Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования «Петрозаводский государственный университет»

Институт математики и информационных технологий
Кафедра информатики и математического обеспечения

Подгорный Никон Игоревич

Отчет о тестировании сервиса авторизации «Vexillum auth»

Направление 09.03.04 — Программная инженерия

Преподаватель: к.ф.–м.н., доцент К.А. Кулаков

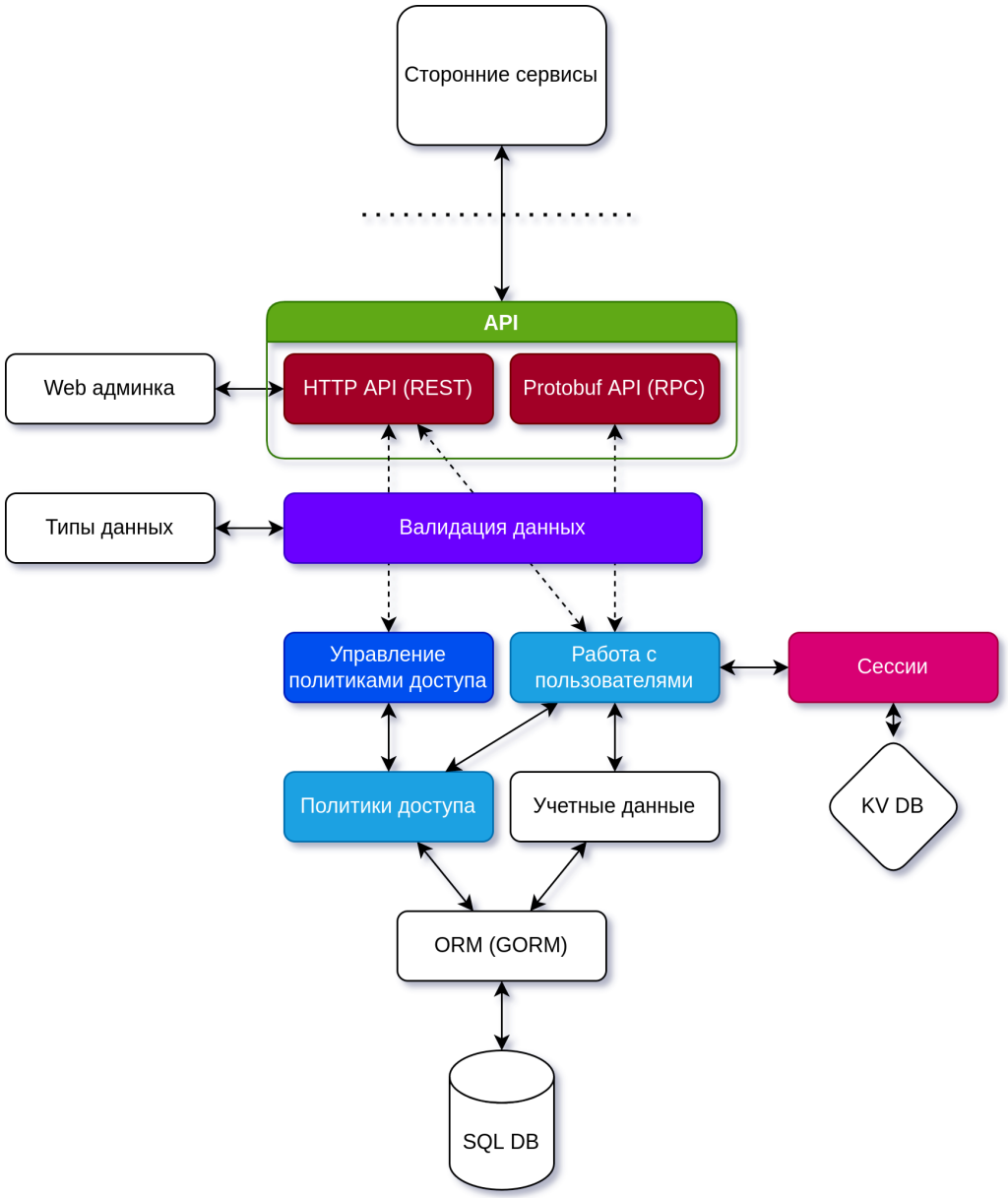
Петрозаводск — 2022

Содержание

1 Объект тестирования

Объект тестирования – сервис авторизации пользователей в приложении «Vexillum». Сервис реализует функции аутентификации и авторизации конечных пользователей приложения, а также функции для администрирования.

1.1 Архитектура системы



1.1.1 Модули и подсистемы

1. server – модуль обработки запросов пользователей и администраторов;
   (a) handlers – подмодуль с обработчиками запросов;
   (b) binders – подмодуль для парсинга и валидации запросов.

2. user-managment – модуль для управления пользователями;
3. policy-managment – модуль для управления политиками доступа;
4. session-managment – модуль для управления сессиями пользователей;
5. models – вспомогательный модуль описывающий модели данных;
6. confirmer – вспомогательный модуль для подтверждения контактных данных пользователей (почта);
7. security – вспомогательный модуль реализующий функции шифрования, подписи и генерации случайных значений.

1.2 Функции объекта тестирования

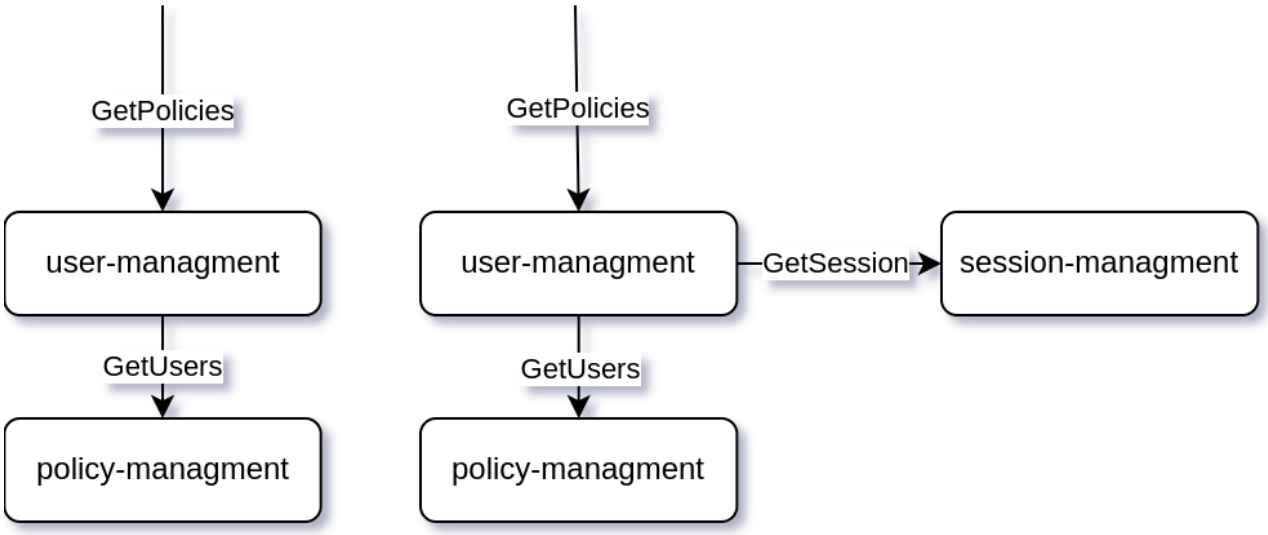
1. Аутентификация пользователей
   (a) Регистрация новых пользователей
   (b) Аутентификация пользователей по логину и паролю
   (c) Деаутентификация пользователей
2. Авторизация пользователей
   (a) Предоставление и проверка прав на выполнение пользователем действия в системе
3. Управление пользователелями
   (a) От имени пользователя
      i. Смена имени пользователя
      ii. Смена пароля
      iii. Загрузка изображения профиля
      iv. Редактирование дополнительной информации
   (b) От имени администратора
      i. Редактирование структуры пользователя (какие поля есть у сущности)
      ii. Управление правами доступа пользователей
4. Получение информации о пользователях

1.2.1 Тестируемые функции

Блочное тестирование проводится для ключевых функций, которые напрямую влияют на работу конечных пользователей. Тестируются следующие функций:

1. Создание пользователя – метод CreateUser(username string, password string): (*User, error)
2. Удаление пользователя – метод User.Delete(): error
3. Добавление политики пользователю – метод User.AddPolicy(policy *Policy): error
4. Удаление политики пользователю – метод User.RemovePolicy(policy *Policy): error
5. Получение сессии пользователя – метод User.GetSession(token string): (*Session, error)
6. Проверить получение политик по сессии пользователя – метод Session.GetPolicies(): ([]*Policy, error)
7. Изменение пользователей – метод User.Edit(username string, password string): error
8. Изменение группы – метод Group.Edit(name string): error
9. Удаления пользователя из группы – User.RemoveFromGroup(group *group): error

Интеграционное тестирование проводится для следующих связей:



2 Стратегия тестирования

2.1 Стратегия блочного тестирования

Блочное тестирование будет проводиться при помощи стандартного модуля Go "testing". Код блочных тестов хранится рядом с кодом модулей, название файлов с кодом тестов должно заканчиваться на «_test.go».

Пример кода блочного теста:

```
1  func TestUserCreateSame(t *testing.T) {
2      err := um.CreateUser(models.User{
3          Username: "testX",
4          Contact:  "contactX",
5      })
6      if err != nil {
7          t.Errorf("Can't create user: %s", err.Error())
8      }
9
10     err2 := um.CreateUser(models.User{
11         Username: "testX",
12         Contact:  "contactY",
13     })
14     if err2 == nil {
15         t.Error("It must be imposible to crate user with same username")
16     }
17 }
```

Пример кода интеграционного теста:

```
1  func IntegrationTestUserToPolicy(t *testing.T) {
2      user1 := GetTestUser()
3      policySet1 := GetTestPolicySet()
4
5      policyManager := NewPolicManager(user1)
6
7      user.UserAddPolicies(policySet1)
8
9      actualPolicySet := user1.GetPolicies()
```

```
10
11      if actualPolicySet != policySet1 {
12          t.Error("Can't get policy by user")
13      }
14
15      usersWithP1 = policyManager.GetUsers(policySet1[1])
16
17      if !contains(usersWithP1, user) {
18          t.Error("Can't get user by policy")
19      }
20  }
```

Запуск блочных тестов:

```
1  go test -v
```

Пример вывода блочных тестов:

```
1  === RUN   TestOldToken
2  --- PASS: TestOldToken (0.1s)
3  === RUN   TestInvalidPolicyKey
4  --- PASS: TestInvalidPolicyKey (0.03s)
5  ...
6  === RUN   TestUserZombieSession
7  --- PASS: TestUserZombieSession (0.1s)
8  PASS
9  ok      github.com/VexillumDev/Vexillum-auth    0.5s
```

2.1.1 Критерии прохождения блочного тестирования

1. тест считается пройденным, если все его кейсы завершились успешно;

2. тестирование считается пройденным если все тесты пройдены.

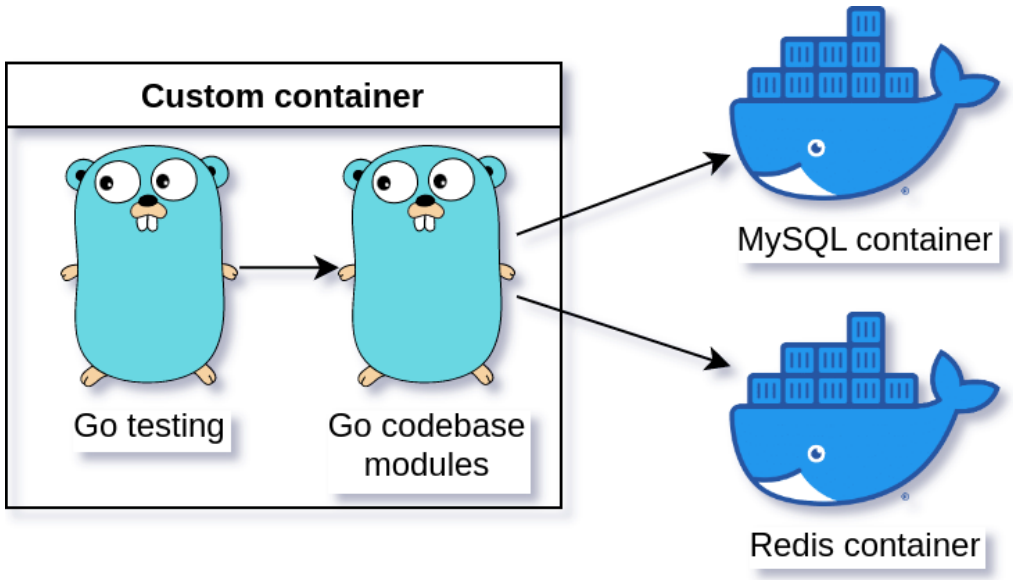
2.2 Стратегия интеграционного тестирования

Интеграционное тестирование также проводится при помощи модуля Go "testing". Для развертывания тестовых БД используется Docker Compose. Отчеты о покрытии для ин-

теграционного тестирования создаются отдельно. Содержимое баз данных заполняется перед проведением тестов.

Используется интеграция "сверху вниз заглушки низкоуровневых модулей реализованы в файлах с кодом тестов.

Окружение для проведения интеграционного тестирования:



2.2.1 Критерии прохождения интеграционного тестирования

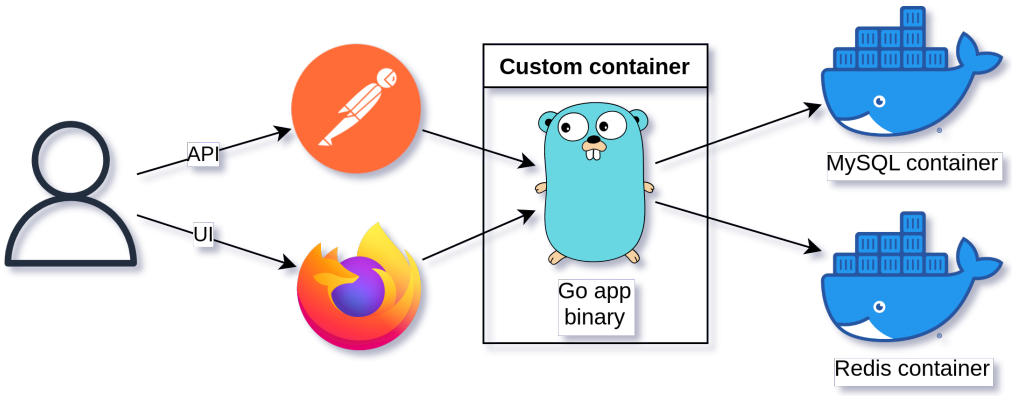
1. тест считается пройденным, если все его кейсы завершились успешно;
   (a) интеграционным тест считается не пройденным, если хотя бы один блочный тест низкоуровневых модулей не пройден.
2. тестирование считается пройденным если все тесты пройдены.

2.3 Стратегия аттестационного тестирования

Аттестационное тестирование веб-интерфейса проводится в ручном режиме в браузерах на базе Gecko и Chromium.

Аттестационное тестирование API проводится при помощи Postman. Для развертывания тестового окружения используется Docker Compose.

Окружение для проведения аттестационного тестирования:



Аттестационное тестирование проводится для следующих методов API:

1. /api/signup
2. /api/signin
3. /api/user
4. /api/signout

Тестирование API производится при помощи Postman.

2.4 Стратегия нагрузочного тестирования

Нагрузочное тестирование HTTP API производится при помощи инструмента «ddosify». Для развертывания тестового окружения используется Docker Compose.

Измеряется время ответа сервера в зависимости от числа уникальных запросов в секунду (RPS). Запросы генерируются от лица независимых тестовых пользователей.

Тестирование проводится для:

1. 1 запроса в секунду
2. 30 запросов в секунду
3. 300 запросов в секунду

2.4.1 Критерии прохождения нагрузочного тестирования

1. время ответа на запросы пользователей не должно превышать 2000 мс при 30 rps;
2. на любой запрос к серверу возвращается ответ (возможно с ошибкой);
3. потребление ресурсов линейно зависит от нагрузки.

3 Переход между этапами тестирования

Блочное тестирование выполняется автоматически при любом изменении в репозитории проекта. Блочное тестирование продолжает выполнятся, если какой-либо из тестов не пройден.

Интеграционное тестирование модуля проводится после прохождения блочного для модулей более низкого уровня.

Аттестационное тестирование проводится при окончании разработки очередной функции системы и прохождения интеграционного тестирования. Также аттестационное тестирование проводится перед окончанием разработки проекта.

Нагрузочное тестирование проводится после прохождения аттестационного тестирования.

4 Детальный план тестов

4.1 Блочное тестирование

ID теста	Б1 (TestUserCreate)
Объект тестирования	Модуль управления пользователями «user-managment»
Цель тестирования	Проверить возможность создания нового пользователя
Входные данные	Вызов метода CreateUser с параметрами username=name password=pass
Ожидаемый результат	CreateUser возвращает структуру пользователя и пустую ошибку nil
Тип	Позитивный

ID теста	Б2 (TestUserCreateSame)
Объект тестирования	Модуль управления пользователями «user-managment»
Цель тестирования	Проверить возможность создания несколько пользователей с одинаковыми именами

Входные данные	Вызов метода CreateUser с параметрами username=name password=pass два раза подряд
Ожидаемый результат	Второй вызов метода возвращает nil в качестве пользователя и ошибку UserExistsError
Тип	Негативный

ID теста	Б3 (TestUserAddPolicy)
Объект тестирования	Модуль управления политиками «policy-managment»
Цель тестирования	Проверить возможность добавления политики
Входные данные	Вызывается метод User.AddPolicy с параметром policy=указатель на структуру Policy
Ожидаемый результат	Метод возвращает nil (пустая ошибка)
Тип	Позитивный

ID теста	Б4 (TestUserZombieSession)
Объект тестирования	Модуль управления сессиями «session-managment»
Цель тестирования	Проверить инвалидацию сессий удаленных пользователей
Входные данные	Вызов метода User.GetSession с параметром token=валидный токен для удаленного пользователя
Ожидаемый результат	Метод возвращает nil в качестве сессии и ошибку InvalidUserSession
Тип	Негативный

ID теста	Б5 (TestEditVoidUser)
Объект тестирования	Модуль управления пользователями «user-managment»
Цель тестирования	Проверить возможность изменения удаленных пользователей
Входные данные	Вызов метода User.Edit с произвольными параметрами для удаленного пользователя

Ожидаемый результат	Возврат ошибки «InvalidUserError»
Тип	Негативный

ID теста	Б6 (TestEditVoidGroup)
Объект тестирования	Модуль управления политиками «policy-managment»
Цель тестирования	Проверить возможность изменения удаленных групп
Входные данные	Вызов метода Group.Edit с произвольными параметрами для удаленной группы
Ожидаемый результат	Возврат ошибки «InvalidGroupError»
Тип	Негативный

ID теста	Б7 (TestUserNoRights)
Объект тестирования	Модуль управления политиками «policy-managment»
Цель тестирования	Проверить возможность появления пользователей без групп
Входные данные	Существует пользователь user1 с одной группой прав group1, вызывается метод user1.RemoveGroup(group1)
Ожидаемый результат	Возврат ошибки «LastGroupError»
Тип	Негативный

ID теста	Б8 (TestDeleteLastAdmin)
Объект тестирования	Модуль управления политиками «policy-managment»
Цель тестирования	Проверить возможность удаления всех администраторов
Входные данные	Существует единственный администратор admin1 и произвольный список пользователей. Вызывается метод admin1.Delete()
Ожидаемый результат	Возврат ошибки «LastAdminError»
Тип	Негативный

ID теста	Б9 (TestOldToken)

Объект тестирования	Модуль управления сессиями «session-managment»
Цель тестирования	Проверить обработку старых токенов
Входные данные	Существует токен доступа token1 с просроченным сроком жизни, вызывается метод User.GetSession(token1)
Ожидаемый результат	Возврат ошибки «InvalidTokenError»
Тип	Позитивный

ID теста	Б10 (TestInvalidPolicyKey)
Объект тестирования	Модуль управления сессиями «session-managment»
Цель тестирования	Проверить обработку токенов с невалидными политиками
Входные данные	Существует токен доступа token1 с невалидным ключом политики, вызывается метод User.GetSession(token1)
Ожидаемый результат	Возврат ошибки «InvalidPolicyKeyError»
Тип	Негативный

ID теста	Б11 (TestZombieService)
Объект тестирования	Модуль управления сессиями «session-managment»
Цель тестирования	Проверить обработку политик для невалидных сервисов
Входные данные	Существует токен доступа token1 с невалидным ключом сервиса, вызывается метод User.GetSession(token1)
Ожидаемый результат	Возврат ошибки «invalidServiceKeyError»
Тип	Негативный

4.2 Интеграционное тестирование

ID теста	И1 (TestUserToPolicy)
Объект тестирования	Связь модулей «user-managment» и «policy-managment»

Цель тестирования	Проверить интеграцию модулей пользователей и политик
Входные данные	Существует пользователь user1 для которого есть набор политик [p1, p2, p3]. Вызывается метод user1.GetPolicies(), внутри которого вызывается метод p1.GetUsers()
Ожидаемый результат	Возвращен список [p1, p2, p3] Возвращен список содержащий user1
Тип	Позитивный

ID теста	И2 (TestUserToPolicyToSession)
Объект тестирования	Связь модулей «user-managment», «policy-managment» и «session-managment»
Цель тестирования	Проверить интеграцию модулей пользователей, политик и сессий
Входные данные	Существует пользователь user1 с токеном доступа token, для этого токена существует сессия session1, вызывается метод user1.GetPolicy(token1), внутри которого вызывается вызывается метод session1.GetUser() и session.GetPolicies()
Ожидаемый результат	Можно получить «User» по «Session» Можно получить список «Policy» по «Session»
Тип	Позитивный

ID теста	И3 (TestUserToPolicyToInvalidSession)
Объект тестирования	Связь модулей «user-managment», «policy-managment» и «session-managment»
Цель тестирования	Проверить интеграцию модулей пользователей, политик и сессий при появлении устаревшей сессии

Входные данные	Существует пользователь user1 с токеном доступа token, для этого токена существует сессия session1, которая является устаревшей, вызывается метод user1.GetPolicy(token1), внутри которого вызывается вызывается метод session1.GetUser() и session.GetPolicies()
Ожидаемый результат	Попытка получения «User» по «Session» возвращает ошибку «InvalidSession» Попытка получения списка «Policy» по «Session» возвращает ошибку «InvalidSession»
Тип	Негативный

4.3 Аттестационное тестирование

ID теста	A1
Объект тестирования	Метод API /api/signin
Цель тестирования	Проверить возможность входа пользователя в систему
Входные данные	БД содержит пользователя с именем user1 и паролем pass1. Выполняется POST /api/signin с полями username=user1 password=pass1
Ожидаемый результат	Ответ 200 Ок содержащий токен доступа accesstoken
Тип	Позитивный

ID теста	A2
Объект тестирования	Метод API /api/signup
Цель тестирования	Проверить возможность регистрации пользователя
Входные данные	БД не содержит пользователя с именем user1 и паролем pass1. Выполняется POST /api/signup с полями username=user1 password=pass1 passwordconfigrm=pass1
Ожидаемый результат	Ответ 303 перенаправление на /api/login
Тип	Позитивный

ID теста	A3
Объект тестирования	Метод API /api/user
Цель тестирования	Проверить возможность получения информации о пользователе по токену
Входные данные	БД содержит пользователя с именем user1 и токеном доступа token1. Выполняется GET /api/user с заголовком Authorization=token
Ожидаемый результат	Ответ 200 Ok с телом - json структура пользователя
Тип	Позитивный

ID теста	A4
Объект тестирования	Метод API /api/signout
Цель тестирования	Проверить возможность выхода из системы
Входные данные	БД содержит пользователя с именем user1 и токеном доступа token1. Выполняется POST /api/logout с заголовком Authorization=token. Затем выполняется тест A3
Ожидаемый результат	Ответ 200 Ok, тест A3 должен завершится с ошибкой 401
Тип	Позитивный

4.4 Нагрузочное тестирование

ID теста	H1
Цель тестирования	Измерить время ответа сервера
Входные данные	1 запрос в секунду от 1 пользователя
Ожидаемый результат	Время ответа на превышает 2000мс

ID теста	H2
Цель тестирования	Измерить время ответа сервера
Входные данные	10 запросов в секунду от 10 пользователей
Ожидаемый результат	Время ответа на превышает 2000мс

ID теста	Н3
Цель тестирования	Измерить время ответа сервера
Входные данные	20 запросов в секунду от 20 пользователей
Ожидаемый результат	Время ответа на превышает 2000мс

ID теста	Н4
Цель тестирования	Измерить время ответа сервера
Входные данные	30 запросов в секунду от 30 пользователей
Ожидаемый результат	Время ответа на превышает 2000мс

ID теста	Н5
Цель тестирования	Измерить время ответа сервера
Входные данные	100 запросов в секунду от 100 пользователей
Ожидаемый результат	Время ответа на превышает 2000мс

ID теста	Н6
Цель тестирования	Измерить время ответа сервера
Входные данные	300 запросов в секунду от 300 пользователей
Ожидаемый результат	Время ответа на превышает 2000мс

5 Результаты тестирования

5.1 Блочное тестирование

ID теста	**Дата**	**Результат**	**Фактический результат**	**Отчёт**
Б1	07.12.22	Пройден		
Б2	07.12.22	Пройден		
Б3	07.12.22	Пройден		
Б4	07.12.22	Пройден		
Б5	07.12.22	Пройден		

Б6	07.12.22	Пройден		
Б7	07.12.22	Пройден		
Б8	07.12.22	Не пройден	Метод User.Delete() завершился без ошибки	ошибка 1: можно удалить всех администраторов
Б9	07.12.22	Пройден		
Б10	07.12.22	Пройден		
Б11	07.12.22	Не пройден	Метод Session.GetPolicies() вернул список политик и завершился без ошибки	ошибка 2: политики доступа могут существовать без привязки к сервисам

5.1.1 Покрытие

Модуль	Покрытие
server	12%
user-managment	71%
policy-managment	64%
session-managment	65%
security	93%

5.2 Интеграционное тестирование

ID теста	**Дата**	**Результат**	**Фактический результат**	**Отчёт**
И1	07.12.22	Пройден		
И2	07.12.22	Не пройден	При вызове метода Session.GetPolicies() из модуля User программа заврешается с ошибкой «invalid policy»	ошибка 3: неверно передан указатель на структуру пользователя

И3	07.12.22	Пройден		

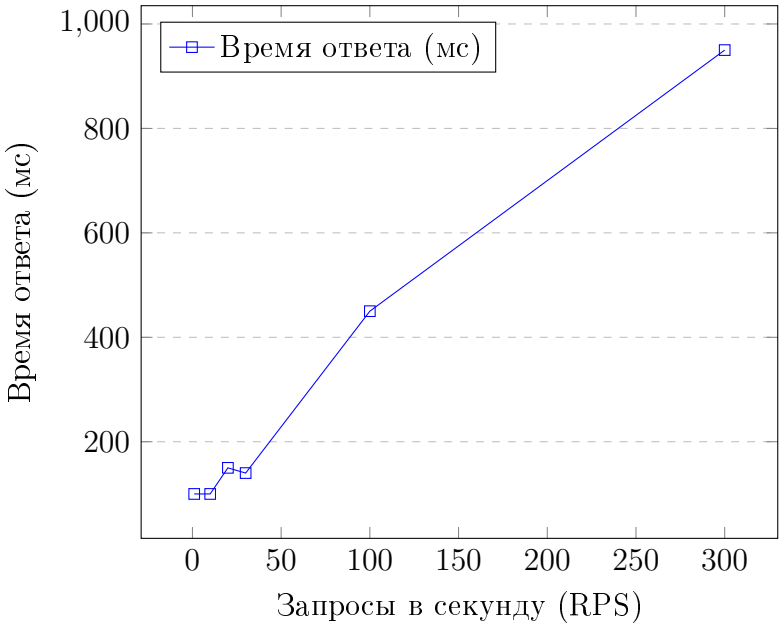
5.3 Аттестационное тестирование

ID теста	Дата	Результат	Фактический результат	Отчёт
А1	12.12.22	Пройден		
А2	12.12.22	Пройден		
А3	12.12.22	Пройден		
А4	12.12.22	Пройден		

5.4 Нагрузочное тестирование

ID теста	Дата	Результат	Отчёт
Н1	11.12.22	Время ответа – 100 мс	
Н2	11.12.22	Время ответа – 100 мс	
Н3	11.12.22	Время ответа – 150 мс	
Н4	11.12.22	Время ответа – 140 мс	
Н5	11.12.22	Время ответа – 450 мс	
Н6	11.12.22	Время ответа – 950 мс	

Время ответа в зависимости от количества запросов



6 Заключение

Проведено блочное, интеграционное, аттестационное и нагрузочное тестирование сервиса Vexillum auth, исправлены найденные ошибки, составлен отчёт о прохождении тестирования.