Шпаргалка для Junior .NET разработчика

Краткое руководство для подготовки к собеседованию на C# Backend

Для начинающих разработчиков, изучающих .NET и C#

Содержание

1 Введение

Шпаргалка предназначена для подготовки к собеседованию на позицию Junior .NET разработчика с фокусом на C# и бэкенд-разработку. Она структурирует ключевые знания по языку C#, платформе .NET, веб-разработке, базам данных, тестированию, версионному контролю и развертыванию, чтобы помочь быстро повторить материал. Также включены частые вопросы на собеседованиях с развёрнутыми ответами, которые звучат естественно и демонстрируют понимание темы, как если бы вы отвечали на интервью. **Требуется**: Знание основ программирования, HTTP, JSON.

2 Основы C#

- **Синтаксис**: Переменные (`int`, `string`), циклы (`for`, `while`), условия (`if`).
- **ООП**: Классы, наследование, полиморфизм, инкапсуляция.
- **Коллекции**: `List<T>`, `Dictionary<TKey, TValue>`, LINQ (`Where`, `Select`).
- **Асинхронность**: `async`/`await` для неблокирующих операций.
- **Исключения**: `try-catch`, пользовательские исключения.

3 .NET и ASP.NET Core

- **.NET Core**: Кросс-платформенная платформа, NuGet для пакетов.
- **ASP.NET Core**: MVC, Web API, middleware, внедрение зависимостей (DI).
- **Конфигурация**: `appsettings.json`, переменные окружения.

4 Веб-разработка

- **HTTP**: Методы (GET, POST), статус-коды (200, 404), заголовки.
- **REST API**: Ресурсы, URI, JSON.
- **Аутентификация**: JWT, OAuth.

5 Базы данных

- **SQL**: Запросы (`SELECT`, `JOIN`), нормализация.
- **Entity Framework Core**: ORM для работы с базами через C#.

6 Тестирование

- **Юнит-тесты**: xUnit, NUnit для проверки кода.
- **Интеграционные тесты**: Проверка взаимодействия компонентов.

7 Версионный контроль

- **Git**: `commit`, `push`, `pull`, `branch`, `merge`.

8 Развертывание

- **IIS**: Веб-сервер для хостинга.
- **Docker**: Контейнеризация приложений.
- **CI/CD**: Автоматизация с Azure DevOps, GitHub Actions.

9 Частые вопросы на собеседовании

- **Что такое разница между `==` и `Equals()` в C#?** Оператор `==` сравнивает значения для значимых типов или ссылки для ссылочных типов, тогда как метод `Equals()` позволяет определить пользовательскую логику сравнения, часто проверяя содержимое объектов. Например, для строк `==` и `Equals()` сравнивают содержимое, но для пользовательских классов `Equals()` можно переопределить, чтобы сравнивать поля, а `==` по умолчанию проверяет только ссылки. Это делает `Equals()` более гибким для сложных объектов.
- **Объясните сборку мусора в .NET.** Сборка мусора в .NET автоматически освобождает память, занятую объектами, на которые больше нет ссылок, предотвращая утечки памяти. Она работает в фоновом режиме, периодически сканируя кучу и удаляя неиспользуемые объекты, что упрощает управление памятью для разработчиков. Это особенно полезно в больших приложениях, где ручное управление памятью было бы сложным.
- **Что такое делегаты в C#?** Делегаты — это типобезопасные указатели на методы, которые позволяют передавать функции как параметры или использовать их для событий. Они широко применяются для реализации колбэков и событий, например, в обработчиках кнопок в UI или асинхронных операциях. Это делает код более модульным и гибким, так как методы можно динамически привязывать к делегатам.
- **В чём разница между абстрактным классом и интерфейсом в C#?** Абстрактный класс может содержать как реализованные, так и абстрактные методы, а также поля и конструкторы, но класс может наследовать только один абстрактный класс. Интерфейс определяет только сигнатуры методов, свойств или событий, и класс может реализовать несколько интерфейсов, что делает их идеальными для множественного наследования поведения. Например, интерфейсы часто используются для определения контрактов в DI, а абстрактные классы — для общей логики в иерархии классов.
- **Как работает внедрение зависимостей (DI) в ASP.NET Core?** Внедрение зависимостей в ASP.NET Core позволяет передавать зависимости (например, сервисы) в классы через конструкторы, вместо их создания внутри. Контейнер DI, встроенный в ASP.NET Core, автоматически создаёт и предоставляет экземпляры зависимостей, что упрощает тестирование и замену компонентов. Например, сервис для работы с базой данных можно внедрить в контроллер, не создавая его вручную.
- **Что такое middleware в ASP.NET Core?** Middleware в ASP.NET Core — это компоненты, которые обрабатывают HTTP-запросы и ответы в конвейере приложения, выполняя задачи вроде аутентификации или логирования. Каждый middleware может изменить запрос или ответ и передать его следующему компоненту в цепочке. Это позволяет гибко настраивать обработку запросов, например, добавляя проверку токенов перед вызовом контроллера.

- **Что такое нормализация в базах данных?** Нормализация — это процесс организации данных в базе для устранения избыточности и обеспечения целостности, разделяя их на таблицы с уникальными ключами. Она помогает избежать дублирования данных и упрощает обновление информации, например, храня данные о клиентах и заказах в отдельных таблицах, связанных через внешние ключи. Обычно применяются три нормальные формы, но в реальных проектах балансируют между нормализацией и производительностью.
- **В чём разница между INNER JOIN и LEFT JOIN в SQL?** INNER JOIN возвращает только строки, где есть совпадения в обеих таблицах, что полезно для получения связанных данных, например, заказов и их клиентов. LEFT JOIN возвращает все строки из левой таблицы и соответствующие строки из правой, заполняя NULL там, где совпадений нет, что удобно для анализа всех клиентов, даже если у них нет заказов. Это влияет на выбор типа JOIN в зависимости от задачи, например, для отчётов или фильтрации.
- **Что такое Entity Framework Core?** Entity Framework Core — это ORM (объектно-реляционный отображатель), который позволяет работать с базами данных через объекты C#, минимизируя написание SQL-запросов. Он поддерживает LINQ для запросов, миграции для управления схемой базы и работает с разными СУБД, такими как SQL Server или PostgreSQL. Это упрощает разработку, но требует понимания, как оптимизировать запросы для производительности.
- **Как вы бы отлаживали сложную проблему в коде?** Сначала я воспроизвожу проблему в тестовой среде, чтобы понять её контекст, и проверяю логи для выявления ошибок. Затем использую отладчик Visual Studio, чтобы шаг за шагом пройти по коду, отслеживая значения переменных и поведение. Если проблема остаётся неясной, я добавляю временное логирование или использую профилировщик для анализа производительности, чтобы найти источник.