Добрый день и добро пожаловать в блог HTTP-сервис на Golang, в котором мы попробуем объединить все наши знания, полученные в предыдущих частях и разобрать по косточкам уже веб-сервис. Мы посмотрим, как веб-сервисы устроены. Мы разберемся, что такое Routing, а заодно со всеми терминами вообще в современных веб-сервисах на Golang. И посмотрим, как правильно организовать своё приложение, свой веб-сервис, погнали.

Итак, как работает HTTP сервис сервер на GO? Соответственно, у нас есть пользователь, который делает запрос к нашему серверу. Я сейчас рассматриваю только backend-овую часть, то есть я не рассматриваю пользователя, кликающего кнопки…, для пользователя, кликающего кнопки на frontend-е, backend-овая часть, соответственно, проходит через запросы с frontend-ом, в данном случае – браузер пользователя, то есть для меня это пользователь.

Пользователь делает запрос, запрос приходит в HTTP сервис, там он соответственно, из этих всех протоколов, сопутствующих разворачивается в непосредственно сам запрос и попадает в роутер. Роутер – это такой специальный инструмент, который по виду запроса, по самому запросу определяет – куда этот запрос пойдет. То есть сам роутер не содержит в себе никакой логики, логика содержится в отдельных компонентах, называемых handler-ами. В других, соответственно, framework-ах они могут называться «контроллеры». И в них уходят запросы с роутера, то есть у нас есть, например, всё про работу с пользователями, всё про работу с заказами пользователя, всё про деньги пользователя – это 3 разных контроллера в утрированной форме.

И соответственно, если пользователь хочет что-то в себе…, в своём профиле изменить или добавить – он идёт в роутер. Роутер видит, что это пользователь и соответственно, кидает его на контроллер пользователя, и так далее.

Соответственно, к каждому из контроллеров привязано один или несколько процессоров – это те ребята, которые содержат в себе уже непосредственно логику. То есть процессят, что-то делает уже с данными, которые пользователь принёс. Контроллеры – они валидируют данные, то есть они смотрят: «Ага, вот пришёл запрос, насколько он правильный? Пришёл запрос на создание пользователя с пустым телом – это правильный запрос или нет? Скорее всего, нет, потому что при создании пользователя нам нужен какой-то список полей. Пришёл запрос на чтение пользователя, например, с отрицательным id-шником – это правильный запрос или нет? Скорее всего, нет, потому что id-шники пользователя начинаются с 0 и вверх».

Соответственно, этим всем занимается контроллер, а процессор уже – он, например, при изменении пользователя, он сверит старые данные с новыми данными, поймет, что менять и сделает какие-нибудь калькуляции. Например, если вы…, если у вас пользователь покупает что-то и меняет данные карточки, то мы смотрим: может ли он поменять данные карточки, то если валидный ли запрос? А потом процессор уже обратится к сервису, который делает авторизацию карт, переавторизует новую карту. Всё соберет вместе, и, если всё успешно – засунет данные в storage. Storage – это такой специальный слой в сервере нашем, который работает с какой-то базой данных. Как правило, сервисы у нас stateless, то есть они внутри себя не запоминают ничего – они обрабатывают данные, кладут их в базу и база выполняет роль stateful, то есть базы выполняют роль хранителя тех самых данных. И после этого всё это идёт в обратном порядке, и контроллер оборачивает ответ и отдаёт, соответственно, его пользователям.

С точки зрения пользователя наш сервис выглядит следующим образом: после того, как пользователь сделал какой-то запрос, он обернул свои данные по всей этой модели OSI, которую мы рассматривали в прошлых лекциях, определил нужный…, определился внутри нашего сервиса нужный контроллер, обработал

запрос и вернул ответ. Соответственно, перевод запросов, понятно, в форму для нашего сервера выполняется web server-ом, то есть специальным таким компонентом, слоем, на котором всё у нас базируется. Потом контроллер определяется роутером согласно конфигурации, которую мы задаём. Дальше мы уже отвечаем за обработку данных, за опросы в базы данных и возвращаем ответ, который тоже обернётся web server-ом по всей модели OSI и отправится обратно пользователю.

Мы немножко посмотрим на формат HTTP запрос/ответ в непосредственно практической части, на что хотелось обратить бы внимание и это важно при разработке вашего web-server-а – это то, что каждый HTTP запрос должен получить ответ с каким-то кодом ошибки. И здесь важно понимать, что успешные ответы всегда имеют код 200, 201, но, если у вас что-то произошло внутри сервера, например, ошибка не может быть исправлена сервером. Мы не можем угадать, какие данные пользователи передавали или пользователь написал какую-то фигню, пользователь вызвал несуществующих метод контроллера.

То есть, например, наш сервис умеет создавать…, добавлять и…, вернее, создавать, редактировать и просматривать пользователя. Пользователь хочет что-то удалить. Передал свой (HP3 05:24), например, в строке – у нас такого метода нет, мы должны вернуть какую-то ошибку. И ошибка возвращается тоже с помощью кодов, то есть тоже такой же ответ на запрос, но по формату HTTP, по спецификации он позволит пользователю понять, что что-то случилось.

Например, если у нас нет такого маршрута, у нас ошибка должна выдаться 404. Есть какая-то некая спецификация HTTP кодов, которая гуглится по запросу HTTP кода ответа. Ещё известные коды: это 502, 503, соответственно, плохой запрос, сервер не доступен, то есть, если что-то произошло серьезное, и мы не можем на это никак отреагировать, мы должны вернуть какую-то ошибку.