МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ФГБОУ ВПО

Новосибирский Государственный Технический Университет

Институт Социальных Технологий

ОТЧЕТ

по дисциплине «**Технология разработки программного обеспечения**»

по теме «**Разработка ИС технической поддержки предприятия. Разработка подсистемы материально ответственного**»

Выполнил:

Студент группы И-03

Пакшин А. А.

Проверил:

Преподаватель

Бертик А.А

Новосибирск 2023

**Оглавление**

[Введение 3](#_Toc153109311)

[Анализ предметной области 3](#_Toc153109312)

[**Обзор существующих аналогов** 5](#_Toc153109313)

[Средства разработки 5](#_Toc153109314)

[Реализация проекта 6](#_Toc153109315)

[Заключение 9](#_Toc153109316)

[Список источников 9](#_Toc153109317)

[Приложение 9](#_Toc153109318)

# **Введение**

Техническая поддержка включает в себя различные услуги предприятия по оказанию помощи своим клиентам, то есть пользователям услуг или продуктов данного предприятия. Техническая поддержка помогает клиенту выяснять и решать проблемы, связанные с продуктом и его использованием, но не обучает использовать или индивидуально настраивать конкретный продукт.

Техническая поддержка предоставляется, как правило, по телефону, через интернет, по электронной почте, через онлайн-сервисы поддержки на веб-сайте или через инструменты, посредством которых пользователи могут зарегистрировать заявку и затем проследить историю состояний/этапов её решения. Большие организации часто имеют собственную, внутреннюю службу технической поддержки, которая помогает персоналу решать возникающие проблемы, носящие технический характер.

В некоторых случаях осуществление работником своих трудовых обязанностей предполагает распоряжение материальными ценностями, принадлежащими работодателю. Например, специалисту с разъездным характером деятельности для работы может потребоваться ноутбук, дорогостоящий набор инструментов или даже автомобиль. А кассир или работник банка в ходе выполнения своих повседневных обязанностей может иметь дело с крупными суммами денег.

Материально ответственное лицо — это сотрудник, который непосредственно обслуживает денежные, товарные ценности или иное имущество, вверенные ему на основании специального письменного договора или полученные им по разовому документу, за которые он обязан возмещать причиненный работодателю прямой действительный ущерб в полном размере.

# **Анализ предметной области**

Рынок порталов технической поддержки в различных отраслях и на различных сайтах является довольно разнообразным и динамичным. Анализ данного рынка включает оценку его размера, трендов, ключевых игроков и особенностей в каждой отрасли.

Размер рынка: Рынок порталов технической поддержки охватывает все отрасли, где присутствует высокий спрос на техническую поддержку. Это может быть информационные технологии, электроника, инженерные услуги, автомобильная индустрия, медицинское оборудование и многие другие. Общий объем рынка оценивается в миллиарды долларов, и он продолжает расти с увеличением числа потребителей и использования технологий.

Тренды: в последние годы наблюдается рост спроса на онлайн-порталы технической поддержки из-за увеличения числа пользователей интернета и развития онлайн-бизнеса. Компании все больше отдает предпочтение онлайн-сервисам, таким как онлайн-чаты, форумы поддержки и базы знаний, где пользователи могут найти ответы на свои вопросы без необходимости обращаться в службу поддержки по телефону или лично. Также наблюдается рост использования технологий искусственного интеллекта (ИИ) и чат-ботов для автоматизации поддержки и улучшения пользовательского опыта.

Ключевые игроки: на рынке порталов технической поддержки существует множество компаний, предлагающих свои решения. Крупные игроки в этой области включают Salesforce, Zendesk, Freshdesk, ServiceNow и JIRA. Каждая компания имеет свои особенности и целевую аудиторию, а также специализируется на конкретных отраслях или предлагает решения для различных типов клиентов.

Особенности в отраслях: Различные отрасли могут иметь уникальные потребности в технической поддержке и различные требования к порталам. Например, в сфере информационных технологий важными функциями могут быть создание тикетов для отслеживания запросов пользователей и помощь в решении сложных технических проблем. В инженерной отрасли эффективное управление проектами и обращениями с клиентами могут иметь первостепенное значение. Компании, специализирующиеся на поддержке медицинского оборудования, могут предлагать необходимые сертификации и обновленную информацию о новых системах и доступных ремонтных услугах.

## **Обзор существующих аналогов**

ZenDesk. Англоязычный хелпдеск предлагает быструю обработку заявок из разных источников (соцсети, мессенджеры, почта, телефония, лайв-чаты), создание собственной базы знаний, портала клиентского обслуживания и форумов для комьюнити. Через Zendesk можно принимать входящие звонки, анализировать KPI, формировать прейскуранты, задавать триггерные действия для рутинных задач.

Intraservice. Основной рабочий инструмент сервиса — список заявок. Он формируется через email и веб-форму на сайте, поддерживает групповые операции над задачами, позволяет сортировать запросы с помощью фильтров. Менеджер может создавать заявки от лица другого пользователя, назначать исполнителей и наблюдателей, прикреплять любые файлы и активы. Проанализировать эффективность работы техподдержки помогут отчёты по времени обработки заявок, изменениям статусов, трудозатратам и любым другим выбранным KPI.

FreeScout. Это система поддержки с открытым кодом: любой желающий может увидеть исходный код и проверить его на безопасность. Присутствует возможность отвечать на тикеты из почты, база знаний, клиентский портал самообслуживания, модуль CRM для управления информацией о клиентах, шаблоны ответов, интеграция с WhatsApp, Twitter, Telegram, API (интерфейс прикладного программирования для передачи данных), Jira, плагином WordPress Easy Digital Downloads.

# **Средства разработки**

HTML - стандартизированный язык гипертекстовой разметки документов для просмотра веб-страниц в браузере. Веб-браузеры получают HTML документ от сервера по протоколам HTTP/HTTPS или открывают с локального диска, далее интерпретируют код в интерфейс, который будет отображаться на экране монитора. Элементы HTML являются строительными блоками HTML страниц. С помощью HTML разные конструкции, изображения и другие объекты, такие как интерактивная веб-форма, могут быть встроены в отображаемую страницу. HTML предоставляет средства для создания заголовков, абзацев, списков, ссылок, цитат и других элементов. Элементы HTML выделяются тегами, записанными с использованием угловых скобок. Такие теги, как <img /> и <input />, напрямую вводят контент на страницу. Другие теги, такие как <p>, окружают и оформляют текст внутри себя и могут включать другие теги в качестве подэлементов. Браузеры не отображают HTML-теги, но используют их для интерпретации содержимого страницы.

Notepad++ — это свободный текстовый редактор с открытым исходным кодом для Windows с подсветкой синтаксиса, разметки, а также языков описания аппаратуры.

PhpMyAdmin — веб-приложение с открытым кодом, написанное на языке PHP и представляющее собой веб-интерфейс для администрирования СУБД MySQL. PhpMyAdmin позволяет через браузер и не только осуществлять администрирование сервера MySQL, запускать команды SQL и просматривать содержимое таблиц и баз данных. Приложение пользуется большой популярностью у веб-разработчиков, так как позволяет управлять СУБД MySQL без непосредственного ввода SQL команд.

Flask — фреймворк для создания веб-приложений на языке программирования Python, использующий набор инструментов Werkzeug, а также шаблонизатор Jinja2. Относится к категории так называемых микрофреймворков — минималистичных каркасов веб-приложений, сознательно предоставляющих лишь самые базовые возможности.

# **Реализация проекта**

Передо мной стояла задача тестирования всей системы на работоспособность. Тестирование производилось вручную или с помощью python и библиотеки «pytest».

Тестирование базы данных. Тестирование базы данных заключалось в проверке правильности таблиц, внешних ключей, уникальных ключей, тестирование подключения базы данных к системе технической поддержки и тестирование SQL запросов. Проверка внешних ключей и правильность столбцов в таблицах проводилась вручную, а подключение к базе данных и запросы тестировались автоматический с помощью python скриптов.

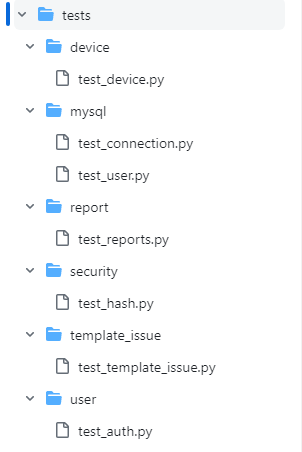


Рис. 1 «Схема файлов»

Файл tests/mysql/test\_connection.py

**from** **src.core.classes.mysql** **import** MySqlBase

**def** **test\_connection**():

db\_connection = MySqlBase().connection()

db\_connection.open()

**try**:

cursor = db\_connection.connection.cursor()

cursor.execute('SELECT 1;')

rows = cursor.fetchall()

**for** row **in** rows:

**assert** row[**0**] == **1**, row[**0**]

**finally**:

db\_connection.close()

Файл tests/mysql/test\_user.py

**import** **time**

**from** **http.client** **import** OK

**from** **src.core.classes.models.user** **import** ModelUser

**from** **src.core.classes.mysql** **import** MySqlBase

**def** **get\_rule\_by\_value**(value: str):

**“””**

**Получение id роли по его названию.**

**“””**

db\_connection = MySqlBase().connection()

db\_connection.open()

**try**:

cursor = db\_connection.connection.cursor()

# Используем параметризованный запрос

cursor.execute("SELECT id FROM rules WHERE name=%s", (value,))

row = cursor.fetchone()

# Проверяем, есть ли результат

**if** row:

rule\_id = int(row[**0**])

**return** rule\_id

**else**:

# Если не найдено, можно вернуть, например, None

**return** **1**

**finally**:

db\_connection.close()

**def** **test\_create\_user**():

MySQl = MySqlBase()

MySQlQueries = MySQl.queries()

db\_connection = MySQl.connection()

db\_connection.open()

**try**:

model = ModelUser()

model.username = 'user-\_\_123'

model.password = 'UserPassword123#'

model.created\_at = int(time.time())

model.rule = get\_rule\_by\_value('user')

model.admin = False

# Import Authentication inside the function to break the circular dependency

**from** **src.core.database.authentication** **import** Authentication

query: MySQlQueries = Authentication(db\_connection).sign\_up(model)

query.execute()

**assert** query.status\_code == OK

**finally**:

db\_connection.close()

Тестирование системы технической поддержки. Тестирование системы тп. заключалось в проверке различных модулей программы, а это модуль безопасности, который отвечает за хеширование паролей, модуль пользователя, который отвечает за проверку на то, что только пользователь может отправлять запросы об ошибках и получать историю своих запросов, а также проверка на то, что он пользователь без привилегии Administrator, не имеет доступ, к странице администратора.

Файл tests/user/test\_auth.py

**from** **http.client** **import** BAD\_REQUEST, CONFLICT, OK

**from** **src.core.classes.user** **import** User

USERNAME = 'user23\_---'

PASSWORD = ')O#$VJ)#@MCO#NJKV43nkjc32v'

EXPIRATION = **600**

**class** **SignUp**:

**“””**

**Класс регистрации, в методе exec(), выполняется функция регистрации, различными тестовыми данными, если ok – регистрация прошла успешно, bad\_request – плохой запрос, conflict – пользователь уже существует.**

**”””**

**def** **\_\_init\_\_**(self) -> None:

self.\_\_ok: int

self.\_\_bad\_request: int

self.\_\_conflict: int

**@property**

**def** **ok**(self) -> int:

**return** self.\_\_ok

**@property**

**def** **bad\_request**(self) -> int:

**return** self.\_\_bad\_request

**@property**

**def** **conflict**(self) -> int:

**return** self.\_\_conflict

**def** **exec**(self):

user\_obj = User().sign\_up()

user\_obj.execute(USERNAME, PASSWORD)

self.\_\_ok = user\_obj.status\_code

user\_obj.execute('1', '43432v42c3')

self.\_\_bad\_request = user\_obj.status\_code

user\_obj.execute(USERNAME, PASSWORD)

self.\_\_conflict = user\_obj.status\_code

**class** **SignIn**:

**“””**

**Класс входа в учетную запись, ok – если вход успешный, token – елси вход успешный, то возвращается токен, bad\_request – плохой запрос.**

**”””**

**def** **\_\_init\_\_**(self) -> None:

self.\_\_ok: int

self.\_\_bad\_request: int

self.\_\_token: str

**@property**

**def** **ok**(self) -> int:

**return** self.\_\_ok

**@property**

**def** **token**(self) -> str:

**return** self.\_\_token

**@property**

**def** **bad\_request**(self) -> int:

**return** self.\_\_bad\_request

**def** **exec**(self):

user\_obj = User().sign\_in()

user\_obj.execute(USERNAME, PASSWORD, EXPIRATION)

self.\_\_ok = user\_obj.status\_code

self.\_\_token = user\_obj.get\_token

user\_obj.execute('1', '43432v42c3', EXPIRATION)

self.\_\_bad\_request = user\_obj.status\_code

**class** **SignOut**:

**“””**

**Класс выхода из учетной записи, ok – удачный выход, bad\_request – плохой запрос.**

**”””**

**def** **\_\_init\_\_**(self, token: str) -> None:

self.\_\_ok = **0** # Set an initial value

self.\_\_bad\_request = **0** # Set an initial value

self.\_\_token = token

**@property**

**def** **ok**(self) -> int:

**return** self.\_\_ok

**@property**

**def** **bad\_request**(self) -> int:

**return** self.\_\_bad\_request

**def** **exec**(self):

user\_obj = User().sign\_out()

user\_obj.execute(self.\_\_token)

self.\_\_ok = user\_obj.status\_code

user\_obj.execute('v23c423v3')

self.\_\_bad\_request = user\_obj.status\_code

**def** **test**():

sign\_up = SignUp()

sign\_up.**exec**()

**assert** sign\_up.ok == OK, sign\_up.ok

**assert** sign\_up.bad\_request == BAD\_REQUEST, sign\_up.bad\_request

**assert** sign\_up.conflict == CONFLICT, sign\_up.conflict

sign\_in = SignIn()

sign\_in.**exec**()

**assert** sign\_in.ok == OK, sign\_in.ok

**assert** sign\_in.bad\_request == BAD\_REQUEST, sign\_in.bad\_request

sign\_out = SignOut(sign\_in.token)

sign\_out.**exec**()

**assert** sign\_out.ok == OK, sign\_in.token

**assert** sign\_out.bad\_request == BAD\_REQUEST, sign\_out.bad\_request

Файл tests/report/test\_reports.py

**import** **datetime**

**from** **http.client** **import** BAD\_REQUEST, CONFLICT, OK, UNAUTHORIZED

**from** **src.core.classes.mysql** **import** MySqlBase

**from** **src.core.classes.report** **import** Report

**from** **src.core.classes.user** **import** User

USERNAME = 'user44\_---'

PASSWORD = ')O#$VJ)#@MCO#N12V43nkjc32v'

EXPIRATION = **600**

**class** **SignUp**:

**“””**

**Класс регистрации, в методе exec(), выполняется функция регистрации, различными тестовыми данными, если ok – регистрация прошла успешно, bad\_request – плохой запрос, conflict – пользователь уже существует.**

**”””**

**def** **\_\_init\_\_**(self) -> None:

self.\_\_ok: int

self.\_\_bad\_request: int

self.\_\_conflict: int

**@property**

**def** **ok**(self) -> int:

**return** self.\_\_ok

**@property**

**def** **bad\_request**(self) -> int:

**return** self.\_\_bad\_request

**@property**

**def** **conflict**(self) -> int:

**return** self.\_\_conflict

**def** **exec**(self):

user\_obj = User().sign\_up()

user\_obj.execute(USERNAME, PASSWORD)

self.\_\_ok = user\_obj.status\_code

user\_obj.execute('1', '43432v42c3')

self.\_\_bad\_request = user\_obj.status\_code

user\_obj.execute(USERNAME, PASSWORD)

self.\_\_conflict = user\_obj.status\_code

**class** **SignIn**:

**“””**

**Класс входа в учетную запись, ok – если вход успешный, token – елси вход успешный, то возвращается токен, bad\_request – плохой запрос.**

**”””**

**def** **\_\_init\_\_**(self) -> None:

self.\_\_ok: int

self.\_\_bad\_request: int

self.\_\_token: str

**@property**

**def** **ok**(self) -> int:

**return** self.\_\_ok

**@property**

**def** **token**(self) -> str:

**return** self.\_\_token

**@property**

**def** **bad\_request**(self) -> int:

**return** self.\_\_bad\_request

**def** **exec**(self):

user\_obj = User().sign\_in()

user\_obj.execute(USERNAME, PASSWORD, EXPIRATION)

self.\_\_ok = user\_obj.status\_code

self.\_\_token = user\_obj.get\_token

user\_obj.execute('1', '43432v42c3', EXPIRATION)

self.\_\_bad\_request = user\_obj.status\_code

**class** **Create**:

**“””**

**Класс отправки жалобы в тех. поддержку, unauthorized – пользователь не авторизирован, bad\_request – плохой запрос, ok – жалоба отправлена.**

**”””**

**def** **\_\_init\_\_**(self, token: str) -> None:

self.\_\_ok = **0** # Set an initial value

self.\_\_bad\_request = **0** # Set an initial value

self.\_\_token = token

self.\_\_unauthorized = **0**

**@property**

**def** **ok**(self) -> int:

**return** self.\_\_ok

**@property**

**def** **unauthorized**(self) -> int:

**return** self.\_\_unauthorized

**@property**

**def** **bad\_request**(self) -> int:

**return** self.\_\_bad\_request

**def** **exec**(self, inventory\_number: str):

user\_obj = Report().create()

user\_obj.execute(self.\_\_token, 'Issue', 'Issue Text', inventory\_number)

self.\_\_ok = user\_obj.status\_code

user\_obj.execute('v23c423v3', '4342c342vv2343c2v4c424vc32v24c2343vc324c234vv', 'BMuLczvcFvASPN8ryybzKVkOut03PvruC', **1**)

self.\_\_bad\_request = user\_obj.status\_code

user\_obj.execute(

'GnpQIcODM9hyQqM1dwNs1uxFnUr7VCJF445lbaDZlHmQVf7TRpYCfeS1IhTIM0Mel9OiS7XJJWDi15edYbL2Laf1Y5Tt0bMKm2oJ6me4NxtWgFmK81bKNFRC8v1aKOoGBB9lb7ZmsZ16iN6halzuz1mIAYQtBF07qgEkRlEg7TKFc9qNSQvHWPIV7ak0DVbrBVO7aHOkJnA50TGLjfTjkqoS54UI87j9Eiofe3n9cIekWdsIDVP90Dq0hXlOMXKG',

'4342c342vv2343c2v4c424vc32v24c2343vc324c234vv', 'BMuLczvcFvASPN8ryybzKVkOut03PvruC',

inventory\_number

)

self.\_\_unauthorized = user\_obj.status\_code

**def** **get\_device\_id\_by\_inventory\_number**(value: str):

"""Temp solution"""

db\_connection = MySqlBase().connection()

db\_connection.open()

**try**:

cursor = db\_connection.connection.cursor()

# Используем параметризованный запрос

cursor.execute("SELECT id FROM devices WHERE inventory\_number=%s", (value,))

row = cursor.fetchone()

# Проверяем, есть ли результат

**if** row:

rule\_id = int(row[**0**])

**return** rule\_id

**else**:

# Если не найдено, можно вернуть, например, None

**return** **1**

**finally**:

db\_connection.close()

**def** **create\_device**(inv\_num: str):

"""Temp solution"""

db\_connection = MySqlBase().connection()

db\_connection.open()

**try**:

cursor = db\_connection.connection.cursor(prepared=True, )

insert\_query = """

INSERT INTO devices VALUES (NULL, %s, %s, %s, %s, %s, %s, %s);

"""

params = (

inv\_num,

'Компьютер',

datetime.datetime(**2006**, **6**, **5**).isoformat(),

**3**,

'321',

'Персональный компьютер',

'testUser'

)

cursor.execute(insert\_query, params)

db\_connection.connection.commit()

**finally**:

db\_connection.close()

**def** **test**():

sign\_up = SignUp()

sign\_up.**exec**()

**assert** sign\_up.ok == OK, sign\_up.ok

**assert** sign\_up.bad\_request == BAD\_REQUEST, sign\_up.bad\_request

**assert** sign\_up.conflict == CONFLICT, sign\_up.conflict

sign\_in = SignIn()

sign\_in.**exec**()

**assert** sign\_in.ok == OK, sign\_in.ok

**assert** sign\_in.bad\_request == BAD\_REQUEST, sign\_in.bad\_request

inventory\_number = '4389CNV23CM8C'

create\_device(inventory\_number)

report\_create = Create(sign\_in.token)

report\_create.**exec**(inventory\_number)

**assert** report\_create.ok == OK, report\_create.ok

**assert** report\_create.bad\_request == BAD\_REQUEST, report\_create.bad\_request

**assert** report\_create.unauthorized == UNAUTHORIZED, report\_create.unauthorized

Автоматизация тестирования. Так как ПО, которое пишется в основном для Linux, пишут люди на Windows, у них могут не оказаться, те программы или библиотеки, которые используются на Linux и поэтому разработчики придумали, такую вещь, как Github/Gitlab Runner, который позволяет выполнять различные действия, во временно созданной изолированной среде, каждый раз, когда изменяется код в репозитории Github/Gitlab. Был создан ci-cd.yml файл, который сам устанавливал python, настраивал его окружение, скачивал библиотеки, запускал базу данных MySql, запускал тесты, затем если тесты прошли успешно, то создается Docker образ, который после отправляется в Github Container registry.

Файл ci-cd.yml

name: CI-CD

on: [push]

jobs:

tests:

name: Tests

runs-on: ubuntu-latest

strategy:

matrix:

python-version: ["3.11"]

steps:

- uses: actions/checkout@v4

- name: Setup Python # Set Python version

uses: actions/setup-python@v4

with:

python-version: ${{ matrix.python-version }}

- name: Update

run: sudo apt-get update

- name: Run MySql Server

run: make docker-deploy

- name: Init venv and requirements

run: make init

- name: Run Tests

run: make test

deploy:

name: Deploy # Сборка образа и отправка на Github Registry, если прошли тесты

needs: tests

runs-on: ubuntu-latest

if: ${{ needs.tests.result == 'success' }}

steps:

- name: Checkout code

uses: actions/checkout@v4

- name: Set up server in release mode

run: |

**python3 -c "import json; data = json.load(open('src/config.json')); data['server\_debug'] = False; json.dump(data, open('src/config.json', 'w'), indent=2)"**

- name: Log in to the Container registry

uses: docker/login-action@65b78e6e13532edd9afa3aa52ac7964289d1a9c1

with:

registry: ghcr.io

username: ${{ github.actor }}

password: ${{ secrets.ACTIONS }}

- name: Extract metadata (tags, labels) for Docker

id: meta

uses: docker/metadata-action@9ec57ed1fcdbf14dcef7dfbe97b2010124a938b7

with:

images: parinovyt/technicalsupportcenterproject

- name: Build and push Docker image

uses: docker/build-push-action@f2a1d5e99d037542a71f64918e516c093c6f3fc4

with:

context: .

push: true

tags: ghcr.io/parinovyt/technicalsupportcenterproject:latest

labels: ${{ steps.meta.outputs.labels }}

mirror-to-gitlab:

name: Mirror to GitLab # Отправка кода на Gitlab, если прошли тесты

needs: tests

runs-on: ubuntu-latest

if: ${{ needs.tests.result == 'success' }}

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Updating

run: |

**sudo apt-get update**

- name: Mirror to GitLab

run: |

**git remote add gitlab ${{secrets.GITLABURL}}**

**git fetch --unshallow**

**git push --quiet gitlab --mirror || true**

**Итог**

В ходе тестирования, были протестированы: база данных, система технической поддержки, также была создана возможность, тестирования ПО, если у разработчика нету, требуемого ПО или библиотек.

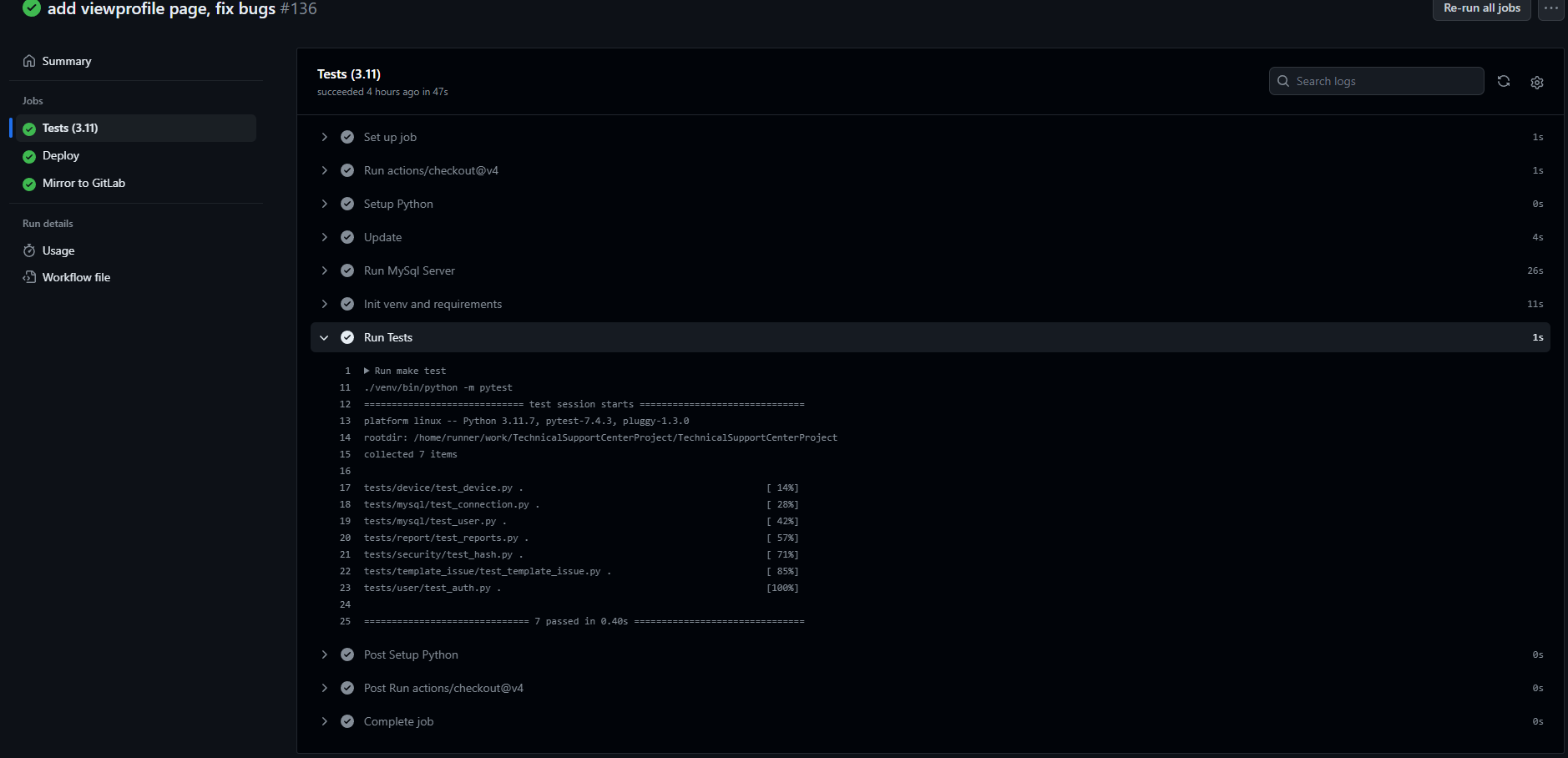


Рис. 2 «Скриншот тестов с помощью Github Actions»

**Источники**

«pytest» / <https://docs.pytest.org/en/7.4.x/>

«Github Actions» / <https://docs.github.com/ru/actions>