/json"));
        //добавления заголовка для формата JSON

    try
    {
        string url = "/login";
        Token? token = LoginOnServer(url,"user", "password");
        if (token == null) {
            throw new Exception("Auth failed");
        }
```

Рис. 2.82. Вызов метода

```
static string GetProductAsync(string path, Token token)
{
    string product = "Not available";
    client.DefaultRequestHeaders.Authorization =
    new AuthenticationHeaderValue("Bearer", token.access_token);
    //прописываем аутентификатор - токен
    HttpResponseMessage response = client.GetAsync(path).Result;
    if (response.IsSuccessStatusCode)
    {
        product = response.Content.ReadAsStringAsync().Result;
    }
    return product;
}
```

Рис. 2.83. GET-запрос на клиенте

```
url = "/weatherforecast";
string foreCast = GetProductAsync(url, token);

Console.WriteLine(foreCast);
```

Рис. 2.84. Сам запрос в теле main

## 2.14. Пример реализации клиент-серверного приложения

Представим вариант, в котором надо реализовать функционал генератора случайных чисел в указанном диапазоне. Для него определим следующие запросы:

– регистрация пользователя (POST) – в ответе должен быть технический токен среди прочего;

– изменить пароль (должен менять и токен) (PATCH);

– указать новые границы генерации (PATCH);

– получение случайного числа в дефолтном диапазоне(GET);

– получение случайного числа в указанном диапазоне(GET);

Для этого варианта был реализован класс RandomGenerator, который возвращает случайное число (рис. 2.85).

Этот класс будет представлять из себя бэкэнд курсового проекта. Этот класс возвращает результаты выполнения своих методов в виде числового ответа (или булевое число). Для него можно разработать сервер для обработки запросов формата HTTP.

Эта часть не является обязательной и может быть объединена с кодом выше для уменьшения объема кода (рис. 2.86). В этом классе для компоновки ответа используется структура RGResult (рис. 2.87).

Этот класс представляет собой генератор чисел по заданным границам. Границы определяются по двум приватным полям low_border и up_border. Метод GetValue реализован с перегрузкой, первый вариант использует для генерации установленные значения атрибутов low_border и up_border, второй принимает на вход два числа. Для обновления приватных low_border и up_border используется метод UpdateBorder.

В вашем курсовой вместо этого класса RandomGenerator необходимо реализовать класс, соответствующий вашему варианту. Следует отметить, что для подобных классов рекомендуется использовать именно приватные поля, поскольку внешнее обновление данных может быть нежелательным и может привести к некорректной работе кода. Также не стоит забывать про проверку корректности полей.

```
Ссылок: 3
public class RandomGenerator {
    private int low_border;
    private int up_border;
    private Random random;

    Ссылок: 1
    public RandomGenerator(
        int lb = 0, int ub = 10
    ) {
        low_border = lb;
        up_border = ub;
        random = new Random();
    }

    Ссылок: 1
    public int GetValue() {
        return random.Next(low_border, up_border);
    }

    Ссылок: 1
    public int GetValue(int lb, int ub) {
        return random.Next(lb, ub);
    }

    Ссылок: 1
    public bool UpdateBorder(int lb, int ub) {
        if (ub <= lb)
            return false;

        low_border = lb;
        up_border = ub;

        return true;
    }
}
```

Рис. 2.85. Класс RandomGenerator

Для этого (их) классов реализация серверной части ПО представлена ниже (рис. 2.86–2.90).

Рисунок 2.86 демонстрирует реализацию класса адаптера для RandomGenerator. Этот подход является рекомендательным, если нет желания усложнять UML-диаграмму, можно объединить оба класса в 1. Адаптер служит только как оболочка, скрывая реализацию и ответы сервера, подменяя их на заготовленные HTTP-ответы.

```
Ссылок: 2
public class RGWebAdapter {
    private RandomGenerator rg = new RandomGenerator();
    public  DBManager db = new DBManager();
    Ссылок: 1
    public async Task<IResult> LogIn(string login, string password, HttpContext context) {
        if (db.CheckUser(login, password)) {
            var claims = new List<Claim> { new Claim(ClaimTypes.Name, login) };
            var claimsIdentity = new ClaimsIdentity(claims, "Cookies");
            await context.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme,
                        new ClaimsPrincipal(claimsIdentity));
            return Results.Ok();
        }

        return Results.Unauthorized();
    }

    Ссылок: 1
    public IResult GetValue() {
        return Results.Ok(new RGResult(rg.GetValue()));// используется RGResult - структура, приложена в следующем скрине
    }

    Ссылок: 5
    public IResult GetValue(int lb, int ub) {
        if (lb >= ub)
            return Results.Conflict("Can't generate value with such borders");

        return Results.Ok(new RGResult(rg.GetValue(lb, ub)));
    }

    Ссылок: 0
    public IResult UpdateBorder(int lb, int ub) {
        if (rg.UpdateBorder(lb, ub))
            return Results.Ok("Borders are updated successfully");

        return Results.Conflict("Can't update borders with such values");
    }
}
```

Рис. 2.86. Класс RGWebAdapter

```
Ссылок: 3
public struct RGResult
{
    Ссылок: 2
    public RGResult(int rv)
    {
        random_value = rv;
    }
    Ссылок: 1
    public int random_value { get; set; }
}
```

Рис. 2.87. Структура RGResult

```
using System.Security.Claims;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

using SixLabors.ImageSharp; // установить через NuGet, для работы с изображениями в запросе
using SixLabors.ImageSharp.PixelFormats;//

var builder = WebApplication.CreateBuilder(args);// инициализация builder

builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme).AddCookie();
builder.Services.AddAuthorization();
// установка настроек аутентификации и авторизации

var app = builder.Build(); // сборка приложения

app.UseAuthentication(); // установка параметров аутентификации и авторизации приложения по умолчанию
app.UseAuthorization();

RGWebAdapter rg = new RGWebAdapter(); // создание объекта класса адаптера

// если вы не реализовывали класс RGWebAdapter, а обработка запросов
// реализовали в RandomGenerator, то RandomGenerator rg = new RandomGenerator();

app.MapGet("/", () => "Ask random to generate random value");
app.MapGet("/random", [Authorize] () => rg.GetValue());// ответ на запрос GET без параметров,
                                                       // бекэнд выдает 2 сгенерированных случайных числа
                                                       // в пределах дефолтных значений границ
app.MapGet("/random/{low}/{up}", [Authorize] (int low, int up) => rg.GetValue(low, up)); // ответ на запрос GET с параметрами,
                                                                                          прописанными в маршруте запроса
app.MapGet("/random_params", [Authorize] (int low, int up) => rg.GetValue(low, up));// ответ на запрос GET с параметрами

app.MapGet("/check_user", [Authorize] (HttpContext context) => {
    if (context.User.Identity == null)
        return Results.BadRequest("User is unknown");
    return Results.Ok(context.User.Identity.Name);
                                                      });// ответ на запрос GET с проверкой параметров в контексте запроса
                                                       // проверка на аутентификацию (необязательно к реализации)
```

Рис. 2.88. Реализация запросов GET

```
41  app.MapPost("/signup", (string login, string password) => {
42      if (rg.db.AddUser(login, password)) // используется атрибут класса RGBAdapter - public  DBManager db. Его реализация будет представлена после сервера
43          return Results.Ok("User " + login + " registered successfull!");
44      else
45          return Results.Problem("Failed to register user " + login);
46  });                                                           //регистрация на сервере
47
48  app.MapPost("/random_params_form", [Authorize] ([FromForm] int low, [FromForm] int up) => rg.GetValue(low, up)).
49  DisableAntiforgery();                                         //запрос Post с использование параметров из формы (в отличие от предыдущего Get)
50
51  app.MapPost("/random_params_header", [Authorize] ([FromHeader] int low, [FromHeader] int up) => rg.GetValue(low, up));
52                                                                //запрос Post с использование параметров из заголовков (в отличие от предыдущего Get)
53
54  app.MapPost("/random_params_json", [Authorize] (string username, [FromBody] Borders bs) => // используется структура Borders
55      rg.GetValue(bs.low, bs.up));                              //запрос Post с использование параметров из тела запроса (в отличие от предыдущего Get)
56
57
58  app.MapPost("/login", (string login, string password, HttpContext context) => rg.LogIn(login, password, context));
59                                                                //авторизация на сервере
60
```

Рис. 2.89. Реализация запросов POST

```
app.MapPost("/picture", (HttpRequest request) => {
    try
    {
        var memoryStream = new MemoryStream();
        request.Body.CopyToAsync(memoryStream).Wait();
        memoryStream.Seek(0, SeekOrigin.Begin);
        Image image = Image.Load<Rgba32>(memoryStream);
        return Results.Ok("Received image " + image.Width + "x" + image.Height);
    }
    catch (Exception exp)
    {
        return Results.BadRequest("Wrong image format: " + exp.Message);
    }
});                                          //запрос POST для выгрузки изображения

app.MapPost("/picture_form", ([FromForm] UploadImageModel model) => { // используется класс UploadImageModel, представлен ниже
    if (model.picture == null || model.picture.Length == 0)
    {
        return Results.BadRequest(new { message = "Файл изображения не найден или пуст." });
    }

    try
    {
        var memoryStream = new MemoryStream();
        model.picture.CopyToAsync(memoryStream).Wait();
        memoryStream.Seek(0, SeekOrigin.Begin);
        Image image = Image.Load<Rgba32>(memoryStream);
        return Results.Ok("Received image " + image.Width + "x" + image.Height);
    }
    catch (Exception exp)
    {
        return Results.BadRequest("Wrong image format: " + exp.Message);
    }
}).DisableAntiforgery();                     //запрос POST для выгрузки изображения из формы
```

Рис. 2.90. Реализация запросов POST для изображения

Для запроса POST на /picture_form требуется класс UploadImageModel (рис. 2.91).

```
        Ссылок: 1
190     public class UploadImageModel
191     {
            Ссылок: 3
192         public IFormFile picture { get; set; } = default!;
193     }
```

Рис. 2.91. класс UploadImageModel

Основные запросы реализованы, далее подключаемся к БД и запускаем сервер (2.92). Для работы с базой данных реализован класс DBManager. его можно реализовать как в рамках Program.cs, так и как в отдельном файле проекта, как на рис. 2.93.

```
const string DB_PATH = "ПУТЬ ДО БАЗЫ ДАННЫХ/users.db";
if (!rg.db.ConnectToDB(DB_PATH)) { // подключение к БД
    Console.WriteLine("Failed to connect to db " + DB_PATH);
    Console.WriteLine("Shutdown!");
    return;
}
app.Run();// запуск серверного приложения
rg.db.Disconnect();// отключение от БД
```

Рис. 2.92. Подключение к базе данных

```
using System.Runtime.InteropServices;
using System.Security.Cryptography;
using System.Text;
using Microsoft.Data.Sqlite;

Ссылок: 2
public class DBManager {
    private SqliteConnection? connection = null;

    Ссылок: 2
    private string HashPassword(string password) {// подсчет хеш для пароля
        using (var algorithm = SHA256.Create()) {
            var bytes_hash = algorithm.ComputeHash(Encoding.Unicode.GetBytes(password));
            return Encoding.Unicode.GetString(bytes_hash);
        }
    }

    Ссылок: 1
    public bool ConnectToDB(string path) ...

    Ссылок: 1
    public void Disconnect() ...

    Ссылок: 1
    public bool AddUser(string login, string password) ...

    Ссылок: 1
    public bool CheckUser(string login, string password) ...
}
```

Рис. 2.93. Отдельный файл для работы с базой данных

Каждая функция подробнее представлена на рис. 2.94–2.97.

```
Ссылок: 1
public bool ConnectToDB(string path) {// подключение к БД
    Console.WriteLine("Connection to db...");

    try
    {
        connection = new SqliteConnection("Data Source=" + path);
        connection.Open();

        if (connection.State != System.Data.ConnectionState.Open) {
            Console.WriteLine("Failed!");
            return false;
        }
    }
    catch (Exception exp) {
        Console.WriteLine(exp.Message);
        return false;
    }

    Console.WriteLine("Done!");
    return true;
}
```

Рис. 2.94. Функция ConnectToDB

```
Ссылок: 1
public void Disconnect() {// отключение от БД
    if (null == connection)
        return;

    if (connection.State != System.Data.ConnectionState.Open)
        return;

    connection.Close();

    Console.WriteLine("Disconnect from db");
}
```

Рис. 2.95. Функция Disconnect

```
Ссылок: 1
public bool AddUser(string login, string password) {// Добавление пользователя в БД при регистрации
    if (null == connection)
        return false;
    if (connection.State != System.Data.ConnectionState.Open)
        return false;
    string REQUEST = "INSERT INTO users (Login, Password) VALUES ('" + login + "', '" + HashPassword(password) + "')";
    var command = new SqliteCommand(REQUEST, connection);
    int result = 0;
    try
    {
        result = command.ExecuteNonQuery();
    }
    catch (Exception exp) {
        Console.WriteLine(exp.Message);
        return false;
    }

    if (1 == result)
        return true;
    else
        return false;
}
```

Рис. 2.96. Функция AddUser

```
Ссылок: 1
public bool CheckUser(string login, string password) {// проверка наличия пользователя в БД при авторизации
    if (null == connection)
        return false;

    if (connection.State != System.Data.ConnectionState.Open)
        return false;
     string REQUEST = "SELECT Login,Password FROM users WHERE Login='" + login + "' AND Password = '"
                                                                + HashPassword(password) + "'";

    var command = new SqliteCommand(REQUEST, connection);
    try
    {
        var reader = command.ExecuteReader();

        if (reader.HasRows)
            return true;
        else
            return false;
    }
    catch (Exception exp) {
        Console.WriteLine(exp.Message);
        return false;
    }
}
```

Рис. 2.97. Функция CheckUser

Перед написанием клиентской части надо уточнить параметры запуска сервера. Их можно найти в файле launchsettings.json, в дереве зависимости проекта (рис. 2.98). В нем указаны адреса запуска приложения.

Рис. 2.98. Параметры запуска сервера в коде

Эти адреса будут использоваться клиентом для отправки запросов.

Клиентская часть должна иметь доступ ко всему функционалу сервера. На рис. 2.99 представлены функции клиента.

```
3   using System.Net;
4   using System.Net.Http.Headers;
5   using System.Net.Http.Json;
6   using System.Text;
7   using System.Text.Json;
8
9   CookieContainer cookies = new CookieContainer();//создание контейнера для куки клиента
10  HttpClientHandler handler = new HttpClientHandler();
11  HttpClient client = new HttpClient(handler);
12  handler.CookieContainer = cookies;// определяем куки контейнер во все будущие запросы
13
    Ссылок: 1
14  bool LoginOnServer(string? username, string? password)...
32
    Ссылок: 1
33  string GetRandomWOParams()...
44
    Ссылок: 1
45  string GetRandom(int low, int up)...
55
    Ссылок: 1
56  string GetRandomParams(int low, int up)...
66
    Ссылок: 1
67  string GetRandomParamsForm(int low, int up)...
82
    Ссылок: 1
83  string GetRandomParamsHeader(int low, int up)...
98
    Ссылок: 1
99  string GetRandomParamsJson(int _low, int _up)...
```

Рис. 2.99. Функции клиента

Эти функции должны быть модифицированы у каждого согласно варианту задания. Далее представлено подключение клиента к серверу (рис. 2.100) и примеры реализации запросов со стороны клиента.

На рис. 2.100 показаны начальные настройки соединения: адрес сервера и считывание логина и пароля с проверкой авторизации. Рисунки с 2.101 по 2.106 при правильной реализации серверной части будут выглядеть примерно так же. Особое внимание уделите сочетанию маршрута и типа запроса, поскольку неправильное сочетание не даст результата. Пара маршрут-запрос должны совпадать как на сервере, так и на клиенте.

```
99   string GetRandomParamsJson(int _low, int _up)[...]
117
118  const string DEFAULT_SERVER_URL = "http://localhost:5247"; // адрес сервера
119  Console.WriteLine("Введите адрес сервера (http://localhost:5247 по умолчанию):");
120  string? server_address = Console.ReadLine();// изменение адреса сервера, если вдруг нужно обратиться по другому маршруту
121  if (server_address == null || server_address.Length == 0) {
122      server_address = DEFAULT_SERVER_URL;
123  }
124
125  Console.WriteLine("Selected server {0}", server_address);
126  client.BaseAddress = new Uri(server_address);//установка URI сервера в качестве дефолтного для запросов клиента
127
128  Console.Write("Enter username: "); //ввод логина и пароля
129  string? username = Console.ReadLine();
130  Console.Write("Enter password: ");
131  string? password = Console.ReadLine();
132
133  try
134  {
135      if (!LoginOnServer(username, password))// попытка авторизации
136      {
137          Console.WriteLine("Login Failed");
138          return;
139      }
140
141      Console.WriteLine(GetRandomWOParams());// запросы клиента к серверу.
142      Console.WriteLine(GetRandom(100, 1000));// в этой части, вместо строк с 141 по 147 должен быть функционал по выбору варианта действий
143      Console.WriteLine(GetRandomParams(1000, 10000));
144
145      Console.WriteLine(GetRandomParamsForm(900, 999));
146      Console.WriteLine(GetRandomParamsHeader(500, 550));
147      Console.WriteLine(GetRandomParamsJson(1111, 2222));
148  }
149  catch (Exception e)
150  {
151      Console.WriteLine(e.Message);
152  }
```

Рис. 2.100. Подключение клиента к серверу

```
Ссылок: 1
string GetRandomWOParams() {// отправка без параметров
    string result = "Not available";
    string request = "/random";//маршрут запроса
    HttpResponseMessage response = client.GetAsync(request).Result;//отправка запроса и ожидание результата
    if (response.IsSuccessStatusCode)
    {
        result = response.Content.ReadAsStringAsync().Result;
    }

    return result;
}
```

Рис. 2.101. Запрос GET без параметров

```
Ссылок: 1
string GetRandom(int low, int up) {// запрос с параметрами, указанными в маршруте
    string result = "Not available";
    string request = "/random/" + low.ToString() + "/" + up.ToString();//формирование маршрута запроса с параметрами
    var response = client.GetAsync(request).Result;//отправка запроса и ожидание результата
    if (response.IsSuccessStatusCode) {
        result = response.Content.ReadAsStringAsync().Result;
    }

    return result;
}
```

Рис. 2.102. Запрос GET с параметрами, указанными в маршруте

```
Ссылок: 1
string GetRandomParams(int low, int up) {//запрос с параметрами
    string result = "Not available";
    string request = "/random_params?low=" + low.ToString() + "&up=" + up.ToString();//формирование маршрута запроса с параметрами
    HttpResponseMessage response = client.GetAsync(request).Result;//отправка запроса и ожидание результата
    if (response.IsSuccessStatusCode) {
        result = response.Content.ReadAsStringAsync().Result;
    }

    return result;
}
```

Рис. 2.103. Запрос с параметрами

```
Ссылок: 1
string GetRandomParamsForm(int low, int up) {//запрос с параметрами в форме запроса
    string result = "Not available";
    string request = "/random_params_form";

    var multipartContent = new MultipartFormDataContent();// добавление контейнера-формы в запрос
    multipartContent.Add(new StringContent(low.ToString()), "low");//добавление параметров в контейнер
    multipartContent.Add(new StringContent(up.ToString()),  "up");

    HttpResponseMessage response = client.PostAsync(request, multipartContent).Result;//отправка запроса и ожидание результата
    if (response.IsSuccessStatusCode) {
        result = response.Content.ReadAsStringAsync().Result;
    }

    return result;
}
```

Рис. 2.104. Запрос POST
с параметрами в контейнере-форме

```
Ссылок: 1
string GetRandomParamsHeader(int low, int up) {//запрос с параметрами в заголовках запроса
    string result = "Not available";
    string request_text = "/random_params_header";

    var request = new HttpRequestMessage(HttpMethod.Post, request_text);// изменение метода запроса с GET на POST
    request.Headers.Add("low", low.ToString());//добавление параметров в заголовки запроса
    request.Headers.Add("up",  up.ToString());

    HttpResponseMessage response = client.SendAsync(request).Result;//отправка запроса и ожидание результата
    if (response.IsSuccessStatusCode) {
        result = response.Content.ReadAsStringAsync().Result;
    }

    return result;
}
```

Рис. 2.105. Запрос POST с параметрами в заголовках

```
Ссылок: 1
string GetRandomParamsJson(int _low, int _up) {//запрос с параметрами в json-контейнере, помещенному в тело запроса
    string result = "Not available";
    string request = "/random_params_json";

    var data = new {// формирование структуры json
        low = _low,
        up = _up,
    };
    string jsonBody = JsonSerializer.Serialize(data);// сериализация данных в jsonBody
    var content = new StringContent(jsonBody, Encoding.UTF8, "application/json");// помещение сформированного json в тело запроса

    HttpResponseMessage response = client.PostAsync(request, content).Result;//отправка запроса и ожидание результата
    if (response.IsSuccessStatusCode) {
        result = response.Content.ReadAsStringAsync().Result;
    }

    return result;
}
```

Рис. 2.106. Запрос POST с параметрами в json-контейнере в теле запроса

## Литература

1. Nottingham M. Building Protocols with HTTP: Request for Comments RFC 9205. Internet Engineering Task Force, 2022. – 27 p.

2. Nielsen H. et al. Hypertext Transfer Protocol – HTTP/1.1: Request for Comments RFC 2616. Internet Engineering Task Force, 1999. – 176 p.

3. Berners-Lee T., Fielding R.T., Masinter L.M. Uniform Resource Identifier (URI): Generic Syntax: Request for Comments RFC 3986. Internet Engineering Task Force, 2005. – 61 p.

4. Bray T. The JavaScript Object Notation (JSON) Data Interchange Format: Request for Comments RFC 8259. Internet Engineering Task Force, 2017. – 16 p.

*Учебно-методическое издание*

# ОСНОВЫ ПРОГРАММИРОВАНИЯ

***Б.С. Лодонова, С.С. Харченко***
***(составители)***

*Учебно-методическое пособие по курсовой работе*

Верстка – В.М. Бочкаревой
Печатается без корректуры, в авторской редакции

---

В-Спектр (ИП Бочкарева В.М.)
Подписано к печати 26.02.2026.
Формат $60\times84^{1}/_{16}$. Печать трафаретная.
Печ. л. 5,6. Тираж 100 экз. Заказ 1.

---

Тираж отпечатан ИП В.М. Бочкаревой
ИНН 701701817754
634055, г. Томск, пр. Академический, 13-24, тел. 8-905-089-92-40
Эл. почта: bvm-1@list.ru