5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

5.1 Модульное тестирование

Модульное тестирование необходимо для обнаружения ошибок в реализации алгоритмов и вообще работы программного средства, в рамках требуемой бизнес-логики. Оно позволяет проверять корректность выполнения кода как непосредственно после его написания, так и в течение жизненного цикла программного средства.

Написанные модульные тесты позволяют отслеживать ошибочные или преднамеренные изменения в программном коде в процессе изменения функциональности всего модуля. Это один из самых действенных инструментов поддержания правильности выполнения программы и контроля ее изменения.

В рамках данного дипломного проекта модульные тесты писались для наиболее важных с точки зрения безопасности и уязвимых модулей. Тесты были созданы для модуля управления пользователями, ролями и разрешениями на доступ. Дополнительно тесты были написаны на модуль, который осуществляет импорт данных в систему. Это необходимо, так как нужно осуществлять корректность обработки файлов импорта в системе перед созданием их в базе данных.

Тест `UsersShouldNotBeAbleToRegisterIfNotAllowed` осуществляет проверку на то, что пользователь в системе не может зарегистрироваться, если данная опция не установлена на сайте. В тесте устанавливается настройка, запрещающая регистрацию, и производится попытка регистрации нового пользователя. В результате должно возникнуть исключение `HttpNotFoundResult`.

Тест `UsersShouldBeAbleToRegisterIfAllowed` осуществляет проверку на возможность регистрации пользователя в системе. В тесте устанавливается настройка для сайта, разрешающая регистрацию, и производится регистрация нового пользователя. В результате в системе должен создаться новый пользователь.

Тест `UsersShouldNotBeAbleToRegisterIfInvalidEmail` осуществляет проверку электронной почты пользователя при регистрации, которая должна быть правильно написана. В тесте создаются различные значения электронной почты, которые являются неправильно написанными, и производится попытка регистрации. После этого производится проверка на то, что в состоянии модели, которая приходит на сервер, содержится одна ошибка. Это значит, что ошибочная почта обнаружена.

Тест `UsersShouldBeAbleToRegisterIfValidEmail` осуществляет проверку на возможность регистрации пользователя с правильно почтой. В тесте создается набор из различных правильных почт и производится регистрация пользователя с ними. В результате пользователи

должны быть созданы и перенаправлены на домашнюю страницу сайта.

Тест `RegisteredUserShouldBeRedirectedToHomePage` осуществляет проверку, что пользователь, успешно прошедший регистрацию, будет перенаправлен на домашнюю страницу сайта. В тесте происходит регистрация пользователя, а затем производится проверка, что он был перенаправлен системой на домашнюю страницу.

Тест RegisteredUserShouldBeModerated осуществляет проверку на то, что вновь зарегистрировавшийся пользователь должен иметь статус, предусматривающий его активацию администратором. В тесте происходит регистрация нового пользователя и проверяется его статус, который должен соответствовать статусу ожидающего активацию.

Тест `SuperAdminReceiveAMessageOnUserRegistration` осуществляет проверку на то, что после регистрации пользователю с полными правами администратора должно прийти письмо об этом событии. Это необходимо для того, чтобы администратор мог активировать нового пользователя. В тесте происходит регистрация нового пользователя. Затем производится проверка на то, что администратор получил новое письмо об этом событии.

Тест `InvalidLostPasswordRequestShouldNotAnError` организует проверку на то, чтобы при восстановлении пароля пользователем, если он ввел неправильный логин, то он не должен авторизоваться в систему под ним. В тесте регистрируется новый пользователь. Затем он осуществляет запрос на восстановление пароля не для своего аккаунта. После этого происходит проверка на то, что он не может авторизоваться в систему под чужим аккаунтом.

Тест `ResetPasswordLinkShouldBeSent` осуществляет проверку на то, что при запросе о восстановлении пароля, система отправляет на почту ссылку на его сброс. В тесте выполняется регистрация нового пользователя, который затем осуществляет запрос на восстановление пароля. После этого проверяется, что количество сообщений для этого пользователя равняется единицы, а значит, письмо было отправлено.

Тест `CreateRoleShouldAddToList` осуществляет проверку на то, что вновь создаваемая роль для пользователей в системе должна отобразиться в списке. В тесте осуществляется создание трех новых ролей. Затем вызывается метод сервиса `RoleService`, который возвращает все роли в системе. После этого осуществляется проверка на то, что ролей в системе три, а также проверяется правильность их названий.

Тест `PermissionChangesShouldBeVisibleImmediately` выполняет проверку на то, что при добавлении к роли определенных разрешений, они должны быть применены немедленно и отображены на станице. В тесте осуществляется создание новой роли. Затем к ней применяется набор новых разрешений на доступ к определенным модулям. После этого с помощью сервиса `RoleService` осуществляется получение

объекта роли и выполняется проверка на количество примененных к ней разрешений. Их должно оказаться, сколько же, сколько было добавлено.

Тест `ShouldNotCreateARoleTwice` осуществляет проверку на то, что имя роли должно быть уникальным. Таким образом, при попытке создания роли с уже имеющимся именем, в систему новая роль не должна добавиться. В тесте происходит создание двух новых ролей. Затем производится создание еще одной роли с уже имеющимся в системе именем. После этого из сервиса `RoleService` осуществляется получение всех ролей и производится проверка на то, что их количество равняется двум, и они имеют имена, которые были заданы выше.

Тест `CreateUserShouldAllocateModelAndCreateRecords` осуществляет проверку, что вновь созданный пользователь записывается в базу данных. Для этого в тесте создается новый пользователь с заданным именем и почтовым адресов. После этого вызывается сервис `MembershipService`, который возвращает только что созданного пользователя. Затем происходи проверка на то, что имя пользователя и почтовый адрес корректно записались и прочитались из базы данных.

Тест `DefaultPasswordFormatShouldBeHashedAndHaveSalt` осуществляет проверку на то, что пароль для пользователя по умолчанию должен храниться в базе захэшированным и иметь ключ для обратного разхэширования. Для этого осуществляется создание нового пользователя и получение его из базы данных с помощью сервиса `MembershipService`. После этого в свойствах полученного объекта пользователя проверяется, что его пароль был захэширован и что имеется ключ для процесса обратного разхэшировния.

Тест `SaltAndPasswordShouldBeDifferentSamePassword` осуществляет проверку на то, что для одинаковых паролей в системе будут созданных различные ключи в базе данных для их расхэширования. В тесте происходит создание двух пользователей, имеющих различные имена, но одинаковые пароли. После этого осуществляется их получение из базы данных. Далее производится проверка на то, что их ключи отличаются друг от друга, а также что захэшированные пароли также отличаются в базе данных.

Тест `UserReturnNullIfUserOrPasswordIsIncorrect` осуществляет проверку на то, что сервис `MembershipService` возвращает пустое значение, если запрашивается пользователь, который не существует или введенный для него пароль является неверным. В тесте создается новый пользователь с именем и паролем. Затем осуществляется попытка получить его по неправильному имени или с неправильным паролем. Данные операции должны возвратить пустое значение объекта. После этого осуществляется попытка получения пользователя с правильным именем и паролем и проверяется, что возвращен действительно тот пользователь.

Тест CanSendEmailUsingAddresses осуществляет проверку на то, что система может отправить письмо на указанный электронный адрес. В тесте осуществляется вывозов метода отправки письма с определенными параметрами. Затем проверяется, что количество отправленных писем изменилось на единицу.

Тест OneMessageIsSentUsingMultipleRecipients осуществляет проверку, что система может отправить одно и то же письмо на несколько электронных адресов. Для этого вызывается метод, отправляющий письма с параметрами, которые содержат несколько адресов. После этого осуществляется проверка, что количество отправленных писем увеличилось на столько, сколько адресов указано в параметрах метода.

Тест parse__null_parameters__exception_thrown проверяет, что при попытке разбора файла, которого не существует, возникает исключение ArgumentNullException. В тесте создается новый объект для разбора файла импорта и передается для разбора нулевое значение.

Тест parse__valid_file_and_scheme осуществляет проверку на работоспособность метода по разбору файла для импорта. Внутри теста происходит создание объекта для разбора текста, создается строковое представление трех параметров для разбора и вызывается непосредственно метод. Затем проверяется, что метод корректно отработал и вернул все те же три разобранных строки.

Тест parse__empty_file__exception_thrown осуществляет проверку, что при попытке разбора пустого файла для импорта возникает исключение FeedParsingException. В тесте создается новый объект разбора файла, затем для выполнения методу этого объекта передается пустой файл и проверяется его поведение.

Тест parse__header_and_scheme_inconsistent_exception осуществляет проверку на то, что если при разборе файла импорта обнаруживается поле, которое отсутствует в базе данных, то возникает исключение FeedParsingException. В тесте создается объект для разбора файла, а также тестовый файл с лишним полем. Затем осуществляется попытка разбора и проверяется, что метод создает требуемое исключение.

Тест parse__missing_values_for_item__error_result осуществляет проверку на корректность выполнения разбора файла в случае, если в его теле в конце файла отсутствуют какие либо поля. Это считается корректным случаем, так как допускается в конце не указывать поля, не имеющие значения. В тесте создается новый объект для разбора файла, затем создается тестовый файл с тремя записями, у которых отсутствуют поля в конце. Созданные тестовые данные передаются на выполнение методу. Затем проверяется, что метод корректно отработал и все три записи успешно разобраны. Кроме того проверяются полученные разобранные значения.

5.2 Проверка корректности входных данных

Помимо написания модульных тестов для проверки корректности написания кода непосредственно разработчиком, необходимо проверять на правильность входные данные, которые поступают в систему.

Входные данные в основном поступают от пользователей, которые вводят их в специальные формы на страницы. При этом существуют требования к типу вводимых данных, их размеру и количеству.

Для выполнения таких требований применяется серверная проверка данных. Для этого, пользуясь средствами применяемого языка программирования, при возвращении вводимых данных на сервер они проверяются на правильность.

К полям, в которые записываются данные, применяются специальные атрибуты, которые и описывают необходимые требования.

Атрибут `StringLength` определяет максимальную допустимую длину строки, которую можно записать в данную переменную. Это необходимо для ограничения на ввод данных пользователем, например, при описании дополнительной информации при запросе на машину.

Атрибут `DisplayName` позволяет выбрать название для поля, которые будет отображаться для пользователя на странице. Это удобно, когда имя поля состоит из нескольких слов.

Атрибут `FieldGroup` позволяет отнести используемое поле к одной из созданных ранее групп. Это удобно делать при проверке вводимых на форме полей, когда должно быть заполнено одно из полей группы.

Атрибут `Required` позволяет определить поле как обязательное для заполнения на форме. При этом если пользователь не заполнит данное поле, то он не сможет сохранить форму. Этот атрибут необходим, когда необходимо организовать ввод полей, которые необходимы для некоторой конкретной сущности и без которых она не имеет смысла.

Несколько атрибутов сравнения `GreaterThan`, `LessThan`, `EqualTo` применяются для сравнения значений на водимой форме между собой. Преимуществом их применения является то, что написав логику работы проверки один раз, можно использовать их в дальнейшем для различных полей, просто пометив их соответствующим атрибутом. При этом не нужно писать дополнительную логику проверки после отправки формы на сервер.

Атрибут `Compare` осуществляет сравнение сохраняемого значения с каким-либо из выбранных разработчиком. Этот атрибут удобно проверять при введении пользователем телефонов, индексов или почтовых адресов. При этом сравнении происходит с помощью регулярных выражений. При несовпадении вводимого значения с шаблонов в выражении пользователю возвращается ошибка с сообщением об этом.