# Экспериментальная часть

Для проверки корректности разработанного программного комплекса планируется проведение следующих видов тестирования:

1. тестирование корректности работы WEB API методом черного ящика;
2. модульное тестирование;
3. кроссбраузерное тестирование интерфейсов пользовательской части.

## Тестирование корректности работы WEB API методом черного ящика

Тестирование методом черного ящика – это функциональное и нефункциональное тестирование без доступа к внутренней структуре компонентов системы [?]. Метод тестирования «черного ящика» – процедура получения и выбора тестовых случаев на основе анализа спецификации (функциональной или нефункциональной), компонентов или системы без ссылки на их внутреннее устройство.

В рамках данного тестирования будут проверены следующие контроллеры WEB API:

1. messages;
2. users;
3. contacts;
4. files;
5. delivery-services;
6. delivery-queues;
7. accesses;
8. sales.

Пример тестов представлены в таблице 1.1.

Таблица 1.1.

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Входные данные | Ожидаемый результат | Соответствие полученного результата ожидаемому |
| 1 | Запрос: GET https://localhost:44306/api/messages/list HTTP/1.1 | {  "attachedFiles": null,  "deliveryQueue": {  "deliveryQueueId": 2,  "sendingIntervalSec": 40  },  "chosenDeliveryService": {  "deliveryServiceId": 3,  "deliveryServiceName": "SendGrid",  "standartPriority": 1,  "connectionString": "'sendGrid road'"  },  "usedDeliveryService": null,  "user": {  "userId": 4,  "userName": "Evgeny Sukharev",  "email": "r0bari@yandex.ru",  "isActive": true,  "role": 2  },  "messageId": 7,  "theme": "Тема",  "body": "Обычный текст",  "destinationDate": "2021-01-16T21:05:00",  "destinationEmail": "r.pattinson@planfact.io",  "size": 50,  "isScheduled": false,  "scheduleDate": null,  "isSent": false,  "deliveryQueueId": 2,  "chosenDeliveryServiceId": 3,  "usedDeliveryServiceId": null,  "deliveryStatus": 1,  "userId": 4  } | Да |
| 2 | Запрос: POST <https://localhost:44306/api/auth/sign-in>  Body:  {  "email": "r0bari@yandex.ru",  "password": "Eviguf@77"  } | {  "data": {  "userId": 4,  "userName": "Evgeny Sukharev",  "email": "r0bari@yandex.ru",  "isActive": true,  "role": 2  },  "isSuccess": true,  "errorMessage": null  } | Да |
| 3 | Запрос: GET <https://localhost:44306/api/contacts/3> | {  "data": {  "user": {  "userId": 4,  "userName": "Evgeny Sukharev",  "email": "r0bari@yandex.ru",  "isActive": true,  "role": 2  },  "contactId": 3,  "userId": 4,  "contactEmail": "maxonfjvipon@yandex.ru",  "contactName": "Максим Трунников"  },  "isSuccess": true,  "errorMessage": null  } | Да |

## Модульное тестирование

Модульное тестирование, или юнит-тестирование (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы[?].

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

В рамках модульного тестирования будут созданы новые проекты DeliveryRely.ServiceLayer.Tests и DeliveryRely.Domain.Tests.

В проекте Delivery.Domain.ServiceLayer.Tests содержатся тесты для методов класса Hash и других вспомогательных классов, таких как AutoMapper. Пример модульного теста из данного проекта представлен в листинге 1.

Листинг 1. Модульный тест механизма хэширования

|  |
| --- |
| using Microsoft.VisualStudio.TestTools.UnitTesting;  namespace DeliveryRely.ServiceLayer.Tests  {  [TestClass]  public class HashSHA512Tests  {  [TestMethod]  public void MakeHashTest()  {  string password = "Eviguf@77";  string passwordHash = "dba23bd80b85edfd9ea2f28e4cc4bb953" +  "86df6b0183567ca3179cb156120f5c4" +  "0312fae40f7ad74431ca921ce4c2c27" +  "d4c92b604051bf090947649801dc54e5e";  string actualPasswordHash = new HashSHA512().MakeHash(password);  Assert.Equals(passwordHash, actualPasswordHash);  }  }  } |

В проекте DeliveryRely.Domain.Tests содержатся тесты для методов получения (методы Get…), добавления (методы Insert…), изменения (методы Update…) и удаления (методы Delete…) класса EFRepository. Пример теста представлен в листинге 2.

Листинг 2. Модульный тест контактов

|  |
| --- |
| using Domain.Models.Contacts;  using Domain.Models.Users;  using Microsoft.VisualStudio.TestTools.UnitTesting;  namespace DeliveryRely.Domain.Tests  {  [TestClass]  public class ContactTest  {  [TestMethod]  public void FieldsTest()  {  var contact = new Contact  {  ContactId = 1,  ContactName = "contact",  ContactEmail = "contact@yandex.ru",  UserId = 4,  User = new User  {  Email = "user@yandex.ru"  }  };  Assert.AreEqual("contact", contact.ContactName);  Assert.AreEqual("contact@yandex.ru", contact.ContactEmail);  Assert.AreEqual(4, contact.UserId);  Assert.AreEqual("user@yandex.ru", contact.User.Email);  }  }  } |

## Кроссбраузерное тестирование интерфейсов пользовательской части

Тестирование кроссбраузерности — вид тестирования, направленный на поддержку и правильное полное отображение программного продукта в разных браузерах, мобильных устройствах, планшетах, экранах различного размера[?].

В рамках данного тестирования в самых распространенных браузерах были проверены основные интерфейсы клиентской части.

Список проверяемых браузеров:

1. Google Chrome 89.0;
2. Safari 14;
3. Mozilla Firefox 91.6 ESR;
4. Opera 12.

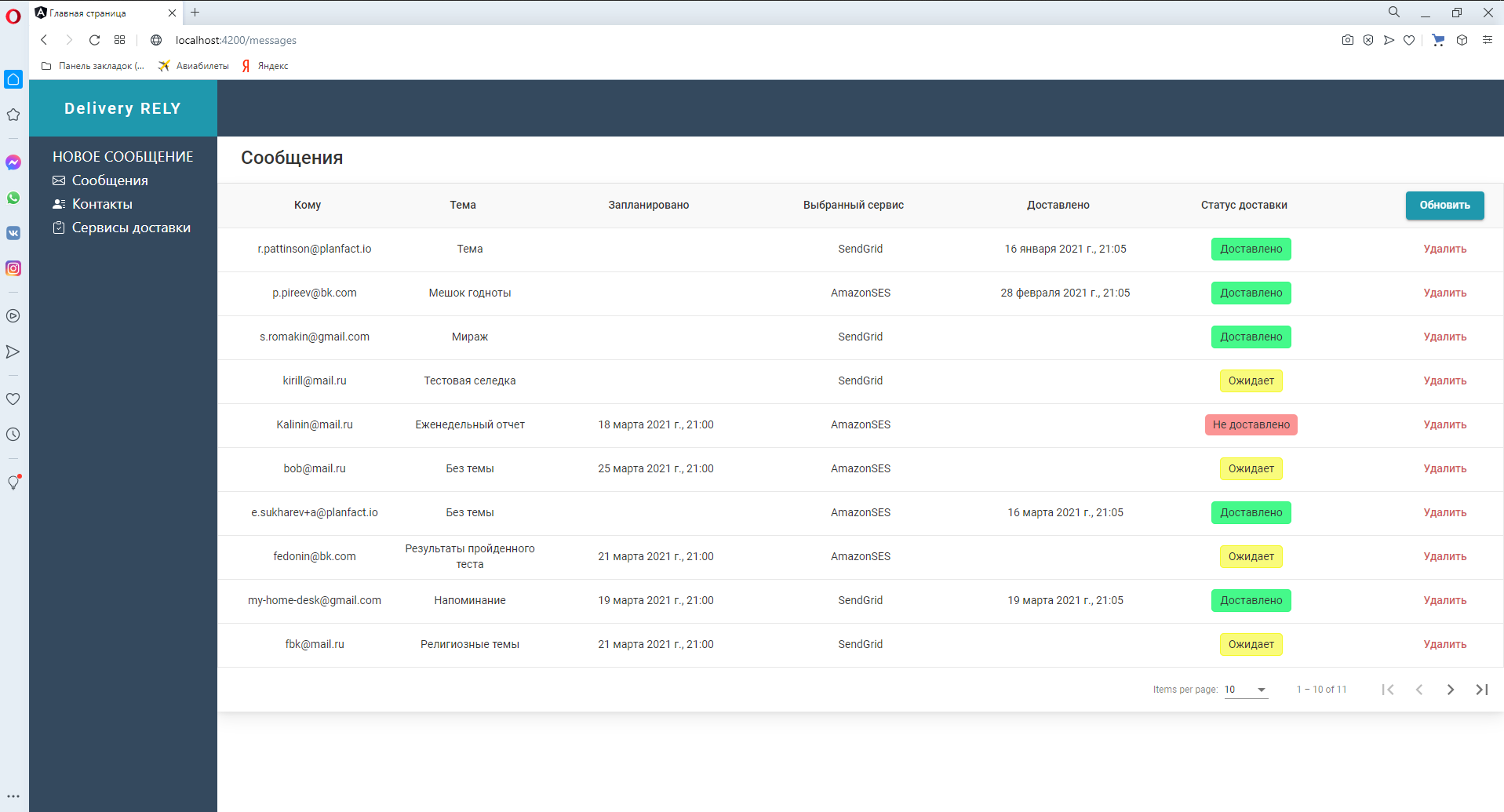
Список сравниваемых интерфейсов:

1. страница авторизации;
2. главная страница;
3. список сервисов доставки;
4. список сообщений;
5. список контактов;
6. страница создания сообщения.

В качестве примера приводятся скриншоты интерфейсов страницы сообщений в браузерах Google Chrome (рис. 1) и Opera (рис. 2).



***Рис. 1. Список сообщений в Google Chrome***

******

***Рис. 2. Страница сообщений в Opera***