Стабилизатор напряжения предназначен для преобразования автомобильного бортового напряжения 12В в требуемое напряжение 5В для питания схемы.

Модуль приемопередатчика, работающего на частоте 433 МГц, является входным и выходным трактом радиоканала блока управления [4].

Датчик давления преобразует величину давления в электрический сигнал. Приемная антенна осуществляет прием сигналов от датчиков давления и передачу их в блок управления. В качестве приемной антенны возможно использование антенны центрального замка автомобиля.

Источник питания представляет из себя гальванический элемент на основе диоксида литий-марганца (Li/MnO_2). Он поддерживает работоспособность датчика в течение 5-10 лет.

Для блока управления представлен алгоритм работы всей системы и разработана печатная плата в программе DipTrace.

Список использованных источников.

1.Система контроля давления в шинах [Электронный ресурс]. – Режим доступа: http://systemsauto.ru/electric/tires_pressure.html. – Дата доступа: 27.10.2017г.

2.Система контроля давления в шинах [Электронный ресурс]. – Режим доступа: http://www.automaster.net.ua/drukujpdf/artykul/50224. – Дата доступа: 10.11.2017г.

3.Система контроля давления в шинах [Электронный ресурс]. – Режим доступа: http://myauto26.ru/electric/tires_pressure.php. – Дата доступа: 16.11.2017г.

4.Радиочастотные микросхемы в решениях для систем контроля давления в шинах [Электронный ресурс]. – Режим доступа: http://www.rlocman.ru/review/article.html?di=130443. – Дата доступа: 16.11.2017г.

АВТОРИЗАЦИЯ НА ОСНОВЕ ПОЛИТИКИ ASP.NET CORE

Институт информационных технологий БГУИР,
г. Минск, Республика Беларусь

Черник Д.В.

Бакунова О.М. – ст. преподаватель каф. ИСиТ, м.т.н.
Бакунов А.М. – ст. преподаватель каф. ИСиТ, м.т.н.
Калетеня И.Л. ассистент каф. ИСиТ, м.т.н.

Механизм авторизации программного обеспечения гарантирует, что текущему пользователю разрешен доступ к данному ресурсу, выполнить задание или выполнить задание на данном ресурсе. В ASP.NET Core существует два способа настройки уровня авторизации. Вы можете использовать механизм предоставления ролей или же использовать политики авторизации. Авторизация на основе ролей основана на предыдущих версиях платформы ASP.NET, в то время как авторизация на основе политики является новой для ASP.NET Core.

Роли использовались в приложениях ASP.NET с релиза технологии. Технически говоря, роль - простая строка. Ее значение, однако, рассматривается как метаинформация для механизма безопасности приложения (проверяется на наличие в объекте IPrincipal) и используется приложениями для сопоставления набора прав и разрешений конкретному аутентифицированному пользователю. В ASP.NET вошедший в систему пользователь идентифицируется объектом IPrincipal, а в ASP.NET Core ему соответствует класс ClaimsPrincipal. Этот класс предоставляет набор идентификаторов, и каждый идентификатор представлен объектами IIdentity, в частности ClaimsIdentity. Это означает, что любой зарегистрированный пользователь имеет список претензий(claims), которые являются сведениями о его статусе. Имя пользователя и его роль – два общих типа претензий для пользователей приложения ASP.NET Core. Однако наличие или отсутствие роли зависит от хранилища идентификационных данных. Например, если вы используете социальную аутентификацию, вы никогда не увидите роли.

Авторизация следует за аутентификацией. Аутентификация — это поиск личности пользователя, тогда как авторизация — это определение прав, которыми пользователь обладает для доступа к конечным точкам приложения. Роли пользователя обычно хранятся в базе данных и извлекаются, когда учетные данные пользователя проверяются, и в какой момент работы приложения информация о роли связывается с учетной записью конкретного пользователя. Интерфейс IIdentity имеет метод IsInRole, который должен быть реализован для корректной проверки учетной записи пользователя. Класс ClaimsIdentity делает это, проверяя, что утверждение Role доступно в коллекции претензий, полученных в результате процесса аутентификации. В любом случае, когда пользователь пытается вызвать метод защищенного контроллера, его роль должна быть доступна для проверки. В противном случае пользователю запрещается вызов любых защищенных методов.

Атрибут Authorize — это декларативный способ защиты контроллера или некоторых его методов.

Указанный без аргументов атрибут проверяет только аутентифицирован ли пользователь. Однако атрибут также поддерживает некоторые дополнительные параметры, такие как Roles. Свойство Roles указывает, что пользователям в любой из перечисленных ролей будет предоставлен доступ. Чтобы потребовать несколько ролей, вы можете применить атрибут Authorize несколько раз или написать собственный фильтр.

При желании атрибут Authorize также может принимать одну или несколько схем аутентификации через свойство ActiveAuthenticationSchemes.

Свойство ActiveAuthenticationSchemes — это строка с разделителями-запятыми, в которой перечислены компоненты промежуточного ПО аутентификации, которым механизм авторизации будет доверять в текущем контексте. Другими словами, он утверждает, что доступ к классу контроллера разрешен только в том случае, если пользователь аутентифицирован по выбранной схеме и имеет любую из перечисленных ролей. Строковые значения, переданные в свойство ActiveAuthenticationSchemes, должны совпадать с промежуточным ПО проверки подлинности, зарегистрированным при запуске приложения.

В ASP.NET 2.0 промежуточное ПО проверки подлинности заменяется службой с несколькими обработчиками. В результате схема аутентификации является меткой, которая выбирает необходимый обработчик.

Информация, предоставляемая атрибутом Authorize, используется встроенным фильтром авторизации. Поскольку он отвечает за проверку того, может ли пользователь выполнить запрошенную операцию, этот фильтр запускается перед любым другим фильтром ASP.NET Core. Если пользователь не авторизован, фильтр останавливает конвейер обработки и отменяет запрос.

Общая схема движения HTTP запроса в ASP.NET приложении имеет следующий вид (рисунок 1):

Роли — это простой способ группировать пользователей приложения на основе того, что они могут или не могут сделать. Но они не очень выразительны; по крайней мере, недостаточно для удовлетворения потребностей большинства современных приложений.

Роли по существу являются плоскими понятиями. Как только количество вариантов использования приложения возрастает, количество требуемых ролей значительно возрастает. Из-за минусов системы авторизации на базе ролей и в результате эволюции платформы, была создана авторизацию на основе политики.

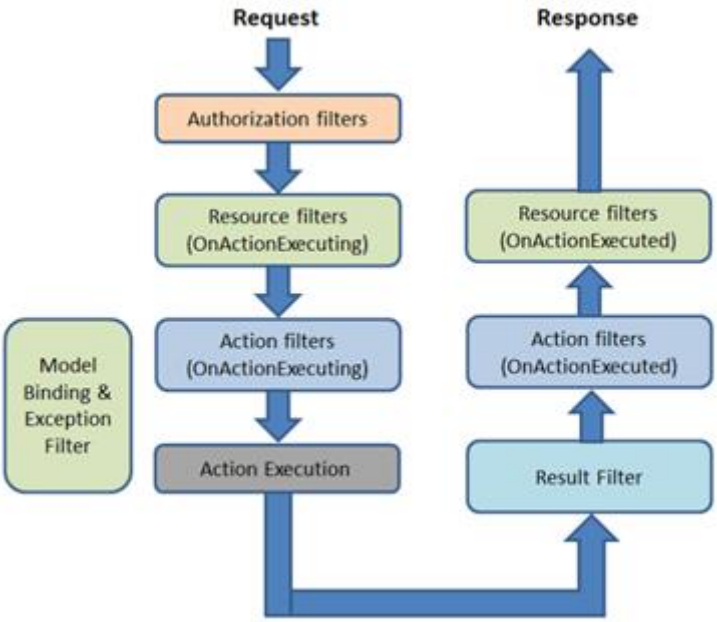

Рисунок 1 - Схема движения HTTP запроса

В ASP.NET Core, основанный на политиках механизм авторизации предназначен для разделения слоев авторизации и логики приложения. Проще говоря, политика — это сущность, разработанная как совокупность требований, которые сами по себе являются условиями, которым должен удовлетворять текущий пользователь.

Простейшая политика представляет собой набор требований, наиболее общими из которых являются: пользователь аутентифицирован, пользователь связан с данной ролью. Другое общее требование заключается в том, чтобы пользователь имеет конкретное претензий(claim) или конкретное требование с определенным значением. В наиболее общих терминах претензии— это утверждение о пользователе, который пытается получить доступ к методу, что претензия установлена и имеет значение true.

Для создания политики необходимо использовать объект AuthorizationPolicyBuilder, который предназначен для конфигурирования и сбора требований, используя различные методы расширения, а затем создает экземпляр политики. Требования действуют на статус аутентификации и схемы, роль и любую комбинацию претензий, прочитанных через файл Cookie или токен-носитель.

Если ни один из предопределенных методов расширения для определения требований вас не устраивает, вы всегда можете прибегнуть к определению собственного метода расширения для объекта AuthorizationPolicyBuilder.

Однако недостаточно просто создать объект политики - вы также должны зарегистрировать вашу политику для механизма авторизации использую промежуточное ПО. Вы делаете это, добавляя промежуточное ПО как службу в методе ConfigureServices для класса Startup. Каждая политика, добавленная к промежуточному ПО, имеет имя, которое используется для ссылки на политику в атрибуте Authorize в классе контроллера.

Атрибут Authorize позволяет вам устанавливать политику декларативно, но политики также могут быть запрограммированы программно прямо в методе действия. Если программная проверка разрешений не удалась, вы можете вернуть объект ForbiddenResult. Другой вариант - возвращение объекта ChallengeResult.

Также вы имеете возможность выполнить программную проверку политик в режиме Razor. Для этого вы должны сначала ввести зависимость от службы авторизации. Использование службы авторизации в представлении может помочь скрыть элементы пользовательского интерфейса, которые не должны находиться в пределах досягаемости текущего пользователя, учитывая текущий контекст. Однако имейте в виду, что просто скрывать варианты в представлении недостаточно. Вы всегда должны применять политики в контроллере.

Базовые требования политики в основном охватывают претензии(claims), аутентификацию и обеспечивают универсальный механизм настройки на основе утверждений, но вы также можете создавать пользовательские требования. Требование к политике состоит из двух элементов: класса требований, содержащего только данные, и обработчика полномочий, который проверяет данные пользователя. Пользовательские требования расширяют ваши возможности настройки определенной политики.

Требование должно иметь по крайней мере один обработчик авторизации. Обработчик имеет тип AuthorizationHandler<T>, где T - тип требования.

Обработчик требований авторизации считывает претензии, связанные с конкретным пользователем, и проверяет на соответствие с собственными претензиями. Если всех соответствий не найдено, обработчик возвращает отрицательный результат. Положительный результат возвращается только в том случае, если претензии существует и содержит указанное значение. Предполагается, что пользовательское претензии будут частью информации, связанной каким-то образом с пользователем, например, столбцом в одной из таблиц базы данных.

В свою очередь обработчик авторизации вызывает метод Succeed, передавая текущее требование, чтобы уведомить, что требование было успешно проверено. Если это требование не прошло валидацию, обработчик не должен ничего делать и может просто вернуться к дальнейшей работе. Однако, если обработчик хочет определить отказ требования независимо от того, что другие обработчики поэтому же требованию могут завершиться успешно, он вызывает метод Fail на объекте контекста авторизации.

Также, вам необходимо зарегистрировать новый обработчик с системой DI в рамках типа обработчика IAuthorization.

Каждое требование может иметь несколько обработчиков. Когда несколько обработчиков зарегистрированы в системе DI для одного и того же требования достаточно, чтобы по крайней мере один был завершен успешно.

Авторизация на основе политики — это новый подход, который обеспечивает более богатую и выразительную модель. Это связано с тем, что политика представляет собой набор требований на основе требований и пользовательской логики на основе любой другой информации, которая может быть введена из контекста HTTP или внешних источников. Эти требования связаны с одним или несколькими обработчиками, которые отвечают за фактическую оценку требования.

Список используемых источников

1.Документация ASP.NET Core. [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/en-us/aspnet/core. – Дата доступа: 20.03.2018.

2.Документация .NET [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/library – Дата доступа: 23.03.2018.

3.Документация и исходный код. [Электронный ресурс]. – Код доступа: https://github.com/aspnet . – Дата доступа: 15.03.2018.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ ДЛЯ УПРАВЛЕНИЯ АДМИНИСТРАТИВНОЙ ПАНЕЛЬЮ MAGENTO

Институт информационных технологий БГУИР,
г. Минск, Республика Беларусь

Черный М.С.

Пачинин В. И. – зав. кафедрой ИСиТ, к.т.н., доцент
Сечко Г. В. – доцент каф. ИСиТ, к.т.н,, доцент

Объектом исследования является программное обеспечение выполняющее функцию управления административной панелью Magento с мобильного устройства. В административной панели Magento располагается информация о пользователях ресурса, продуктах, заказах и транзакциях. Также ее главной сильной стороной является обилие встроенных функций: речь идет о валюте, языках, скидках и купонах и многом другом.

Для любой системы онлайновой торговли очень важно обеспечить непрерывность функционирования (отсутствие простоев и замедлений в работе). По сути, это является ключевым требованием, которое крайне важно, как с точки зрения маркетинга и PR, так и с точки зрения бизнеса в целом.

Целью проекта является разработка программного обеспечения для управления административной панелью Magento, с целью своевременного обнаружения неполадок в работе сервера и получения актуальной информации о текущем состоянии аппаратного обеспечения, товарах, пользователях и отчетах за выбранный период.

Для выполнения поставленной задачи было разработано клиент-серверное программное средство, состоящее из iOS приложения, а также серверной части, представленной в виде Magento и Gateway серверов. Архитектура программного средства представлена на рисунке 1.