И мы с вами подошли к написанию тестов. В этом уроке мы посмотрим немножко на теорию и виды тестов. Дальше переключимся к практике, посмотрим, как в go можно писать тесты.

Тестирование можно разбить на два больших направления. Самый очевидный и простой вид тестирования — это ручное тестирование. Здесь просто специалист садится за компьютер и руками начинает тестировать наш функционал. Это может быть вызов каких-то функций, вызов API методов, либо тыканье по интерфейсу в поисках каких-то проблем. В ручном тестировании есть очевидные минусы. Самый первый – это человеческий фактор, так как все это тестирование производится человеком и руками, то есть большой шанс ошибиться, потому что люди, все-таки, не машины. Еще один такой очевидный нюанс, что ручное тестирование проходит слишком долго. Опять-таки, из-за того, что все это проделывается руками, мы тратим на это очень много времени. Ручные тесты невозможно моделировать по нагрузке, то есть: один, два, три, четыре человека - все равно не смогут смоделировать очень большую нагрузку, которую бы мы могли, например, создать путем каких-то инструментов в программе. Ну и есть сложности в повторном использовании. Как правило, ручное тестирование проходит по каким-то кейсам, описанным заранее тестировщиками. То есть, есть какой-то скрипт поведения, по которому тестировщик садится и проходит шаг за шагом. И от специалиста к специалисту эти шаги могут повторяться с некоторыми расхождениями. Поэтому один человек может их выполнить одним образом, а другой человек выполнит их с какими-то отклонениями. Поэтому есть сложности в повторяемости тест-кейсов при ручном тестировании.

И второе большое направление это автотесты. Автоматическое тестирование решает большинство проблем ручного тестирования и существенно прибавляет в скорости и производительности. Плюс, при автоматическом тестировании мы можем моделировать нагрузку так, как мы хотим, и проверить более

специфические тест-кейсы. Но большим жирным минусом в автотестах является их стоимость. Организовать автоматическое тестирование намного дороже, чем организовывать ручное тестирование, потому что для автотестов уже нужно нанимать людей, разбирающихся в программировании, которые могут писать код и знают инструменты автоматического тестирования. Они знакомы с фреймворками и могут их использовать на практике.

Остановимся чуть детальнее на автоматическом тестировании. Для этого рассмотрим их в контексте микросервисной архитектуры. Есть такое понятие, как пирамида тестирования. И в основе этой пирамиды лежат модульные тесты. Модульные тесты — это вид автоматического тестирования, которые проверяют какие-то фрагменты нашего сервиса или кода. Эти фрагменты могут быть классами. Ну, в случае с go – это методы, структуры и различные функции. Модульные тесты считаются самыми быстрыми и легкими в реализации.

На следующем уровне у нас находятся интеграционные тесты. Данный вид тестирования проверяет, как наш код может интегрироваться с инфраструктурными компонентами, например, как базы данных и так далее.

Уровнем чуть выше находятся компонентные тесты. Эти тесты уже являются приемочными тестами для нашего приложения, то есть у нас есть какой-то сервис, и мы пишем компонентные тесты, которые проверяют функциональность этого сервиса и являются приемочными тестами к сервису.

И вершина нашей пирамиды — это сквозные тесты. Здесь уже нужно писать приемочные тесты для целого приложения. То есть приложение может состоять из нескольких сервисов, но мы пишем тесты, которые проверяют все сервисы в совокупности. И отправляют запросы в сквозную через несколько сервисов. Как вы могли догадаться, это самый сложный и долгий вид тестирования, потому что для

реализации сквозных тестов нужно поднимать всю инфраструктуру, которая необходима для приложения. То есть, если в приложении используется несколько сервисов, мы должны всех их поднять, чтобы в рамках тестирования проверить все наши запросы.