

**МИНОБРНАУКИ РОССИИ  
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ  
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МОЕВМ**

**ОТЧЕТ**

**по «UI-тестирование» практике**

**Тема: UI-тестирование на системе gmail**

Студент гр. 3388

\_\_\_\_\_

Тимошук Е.  
Потапов Р.  
Кулач Д.

Руководитель

\_\_\_\_\_

Шевелева А. М.

Санкт-Петербург

2025

## ЗАДАНИЕ

### НА «UI-тестирование» ПРАКТИКУ

Студенты: Тимошук Е. Потапов Р. Кулач Д.

Группа 3388

Тема практики: **Gmail.com**

#### Задание на практику:

Написать архитектуру для реализации 10 UI-тестов для системы Gmail

Сроки прохождения практики: 26.06.2025 – 05.07.2025

Дата сдачи отчета: 07.07.2025

Дата защиты отчета: 05.07.2025

Студент

---

Тимошук Е.  
Потапов Р.  
Кулач Д.

Руководитель

---

Шевелева А.М.

## **АННОТАЦИЯ**

В данном отчете представлены результаты прохождения практики, посвященной UI-тестированию интерфейса Gmail. Были рассмотрены основные элементы пользовательского интерфейса, проведено функциональное и юзабилити-тестирование, а также проанализирована работа сервиса в разных браузерах и на различных устройствах.

В ходе тестирования использовались ручные методы проверки и инструменты разработчика (DevTools). Составлены тест-кейсы, выявлены возможные баги и предложены рекомендации по улучшению интерфейса.

Отчет содержит описание методологии тестирования, примеры тест-кейсов с результатами, список обнаруженных проблем и выводы по проделанной работе

## **SUMMARY**

This report presents the results of the practice dedicated to UI testing of the Gmail interface. The main elements of the user interface were reviewed, functional and usability testing was conducted, and the service's performance in different browsers and on various devices was analyzed.

During the testing, manual verification methods and developer tools (DevTools) were used. Test cases were created, possible bugs were identified, and recommendations for improving the interface were proposed.

The report contains a description of the testing methodology, examples of test cases with results, a list of detected issues, and conclusions on the work performed.

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	5
ВЫПОЛНЕНИЕ РАБОТЫ.....	6
1. РЕАЛИЗУЕМЫЕ ТЕСТЫ.....	6
2. ОПИСАНИЕ КЛАССОВ И МЕТОДОВ .....	8
2.1 Классы и методы элементов страницы.....	8
2.2 Классы и методы страницы .....	10
2.3 Базовый класс для тестов .....	14
2.4 Классы и методы тестов.....	15
3. АРХИТЕКТУРА ПРОЕКТА.....	18
4. ТЕСТИРОВАНИЕ .....	20
4.1 Тестирование авторизации .....	20
4.2 Тестирование всех тестов .....	22
UML – диаграмма .....	26
ЗАКЛЮЧЕНИЕ.....	27
СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ .....	28

## ВВЕДЕНИЕ

Настоящий отчет представляет результаты прохождения практики по UI-тестированию, в ходе которой проводилось комплексное исследование пользовательского интерфейса почтового сервиса (Gmail). Актуальность работы обусловлена необходимостью обеспечения высокого уровня удобства, стабильности и отказоустойчивости веб-приложений, особенно таких массовых, как Gmail, который ежедневно используют миллионы людей по всему миру.

### Цель и задачи исследования

Цель работы – изучение методов UI-тестирования на примере Gmail, включая проверку корректности отображения элементов, удобства взаимодействия и функциональной надежности интерфейса.

### Задачи:

- Анализ интерфейса Gmail – изучение структуры, визуального оформления и логики взаимодействия с основными элементами (письма, фильтры, контакты, настройки).
- Проверка юзабилити и функциональности – оценка удобства навигации, скорости выполнения операций, обработки ошибок и адаптивности интерфейса.
- Составление тест-кейсов – разработка сценариев проверки ключевых функций с фиксацией ожидаемых и фактических результатов.

## **ВЫПОЛНЕНИЕ РАБОТЫ**

### **1. РЕАЛИЗУЕМЫЕ ТЕСТЫ**

1. Установка фильтров для писем – тест проверяет создание и работу фильтра в почтовой системе. Пользователь авторизуется, заходит в настройки, создаёт новый фильтр с условием: если письмо от `example@domain.com`, перемещать его в папку «Важное». Ожидается, что письмо от указанного отправителя автоматически попадает в нужную папку. (Рисунок 4- Создание фильтра)
2. Изменение имени и фамилии пользователя – тест проверяет редактирование профиля. Пользователь авторизуется, заходит в настройки профиля, меняет имя и фамилию (например, на "Иван Иванов") и сохраняет изменения. Ожидается, что данные обновляются корректно. (Рисунок 5 - Форма по заполнение ФИО)
3. Удаление письма – тест проверяет удаление и восстановление письма. Пользователь авторизуется, выбирает письмо во «Входящих», удаляет его (оно перемещается в «Корзину»), затем может восстановить. Ожидается, что письмо исчезает из исходной папки, появляется в «Корзине», а при восстановлении возвращается обратно. (Рисунок 6 - Удаление письма)
4. Добавление контакта – тест проверяет создание нового контакта. Пользователь авторизуется, заходит в раздел «Контакты», создаёт новый, заполняет имя («Даниил») и email (`kulachdv@gmail.com`), сохраняет. Ожидается, что контакт появляется в списке. (Рисунок 7- Добавление Kontakta)
5. Проверка восстановления сессии после разрыва соединения – тест проверяет автосохранение данных при обрыве интернета. Пользователь авторизуется, создаёт письмо с текстом и вложением, соединение искусственно обрывается. После восстановления сети система должна сохранить черновик.

6. Смена языка – тест проверяет изменение языка интерфейса. Пользователь авторизуется, заходит в настройки аккаунта, выбирает другой язык в разделе «Общие настройки веб-интерфейса». Ожидается, что интерфейс переключается на выбранный язык. (Рисунок 8 - Смена языка)
7. Отправка письма несуществующему адресату – тест проверяет обработку некорректного email. Пользователь авторизуется, создаёт письмо и вводит неполный адрес (например, user@). Система должна либо заблокировать отправку, либо показать ошибку. (Рисунок 9 - Отправка письма)
8. Постановка задачи – тест проверяет создание новой задачи. Пользователь авторизуется, заходит в раздел «Задачи», добавляет новую с названием и сохраняет. Ожидается, что задача появляется в списке. (Рисунок 10 - Добавление задачи)
9. Пометить письмо как прочитанное/непрочитанное – тест проверяет изменение статуса письма. Пользователь авторизуется, находит непрочитанное письмо во «Входящих», помечает его как прочитанное через контекстное меню. Ожидается, что статус меняется. (Рисунок 11 - Письма с отметкой)
10. Поиск, по ключевым словам, в письмах – тест проверяет работу поиска. Пользователь авторизуется, ищет реальное слово («Здравствуйте») и несуществующее («Несуществующее Слово»), проверяет результаты с фильтрами «Тема» и «Слова в письме». Ожидается корректное отображение найденных писем и пустой результат для несуществующего запроса. (Рисунок 12 - Письма с пометкой "Здравствуйте")

## 2. ОПИСАНИЕ КЛАССОВ И МЕТОДОВ

### 2.1 Классы и методы элементов страницы

Класс ``BaseComponent`` (Утилита для работы с локаторами)

Методы этого класса служат фабриками для создания объектов типа ``By``, которые используются в Selenium для поиска элементов на веб-страницах. Каждый метод возвращает конкретную стратегию локации элемента:

1. ``byClass()``

Создает локатор, находящий элементы по их CSS-классу. Принимает строку с именем класса (без точки) и возвращает объект ``By.className``. Ищет элементы, у которых атрибут ``class`` содержит указанное значение.

2. ``byName()``

Генерирует локатор для поиска элементов по значению атрибута ``name``. Возвращает объект ``By.name``. Особенно полезен для работы с формами (поля ввода, кнопки).

3. ``byCss()``

Создает локатор на основе CSS-селектора. Возвращает объект ``By.cssSelector``. Поддерживает сложные селекторы (комбинаторы, псевдо классы), что позволяет точно находить элементы по их положению в DOM.

4. ``byXpath()``

Формирует локатор с помощью XPath-выражения. Возвращает объект ``By.xpath``. XPath обеспечивает максимальную гибкость поиска, включая навигацию по иерархии элементов и фильтрацию по любым атрибутам.

5. ``byText()``

Специальный метод для поиска элементов по точному текстовому содержимому. Внутри использует XPath-шаблон ``//*[text()='...']``. Находит только элементы, чей текст полностью совпадает с переданным значением.

6. ``byLinkText()``



Оптимизированный метод для поиска гиперссылок (`<a>` тегов) по их тексту. Возвращает объект `By.linkText`. Работает только с текстом внутри тега ссылки.

Класс `BaseElement` (Обёртка для веб-элементов)

Инкапсулирует стандартный `WebElement` Selenium, добавляя автоматическую обработку ожиданий и логирование:

1. Конструктор

Принимает два параметра:

- `WebElement` (базовый элемент Selenium)
- `WebDriverWait` (объект для явных ожиданий)

Инициализирует внутреннее состояние обёртки.

2. `click()`

Выполняет безопасный клик по элементу:

- Автоматически дожидается видимости элемента и его активности (кликабельности) через `WebDriverWait`
- Фиксирует действие в системном логе (информация о целевом элементе)
- Вызывает нативный метод `click()` у базового элемента

Гарантирует, что элемент готов к взаимодействию перед выполнением действия.

3. `enterText()`

Обеспечивает надёжный ввод текста:

- Ожидает видимость элемента в DOM
- Логирует операцию с указанием вводимого текста
- Автоматически очищает поле перед вводом
- Передаёт текст базовому элементу через метод `sendKeys()`

Защищает от распространённых ошибок (ввод в скрытые/недоступные поля).

Принцип работы в связке

1. Создание локатора

Класс страницы вызывает методы ``BaseComponent`` для получения локаторов:

```
`By emailLocator = BaseComponent.byName("email")`
```

## 2. Поиск элемента

Через драйвер находится сырой элемент:

```
`WebElement rawElement = driver.findElement(emailLocator)`
```

## 3. Создание обёртки

Сырой элемент оборачивается в ``BaseElement``:

```
`BaseElement emailField = new BaseElement(rawElement, wait)`
```

## 4. Взаимодействие

Тесты используют безопасные методы обёртки:

```
`emailField.enterText("test@example.com")`
```

## 2.2 Классы и методы страницы

### 1. Базовый класс ``BasePage``

Назначение:

Абстрактный класс, содержащий общую логику для всех страниц приложения.

Ключевые методы:

- ``pauseMillis()``

Выполняет асинхронную паузу через `JavaScriptExecutor`. Используется для искусственных задержек.

- ``pauseFiveSeconds()``

Специализированная пауза на 5 секунд (вызов ``pauseMillis(5000)``).

- ``waitForPageLoad()``

Ожидает полной загрузки страницы через проверку состояния ``document.readyState``.

### 2. Класс ``ComposePage`` (Диалог создания письма)

Назначение:

Управление диалогом "Написать письмо" в Gmail.

Ключевые методы:

- ``enterRecipient()``

Вводит адрес получателя в поле "Кому".

- ``enterSubject()``

Заполняет поле "Тема" письма.

- ``enterBody()``

Вводит текст в тело письма.

- ``clickSend()``

Кликает кнопку "Отправить".

### 3. Класс ``ContactsPage`` (Управление контактами)

Назначение:

Работа с разделом "Контакты" внутри Gmail.

Ключевые методы:

- ``clickNewContact ()``

Открывает форму создания нового контакта.

- ``enterName()``

Заполняет поле "Имя" контакта.

- ``enterEmail()``

Вводит email контакта.

- ``waitForSaveAndReturn()``

Ожидает сохранения и возвращается в основной контекст страницы.

### 4. Класс ``FiltersPage`` (Управление фильтрами)

Назначение:

Работа с разделом "Фильтры и заблокированные адреса".

Ключевые методы:

- ``clickCreateNewFilter()``

Иницирует создание нового фильтра.

- ``enterFromAddress()``

Заполняет поле "От" для фильтрации.

- ``clickNeverSpamOption()``

Активирует опцию "Никогда не отправлять в спам".

## 5. Класс `InboxPage` (Входящие письма)

Назначение:

Основная рабочая область Gmail (список писем).

Ключевые методы:

- `getUnreadCount()`

Возвращает количество непрочитанных писем.

- `selectFirstUnread()`

Выбирает первое непрочитанное письмо.

- `clickMarkAsRead()`

Помечает выделенные письма как прочитанные.

- `openContactsPage()`

Переходит в раздел "Контакты" (возвращает `ContactsPage`).

## 6. Класс `LoginPage` (Авторизация)

Назначение:

Обработка процесса входа в аккаунт Gmail.

Ключевые методы:

- `enterEmail()`

Вводит email в поле идентификатора.

- `clickEmailNext()`

Отключает WebDriver-флаг и переходит к вводу пароля.

- `enterPassword()`

Заполняет поле пароля.

## 7. Класс `MyAccountPage` (Управление аккаунтом)

Назначение:

Работа с персональными данными в Google My Account.

Ключевые методы:

- `openNameEdit()`

Открывает форму редактирования имени (с обработкой сложного UI).

- `updateName()`

Обновляет имя и фамилию пользователя.

## 8. Класс `OfflinePage` (Эмуляция сети)

Назначение:

Управление сетевыми условиями через Chrome DevTools Protocol (CDP).

Ключевые методы:

- `emulateOffline()`

Переводит браузер в режим offline.

- `emulateOnline()`

Восстанавливает онлайн-режим.

- `getOfflineErrorText()`

Возвращает текст системного сообщения об ошибке.

## 9. Класс `ProfileMenuPage` (Меню профиля)

Назначение:

Взаимодействие с выпадающим меню профиля.

Ключевой метод:

- `goToMyAccount()`

Переключается в iframe меню, открывает "Мой аккаунт" в новом окне.

## 10. Класс `SettingsPage` (Настройки Gmail)

Назначение:

Управление настройками почты.

Ключевые методы:

- `clickAllSettings()`

Открывает полный интерфейс настроек.

- `selectLanguage()`

Изменяет язык интерфейса через выпадающий список.

## 11. Класс `TasksPage` (Управление задачами)

Назначение:

Работа с виджетом задач внутри Gmail.

Ключевые методы:

- `clickAddTaskButton()`

Открывает форму создания задачи.

- ``enterTaskTitle()``

Заполняет заголовок задачи.

### 2.3 Базовый класс для тестов

Базовый класс ``BaseTest``, который служит основой для всех тестовых классов в проекте автоматизации. Он содержит общую логику, необходимую для запуска и завершения браузера, настройки параметров `WebDriver`, открытия начальной страницы приложения перед каждым тестом, а также предоставляет универсальный метод авторизации в системе.

Класс использует Selenium `WebDriver` (в данном случае — `ChromeDriver`) и применяет подход объектной модели страницы (`Page Object Model`), что делает тесты более структурированными, читаемыми и поддерживаемыми.

Метод ``setUp()``

Этот метод аннотирован как ``@BeforeEach``, то есть он выполняется перед каждым тестовым методом. В нём происходит:

- Инициализация драйвера браузера Google Chrome с настройками, которые позволяют скрыть факт автоматизации от веб-сайта (это помогает избежать блокировок или дополнительных проверок безопасности).

- Отключение некоторых функций, связанных с автоматизацией (``enable-automation``, ``useAutomationExtension``), чтобы сделать поведение браузера похожим на пользовательское.

- Установка максимального времени ожидания для поиска элементов на странице — 30 секунд.

- Открытие базовой страницы входа в Gmail.

Метод ``tearDown()``

Аннотированный как ``@AfterEach``, этот метод выполняется после каждого тестового сценария и отвечает за корректное закрытие браузера. Это важно для предотвращения утечек памяти и обеспечения изоляции между тестами.

Метод ``auth (String email, String password)``

Этот метод реализует логику входа пользователя в систему. Он принимает логин и пароль, взаимодействует с элементами страницы через класс

`LoginPage`, вводит данные и кликает по кнопкам "Далее". После успешного входа возвращает экземпляр класса `InboxPage`, представляющий собой главную страницу почтового ящика. Таким образом, тесты могут сразу продолжить выполнение действий уже на странице Inbox, без повторного написания логики авторизации.

### Классы `LoginPage` и `InboxPage`

Эти классы являются частью паттерна Page Object. Они инкапсулируют действия и элементы конкретных страниц приложения:

- `LoginPage` — содержит методы для работы с полями ввода email и пароля, а также кнопками перехода.

- `InboxPage` — представляет собой страницу входящей почты и предоставляет методы для дальнейших действий с письмами.

## 2.4 Классы и методы тестов

### 1. Базовые классы (Page Objects)

- `BaseTest`

Базовый класс для всех тестов. Содержит общую логику:

- Инициализация драйвера
- Метод авторизации `auth()`
- Общие ожидания и утилиты

- `InboxPage`

Работа с почтовым ящиком:

- `clickSettingsIcon()`: Открывает настройки
- `openContactsPage()`: Переход в контакты
- `selectFirstUnread()`: Выбор первого непрочитанного письма
- `clickDelete()`: Удаление письма
- `getUnreadCount()`: Получение количества непрочитанных писем

- `AdvancedSearchPage`

Расширенный поиск:

- `openSearch()`: Открытие панели поиска

- `typeFirstQuery()`, `typeSecondQuery()`: Ввод запросов
- `clearFirst()`: Очистка поля
- `submitSearch()`: Запуск поиска

- `SettingsPage`

Управление настройками:

- `clickAllSettings()`: Открытие полных настроек
- `selectLanguage()`: Выбор языка
- `clickSaveChanges()`: Сохранение настроек

- `ContactsPage`

Работа с контактами:

- `clickNewContact()`: Создание контакта
- `enterName()`, `enterEmail()`: Заполнение данных
- `saveContact()`: Сохранение контакта

- `FiltersPage`

Управление фильтрами:

- `clickCreateNewFilter()`: Создание фильтра
- `enterFromAddress()`: Ввод адреса отправителя
- `clickNeverSpamOption()`: Активация опции "Не спам"
- `clickFinalCreateFilter()`: Финализация фильтра

- `ComposePage`

Создание писем:

- `enterRecipient()`, `enterSubject()`, `enterBody()`: Заполнение полей
- `clickSend()`: Отправка письма

`OfflinePage`

Тестирование offline-режима:

- `enableNetwork()`, `emulateOffline()`, `emulateOnline()`: Управление

сетевым статусом

- `getOfflineErrorText()`: Получение сообщения об ошибке

## 2. Тестовые классы



- `AdvancedSearchTest`
  - `shouldPerformAllAdvancedSearchScenarios()`: Проверяет 4 сценария поиска с разными параметрами.
- `ChangeLanguageTest`
  - `shouldChangeInterfaceLanguage()`: Меняет язык интерфейса на английский.
- `ContactsTest`
  - `shouldAddNewContact()`: Создает новый контакт.
- `DeleteMessageTest`
  - `shouldDeleteFirstUnreadMessage()`: Удаляет первое непрочитанное письмо.
- `FiltersTest`
  - `shouldCreateNeverSpamFilter()`: Создает фильтр против спама.
- `MessageSenderTest`
  - `shouldSendEmailSuccessfully()`: Отправляет тестовое письмо.
- `MessageTesterTest`
  - `shouldMarkFirstUnreadAsRead()`: Помечает письмо как прочитанное.
- `OfflineLoginTest`
  - `shouldShowOfflineErrorWhenNetworkDisconnected () `: проверяет поведение при отключении интернета.

### 3. АРХИТЕКТУРА ПРОЕКТА

- ``main/java/org/example`` – содержит классы для работы с элементами интерфейса и Page Objects.

- ``test/java/org/example`` – включает тестовые классы и базовые настройки тестов.

#### 1. Main Module (``main/java/org/example``)

``element`` (Пакет с базовыми элементами интерфейса)

- ``BaseComponent.java`` – абстрактный класс для общих методов взаимодействия с компонентами.

- ``BaseElement.java`` – базовый класс для всех веб-элементов (например, общие методы ``click()``, ``isDisplayed()``).

- ``ButtonElement.java`` – класс для работы с кнопками (наследуется от ``BaseElement``).

- ``ClickElement.java`` – обработка кликабельных элементов (чекбоксы, радиокнопки).

- ``InputElement.java`` – работа с текстовыми полями ввода.

- ``LinkElement.java`` – класс для гиперссылок.

``page`` (Page Objects – модели страниц Gmail) \*\*

Каждый класс соответствует конкретной странице и инкапсулирует её элементы и методы:

- ``BasePage.java`` – родительский класс для всех страниц (общие методы: навигация, ожидания).

- ``LoginPage.java`` – страница авторизации (поля логина/пароля, кнопка входа).

- ``InboxPage.java`` – интерфейс входящих писем (список писем, кнопки управления).

- ``ComposePage.java`` – страница создания нового письма (поля получателя, темы, тела письма).

- `ContactsPage.java` – управление контактами (добавление, редактирование).

- `FiltersPage.java` – настройка фильтров для писем.

- `SettingsPage.java` – общие настройки аккаунта.

- `AdvancedSearchPage.java` – расширенный поиск писем.

- `TasksPage.java` – работа с задачами.

- `ProfileMenuPage.java` – меню профиля (смена имени, выход из аккаунта).

- `OfflinePage.java` – поведение системы при отсутствии интернета.

- `MyAccountPage.java` – управление аккаунтом Google.

## 2. Test Module (`test/java/org/example`)

base` (Базовые настройки тестов)

- `BaseTest.java` – родительский класс для всех тестов:

- Инициализация драйвера (Selenium WebDriver).

- Настройка окружения (браузер, URL).

- Общие методы (`setUp()`, `tearDown()`).

`tests` (Тестовые классы)

Каждый класс тестирует функциональность соответствующей страницы:

- `LoginTest.java` – проверка авторизации.

- `ContactsTest.java` – тесты работы с контактами.

- `FiltersTest.java` – создание и применение фильтров.

- `DeleteMessageTest.java` – удаление и восстановление писем.

- `MessageSenderTest.java` – отправка писем (включая ошибки для невалидных адресов).

- `ProfileNameTest.java` – смена имени пользователя.

- `ChangedLanguageTest.java` – проверка смены языка интерфейса.

- `AdvancedSearchTest.java` – работа расширенного поиска.

- `TasksTest.java` – создание и управление задачами.

- `OfflineLoginTest.java` – поведение системы при разрыве соединения.

## 4. ТЕСТИРОВАНИЕ

### 4.1 Тестирование авторизации

Page Object для авторизации в Gmail. Управляет элементами страницы входа.

Основные методы:

1. ``enterEmail()'` → Вводит логин (Рисунок 2 - Ввод пароля)
2. ``clickEmailNext()'` →
  - Переходит к паролю
3. ``enterPassword()'` → Вводит пароль (Рисунок 2 - Ввод пароля)
4. ``clickPasswordNext()'` →
  - Завершает авторизацию
  - Ожидает загрузки почтового ящика (Рисунок 3 - Главная страница)

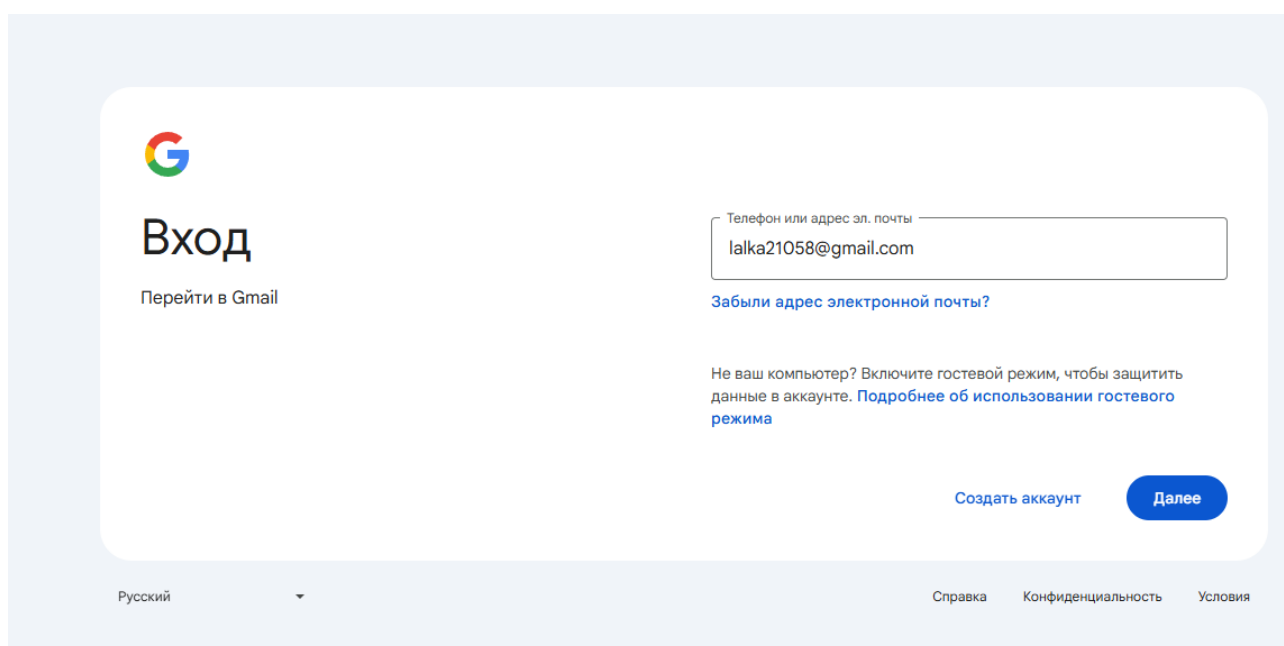


Рисунок 1- Ввод логина

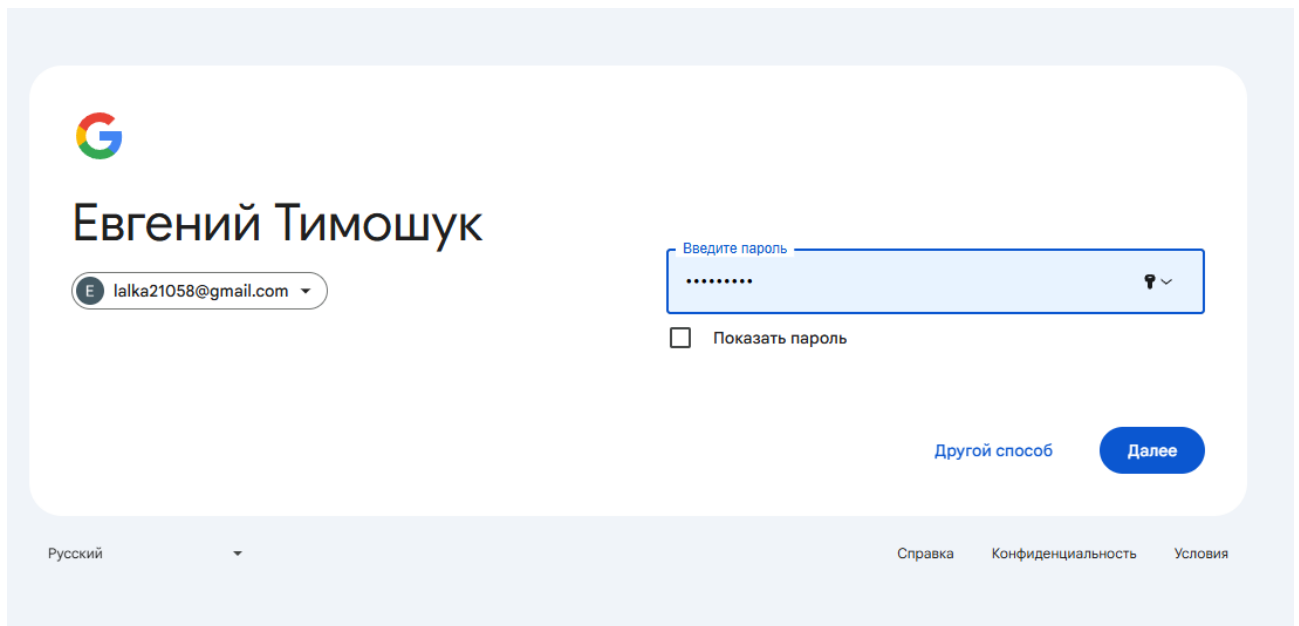


Рисунок 2 - Ввод пароля

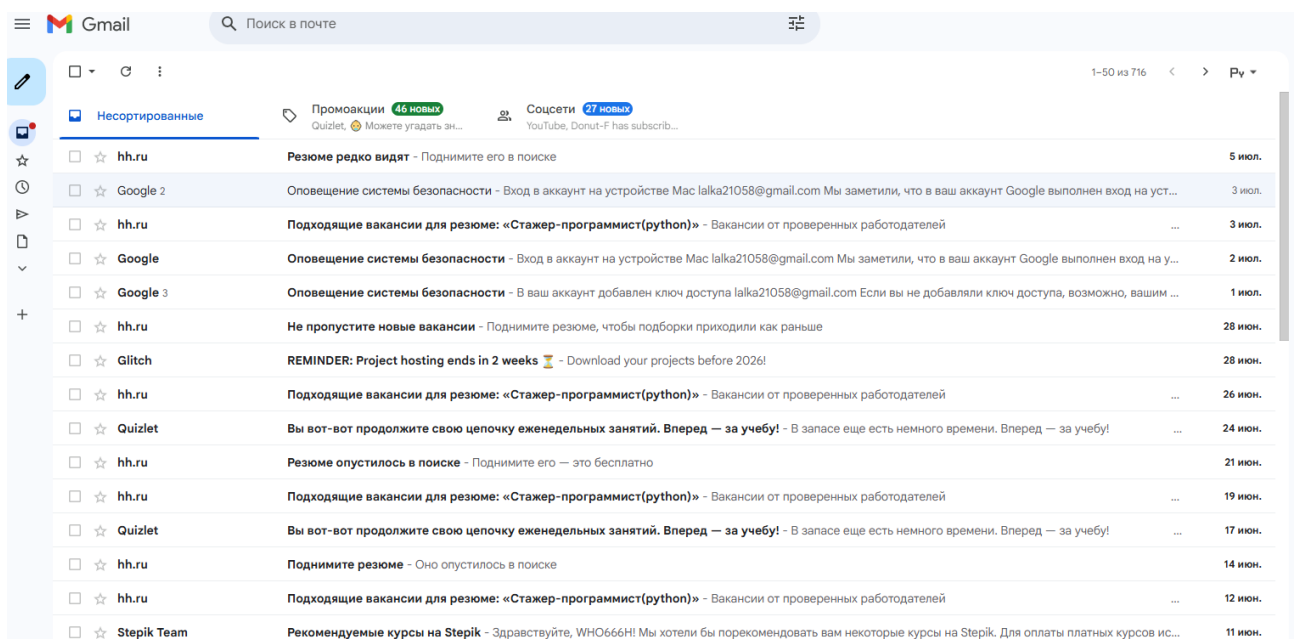


Рисунок 3 - Главная страница

## 4.2 Тестирование всех тестов

### Настройки

Общие Ярлыки Папка "Входящие" Аккаунты и импорт **Фильтры и заблокированные адреса**

Следующие фильтры применяются ко всем входящим письмам:

- ☐ Соответствует: **from:(pelmeshka@gmail.com)**  
Действия: Никогда не отправлять в спам

Выбрать: [Все](#), [Ни одного](#)

Экспорт

Удалить

Рисунок 4- Создание фильтра



← **Имя**

Имя будет изменено во всех сервисах, где используется аккаунт Google. Указанное ранее имя может по-прежнему быть доступно для поиска или показываться в старых сообщениях. [Подробнее...](#)

Имя

Фамилия

**Кто может видеть ваше имя**

 Эту информацию смогут увидеть любые пользователи, которые будут общаться с вами или просматривать созданный вами контент в сервисах Google. [Подробнее...](#) 

Отмена **Сохранить**

Рисунок 5 - Форма по заполнение ФИО

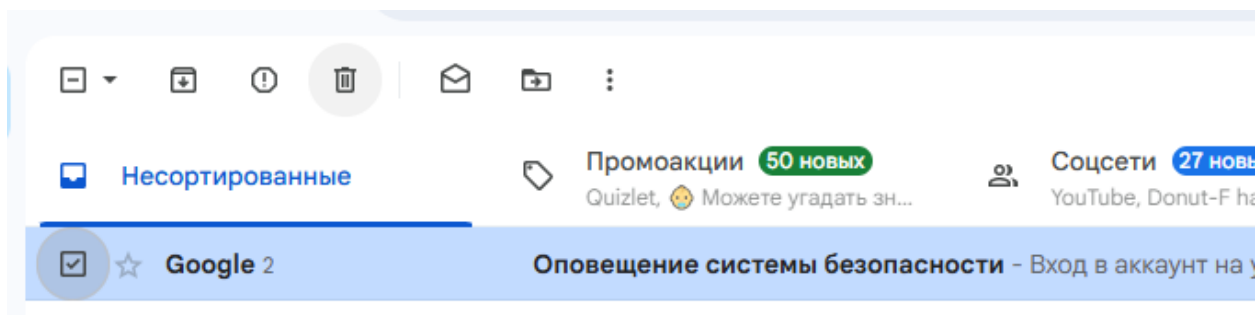


Рисунок 6 - Удаление письма

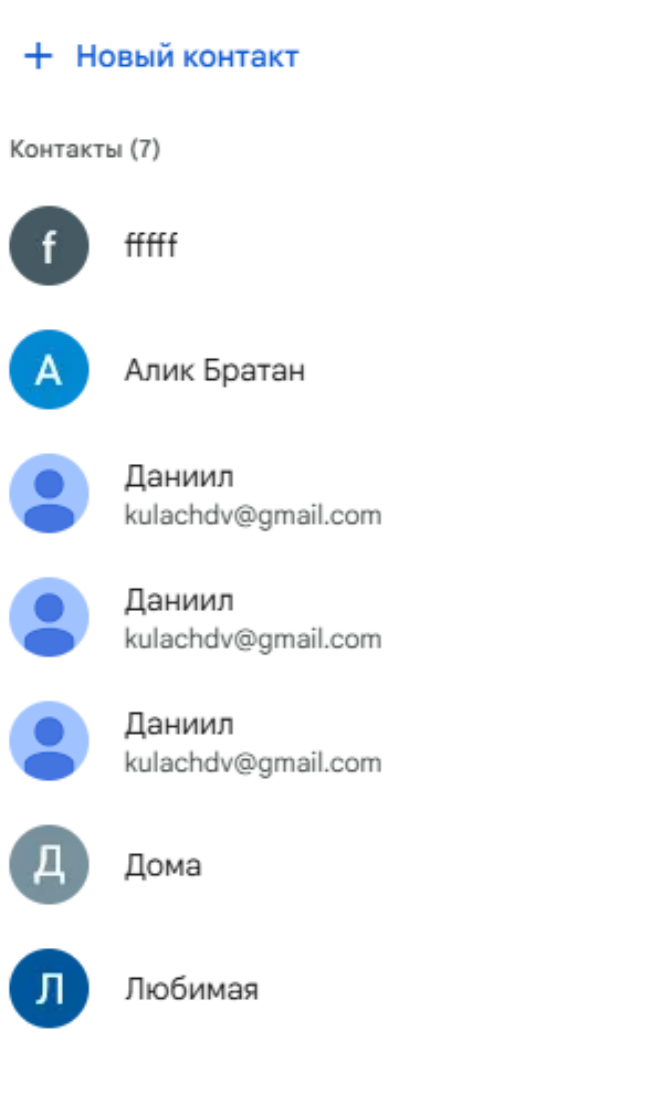



Рисунок 7- Добавление Контакта

## ← Language

Your preferred language for Google services and other languages that you might understand. [Learn more](#)



Changes to your preferred language are reflected on the web. Google might use your language info to show you more relevant content on apps and services. To change the preferred language for mobile apps, go to the language settings on your device.

**Preferred language**

English  
Zimbabwe 

---

**Other languages**

Русский (Russian)  
 Added for you 

[Save](#)

[+ Add another language](#)

Рисунок 8 - Смена языка

### Новое сообщение

asdadadadadadd@gmail.com

Тема

Рисунок 9 - Отправка письма



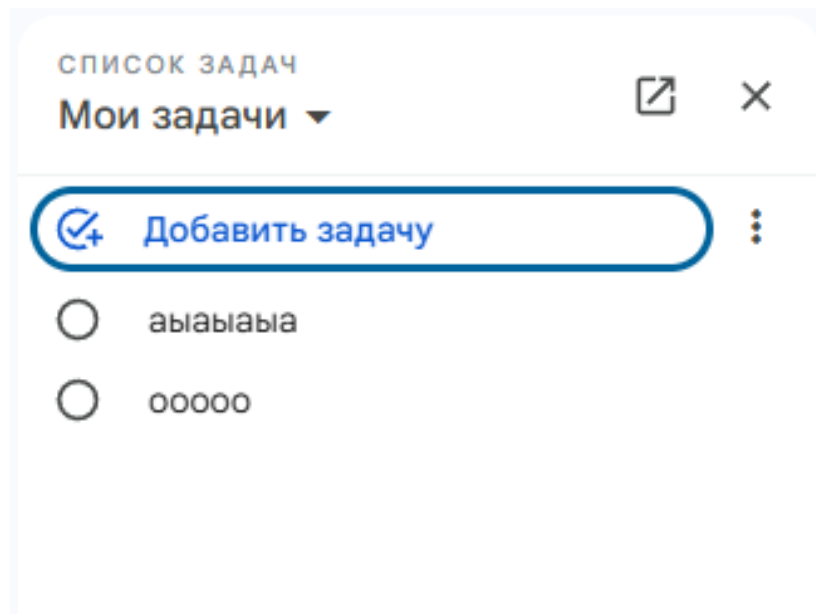


Рисунок 10 - Добавление задачи

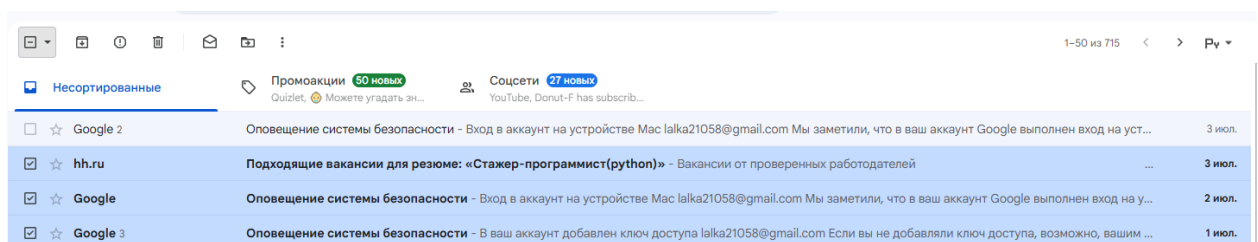


Рисунок 11 - Письма с отметкой

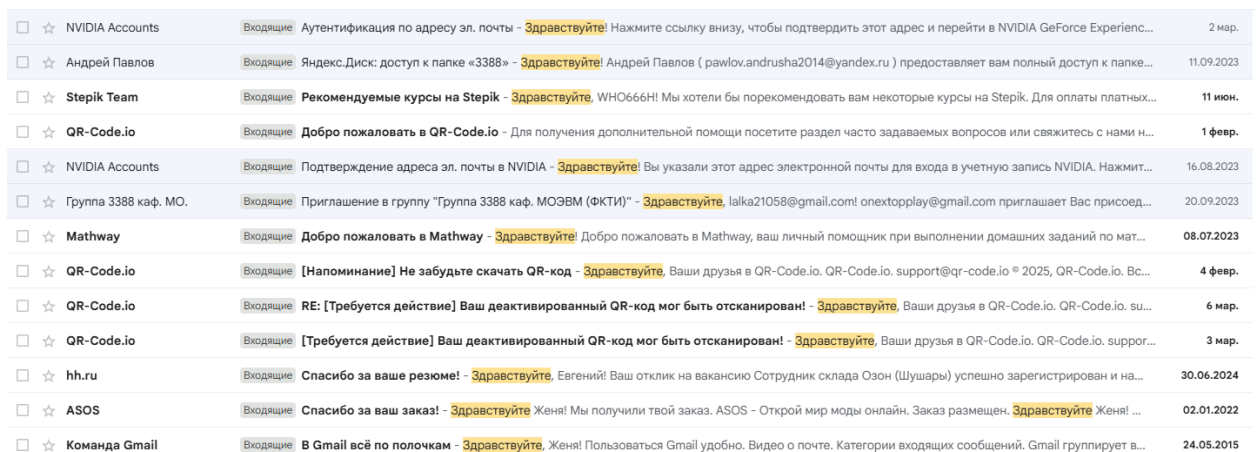


Рисунок 12 - Письма с пометкой "Здравствуйте"

## UML – диаграмма

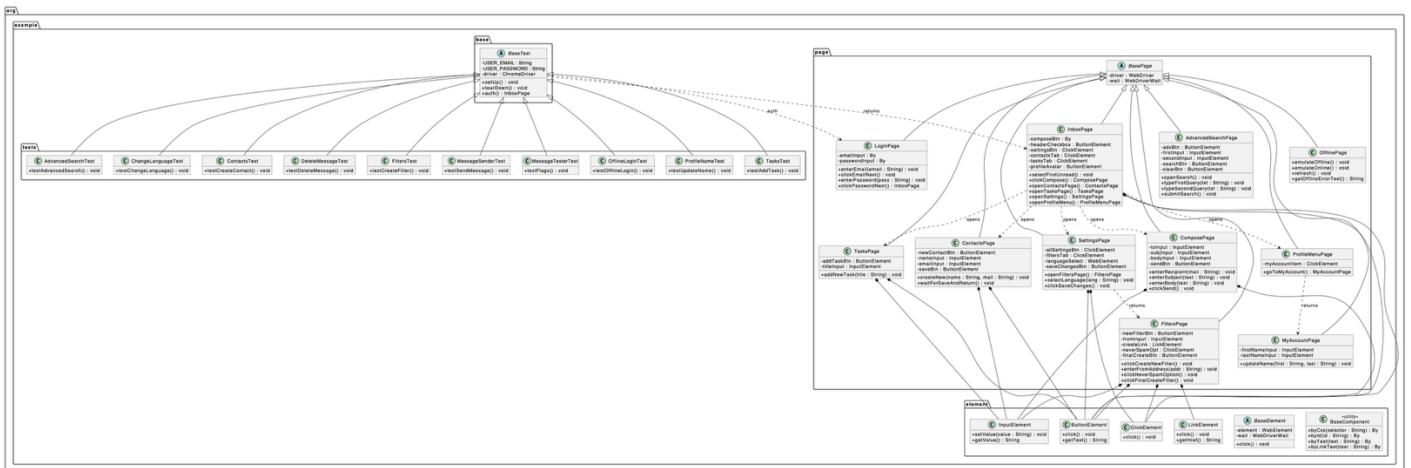


Рисунок 13 - UML

## ЗАКЛЮЧЕНИЕ

Над проектом автоматизации тестирования Gmail мы работали вдвоем, организовав процесс через GitHub с четким разделением обязанностей. Я отвечал за модуль авторизации и функционал работы с письмами, включая удаление сообщений и пометку прочитанных. Второй участник взял на себя реализацию тестов для настроек — фильтров, смены языка интерфейса и расширенного поиска. Третий участник разрабатывал тесты для работы с контактами, задачами, offline-режимом и отправкой писем.

Для координации мы создали общий репозиторий с единой структурой проекта, где сразу установили ключевые соглашения: именование методов в camelCase, обязательное использование Page Object Model и вынесение стандартных локаторов в BaseComponent. Каждый день мы проводили короткие 15-минутные созвоны для синхронизации прогресса, а для управления кодом использовали Git Flow с обязательным ревью перед мерджем в основную ветку.

В процессе столкнулись с типичными проблемами командной разработки. Конфликты версий решали через систему feature-веток и pull-request'ов. Чтобы предотвратить расхождения в стиле кодирования, внедрили Checkstyle с общим конфигурационным файлом. При обнаружении дублирующихся методов сразу выносили их в BasePage, а проблемы с зависимостями фиксировали через жесткое закрепление версий в pom.xml.

## СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Apache Maven. Maven Guides [Электронный ресурс] // Apache Software Foundation. – URL: <https://maven.apache.org/guides/index.html> (дата обращения: 05.07.2025).
2. Google. Gmail [Электронный ресурс] // Google. – URL: <https://mail.google.com/> (дата обращения: 05.07.2025).
3. Oracle. Java Documentation [Электронный ресурс] // Oracle. – URL: <https://docs.oracle.com/en/java/> (дата обращения: 05.07.2025).
4. Potapov, R. Practika [Электронный ресурс] // GitHub. – URL: <https://github.com/rodionpotapov/Practika/tree/main> (дата обращения: 05.07.2025).
5. Selenide. Official Documentation [Электронный ресурс] // Selenide. – URL: <https://selenide.org/documentation.html> (дата обращения: 05.07.2025).