Мною была реализована программа, предназначенная для авторизации пользователей в личном кабинете интернет-магазина. Ее функционал включает в себя регистрацию по номера телефона, вход в личный кабинет по многоразовому паролю, восстановление пароля по номеру телефона и одноразовому паролю, а также двухфакторная аутентификация для совершения важных с точки зрения информационной безопасности действий.

**Техническое описание программы:**

Программа представляет собой web-сервис на платформе Node.js, реализованный с помощью фреймворка Express.js. Авторизация пользователей происходит с помощью модуля Passport.

Также были использованы три базы данных: реляционная БД MySQL для хранения учетных данных пользователей, нереляционная высокопроиводительная БД Redis для хранения данных о текущих сессиях и нереляционная документоориентированная БД MongoDB для хранения временных одноразовых паролей.

Рассмотрим подробнее назначение каждой из них.

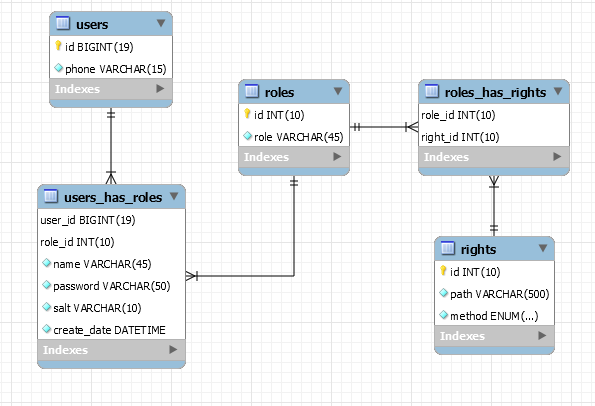
**MySQL:**

MySQL – реляционная система управления базами данных. Данные хранятся в четко структурированных таблицах, связанных между собой. Это очень удобно, когда заранее известна структура данных и предполагается наличие множества однотипных данных, которые можно описать единым образом.

В частности, в данной программе она была использована для хранения учетных записей пользователей, их ролей и прав доступа. Система предполагает две роли, продавца и покупателя. Каждой роли можно прописать соответствующие права. То есть если покупатель может только просматривать товары, то продавец еще и редактировать, то есть у него есть права на запись в таблицу в товарами.

Также есть таблица пользователей, в которой хранятся телефоны, что идентифицируют пользователя. Пользователи связаны с ролями в отдельной таблице. В ней же есть входные данные для пользователя, хеш его пароля и соль. Эти данные хранятся отдельно от списка телефонов для того, чтобы один человек под одним номером телефона мог выступать как в роли продавца, так и в роли покупателя.

Описанная структура данных представлена на рисунке.



**Обоснование выбора:**

MySQL считается самой популярной из всех крупных серверных СУБД. Существует большое количество версий для множества операционных систем, что означает, что приложение не зависит от платформы и может быть расположено практически на любом сервере. Также в связи с популярностью этой СУБД существует множество документации по работе с ней и вариантов решения различных проблем. Система также довольно проста для понимания в отличие от некоторых похожих СУБД. MySQL поддерживает множество функций и имеет свой встроенный функционал по безопасности.

Помимо хранения самих данных, в MySQL будут хранится функции и хранимые процедуры для взаимодействия с этими данными. Существуют средства, которые связывают базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». Это так называемые ORM (англ. Object-Relational Mapping, рус. объектно-реляционное отображение, или преобразование).

Однако есть ряд причин, по которым в данной программе были использованы именно хранимые процедуры для взаимодействия с данными БД. Приведем некоторые из них:

* Как правило, производительность приложения существенно возрастает. В рамках одного подключения при вызове процедуры она компилируется (выполняется синтаксический анализ и генерируется план доступа к данным) и сохраняется в кэш. MySQL заводит отдельный кэш на каждое подключение. Если приложение использует хранимую процедуру несколько раз, она вызывается из кэша, иначе хранимая процедура работает как обычный запрос к базе.
* Использование хранимой процедуры уменьшает трафик между приложением и базой данных. Всего одна команда на выполнение хранимой процедуры позволяет вызвать содержащийся в ней сложный сценарий, благодаря чему можно избежать пересылки через сеть сотен команд, и в особенности, необходимости передачи больших объемов данных для анализа, поскольку это можно реализовать в рамках сервера базы данных.
* Хранимая процедура может вызываться несколькими приложениями, использующими одну базу данных, позволяя не писать код повторно.
* Можно накладывать ограничение на вызов хранимой процедуры в зависимости от пользователя базы данных.

Конечно, есть и недостатки используемого подхода. Во-первых, требуется большее количество памяти, а если при этом используется много логических операторов, то еще и увеличивается нагрузка на ЦП. Также стоит заметить, что разработка хранимых процедур достаточно сложна, особенно учитывая тот факт, что в MySQL нет отладки кода.

Но, поскольку для данного приложения не требуется много различных операций с данными, и все они достаточно просты, то память задействуется не существенно больше, а логика написания весьма прозрачна. При этом действительно важна производительность и размер трафика между приложением БД в случае большого количества пользователей.

Код создания таблиц, связей между ними и хранимых процедур доступен в приложении.

**Redis:**

Redis — резидентная система управления базами данных класса NoSQL с открытым исходным кодом, работающая со структурами данных типа «ключ — значение». Используется как для баз данных, так и для реализации кэшей, брокеров сообщений. Ориентирована на достижение максимальной производительности на атомарных операциях.

Redis действительно обладает высокой производительностью, поскольку хранит данные в оперативной памяти. Это делает ее идеальным инструментов для хранения текущих сессий пользователей. Можно было бы хранить эти данные в памяти процесса, однако это делает приложения менее масштабируемым.

Для взаимодействия с Redis необходимо подключение следующих модулей.

**const** session = require('express-session');

**const** redisStore = require('connect-redis')(session);

app.use(session({  
 secret: config.sessionSecret,  
 store: **new** redisStore({client: dbs.redis}),  
 saveUninitialized: **false**, resave: **false** }));

**MongoDB:**

Для хранения одноразовых временных паролей используется СУБД MongoDB. MongoDB — нереляционная СУБД для хранения JSON объектов. В данной программе используется достаточно однотипные данные, которые логично было бы хранить в реляционной БД, так как у них вполне четкая структура. Однако есть ряд преимуществ, которые склоняют нас к выбору MongoDB. Во-первых, используемая функциональность отправки смс-кодов может значительно поменяться при дальнейшей разработке приложения. На данном этапе неясно, как именно она может меняться. Нереляционная БД гораздо более масштабируемая и гибкая по сравнению с реляционной. Даже если данная структура изменится сильно, это не повлечет за собой глобальных изменений в коде. Во-вторых, данная СУБД «легче» и быстрее. Смс-коды имеют срок действия. Учитывая время, необходимое сервису по отправке смс для обработки запроса, а также время доставки смс, необходимо минимизировать время, используемое для взаимодействия с БД, хранящей данные коды.

**Структура программного средства**

1. Пользователь получает страницу с возможностью зарегистрироваться или зайти в личный кабинет.
2. При выборе варианта «зарегистрироваться» пользователь
   1. Вводит свой номер телефона
   2. Подтверждает, что телефон принадлежит ему, путем ввода смс-кода, отправленного ему на телефон.
   3. Вводит имя и пароль для окончания регистрации.
   4. Затем перенаправляется на страницу входа в личный кабинет.
3. При выборе варианта «зайти» пользователь
   1. Вводит номер телефона в качестве логина и пароль.
   2. Если он не помнит пароль, то может пройти процедуру сброса пароля, получив код по смс для подтверждения, что телефон принадлежит ему, и создав новый пароль.
      1. После чего отправляется на страницу входа в личный кабинет.

3.3 После входа пользователь оказывается в личном кабинете, функционал которого представляет собой возможность выйти из личного кабинета.

Рассмотрим более детально основные функциональные блоки

**Регистрация**

Регистрация происходит по номеру телефона пользователя. С целью исключить возможность регистрации множества фальшивых пользователей, а также использование чужих номеров телефона, пользователю необходимо подтвердить свой номер. Для этого он должен ввести код, полученный в смс-сообщении.

Рассмотрим более детально, как это происходит.

При запросе пользователя зарегистрироваться, сервер получает номер телефона, наличие которого проверяется в базе данных вызовом специальной хранимой процедуры (приложение комочков). При отсутствии номера в базе, генерируется случайное четрехзначное число. Номер телефона, код и время его создания, а также время после которого, код перестанет быть валидным, сохраняется в базу данных. Схема описана в приложении комочков.

Затем код посылается абоненту с помощью сервиса sms.ru. Конфигурационные настройки отправки в приложении комочков.

После чего пользователю отправляется страница с просьбой ввести код для подтверждения номера телефона. Также на странице есть счетчик времени, когда код считается действительным и возможность прислать новый по истечении срока годности.

После того, как пользователь отправляет код, сервер делает запрос к БД и сравнивает пару «телефон-код». В случае ошибки, запрашивает ввести код заново. В случае совпадения дает пользователю страницу с возможностью завершить регистрацию, введя имя и пароль. Пара «телефон-код» удаляется из БД.

router.post('/signup', (req, res, next) =>  
 user.checkCustomerAbsence(utils(req.body.phone))  
 .then(() => smsCode.remove(utils(req.body.phone)))  
 .then(() => smsCode.add(utils(req.body.phone)))  
 .then(code => sms.send(utils(req.body.phone), code)).then(() => res.render('check-code', {  
 phone: req.body.phone  
 }))  
 .catch(err => {  
 res.render('signup', {errorMessage: err.sqlMessage});  
 }));

На пароль наложены ограничения. Он должен содержать не менее 8 символов, строчные и заглавные латинские буквы и цифры. Данные ограничения необходимы для предотвращения слишком простых паролей, которые легко подбираются. Пока пользователь не выполнит всех условий, система не даст ему завершить регистрацию. После того, как пользователь ввел имя и валидный пароль, вызывается хранимая процедура по регистрации. В ней генерируется соль (приложение комочков), хешируется строка, которая представляет собой конкатенацию пароля и соли, хеш-функцией MD5(приложение комочков), затем в базу сохраняется номер телефона пользователя и его роль вместе с именем, хешем пароля и солью.

router.post('/signup/checkcode', (req, res, next) => {  
 smsCode.get(utils(req.body.phone), req.body.code)  
 .then(result => {  
 **if** (!result) **return** Promise.reject(**new** customError('You put a wrong code. Please, try again.', 400));  
 **return** res.render('registration', {  
 phone: req.body.phone,  
 code: req.body.code  
 });  
 })  
 .then(() => smsCode.remove(utils(req.body.phone)))  
 .catch(err => res.render('check-code', {errorMessage: err.message, phone: req.body.phone})  
 )  
});

После успешной регистрации пользователю предлагается войти в систему.

Аналогично происходит восстановление процедура пароля. В случае, если пользователь забыл пароль, он вводит свой номер телефона, на который приходит код подтверждения. Перед отправкой кода проверяется наличие такого телефона в базе. В данном случае, наоборот, нужно, чтобы такой пользователь существовал, иначе программа предупредит пользователя, что такой номер не зарегистрирован и предложит пройти процедуру регистрации.

После подтверждения номера телефона с помощью кода из смс, пользователю предлагается ввести новый пароль, который также должен соответствовать требованиям безопасности.

**Вход в систему**

Для входа в систему пользователь должен ввести свой номер телефона и пароль. Аутентификация происходит с помощью модуля Passport, настройки которого прописаны в приложении комочков. Вызывается хранимая процедура login (приложение комочков). В ней производится поиск записи с указанным номеров телефона. Если таких нет, то пользователю возвращается сообщение об ошибке с просьбой проверить номер телефона или зарегистрироваться. Если такая запись есть, то берется сохраненная в ней соль, складывается с паролем и хешируется хеш-функцией MD5. Затем полученных хеш сравнивается с записанных хешем в БД. В случае несовпадения, пользователь получает сообщение об ошибке в пароле. Если хеши совпадают, то процедура возвращает true, модуль Passport создает сессию и сохраняет ее в БД Redis. Далее пользователь перенаправляется на домашнюю страницу, которая защищена проверкой на авторизованность пользователя. При попытке ввести адрес данной страницы в ручную, система проверит, авторизован ли пользователь, и в случае отрицательного ответа, автоматически перенаправит его на страницу с регистрацией или входом в систему.

Точно так же, при попытке авторизованного пользователя зайти на страницу для авторизации, он будет перенаправлен на свою домашнюю страницу, чтобы избежать повторной авторизации.

**Выход из системы**

При нажатии на кнопку «Log out»,текущая сессия пользователя уничтожается и он перенаправляется на начальную страницу.

Используемые сессии не имеют ограничений по времени. Однако их можно установить, если считается, что приложение имеет важные с точки зрения защиты информации функции и несанкционированный доступ принесет значительный ущерб для кого-либо из сторон. Например, если в него будет встроена возможность оплаты покупок онлайн.

В таком случае, желательным будет использовать двухфакторную аутентификацию. Когда пользователь захочет совершить действие, которое в случае несанкционированного доступа сможет нанести некоторый ущерб, он будет обязан подтвердить свою личность с помощью одноразового пароля, полученного им в смс, несмотря на то, что он уже авторизован в системе.

**Оценка безопасности приложения**

Во второй главе были рассмотрены основные уязвимости парольной аутентификации, а также были даны рекомендации по разработке безопасных веб-приложений. На основе этого был проведен анализ безопасности реализованного приложения. Были отмечены сильные стороны приложения и выявлены его слабые стороны и определен план по усовершенствованию программы стойкой к основным уязвимостям.

Рассмотрим сначала сильные стороны приложения.

Сильные стороны приложения:

1. Сложные пароли.

Программа реализована таким образом, что пользователь не может создать быстро-взламываемый пароль. На него наложены следующие ограничения: пароль должен содержать минимум 8 символов, латинские буквы разных регистров и цифры.

Если накладывать на пароль более жесткие ограничение (например, увеличить минимальную длину до 12 символов или потребовать букв из разных языков), то это вызывает трудности у потенциального пользователя программы, то есть покупателя, которому в данном случае важно удобство, иначе он просто не станет пользоваться данной программой.

Использовать пароли, сгенерированные специальной программой тоже нецелесообразно.

1. Хеш-функция для хеширования паролей

Пароли пользователей хешируются встроенной в хранимую процедуру MySQL хеш-функцией MD5.В настоящий момент времени она считается достаточно надежной для хранения паролей. Единственным недостатком ее считается быстрое время работы. В случае атаки перебором, время на хеширование практически не тратится, поэтому зачастую рекомендуют использовать хеш-функции, время работы которых искусственно замедленно. Однако ограничение по длине возможного пароля решает эту проблему.

1. Безопасное восстановление пароля

Восстановление пароля требует аутентификации по одноразовому паролю, который пользователь получает на свой номер телефона, что означает, что злоумышленник не может изменить чужой пароль, пока не завладеет чужим телефоном или на сможет перехватывать приходящие смс сообщения, что сделать достаточно трудно, особенно учитывая, что для подобной сложности взламывания системы у злоумышленника недостаточно мотивации.

1. Двухфакторная аутентификация для важных действий

В рекомендациях разработчику в главе 2 было указано, что для совершения важных действий необходимо требовать от пользователя повторной аутентификации. В данной программе в таких ситуациях пользователю не приходится повторно вводить свой пароль для совершения важных действий. Ему нужно подтвердить свой телефон введением одноразового пароля, полученного им по смс. Такой способ считается даже надежней, нежели повторная аутентификация.

Отдельно от защищенности самой аутентификации, хочется отметить, что в приложении все страницы защищены от несанкционированного входа путем ввода их адреса в адресную строку. Перед тем, как вернуть страницу пользователю, сервер обязательно проверяет авторизованность данного пользователя в системе.

Также сервер проверяет и обрабатывает все входящие данные от клиента по некоторым параметрам, соответствие которым мы ожидаем от полученных данных. Это снижает риск SQL-инъекций, потому что на введенные данные накладываются ограничения.

Слабые стороны:

1. Данное приложение никак не ограничивает число попыток входа, что подвергает его риску атаки полного перебора паролей.
2. Также нет функционала, который заставлял бы пользователей менять пароль через определенное время. Однако, как правило, пользователи редко запоминают пароли от приложений, которыми они пользуются не каждый день. Это значит, что периодическая смена паролей в какой-то будет достигаться за счет восстановления забытого пароля.
3. Приложение автоматически не закрывает сессию после продолжительной неактивности клиента, что дает возможность несанкционированного доступа злоумышленнику, который каким-то образом завладел компьютером пользователя. Однако для всех важных действий, пользователю необходимо проходить повторную аутентификацию, что означает, что ничего существенного злоумышленник совершить не сможет.
4. Приложение осуществляет связь с сервером по незащищенному протоколу HTTP. Переход на работу по защищенному каналу планируется в дальнейшей работ над приложением.

Вывод:

Приложение достаточно хорошо защищено для реализуемого функционала. Обнаруженные недостатки не считаются критичными в условиях требований обеспечения безопасности интернет-магазина. В дальнейшем планируется ввести дополнительную защиту от перебора паролей, а также перейти на клиент-серверную связь по защищенному HTTPS протоколу.

А также, при расширении используемого функционала планируется увеличение используемых средств защиты информации .