ПРИЛОЖЕНИЕ 2

Программный модуль для автоматизированного тестирования веб-форм

(шифр ПМ АТВ)

Текст программы

Исполнитель:

студент гр. ПИН-51Д _____/ Джугели Д.А./

Руководитель:

к.п.н., доц. _____/ Федотова Е.Л./

Москва 2025

АННОТАЦИЯ

В данном программном документе представлены фрагменты текста программного модуля для автоматизированного тестирования веб-форм (ПМ АТВ), предназначенного для генерации тестовых данных, управления временными email-адресами, автозаполнения веб-форм и обработки входящих писем через SMTP-сервер. Текст программы реализован на языках JavaScript (клиентская часть) и Python (серверная часть).

Основные функции ПМ АТВ:

- Генерация тестовых данных для заполнения веб-форм;
- Управление временными email-адресами для приема сообщений;
- Автоматическое заполнение веб-форм через браузерное расширение;
- Обработка входящих писем через SMTP-сервер;
- Уведомления в реальном времени через WebSocket.

Перечень ключевых функций, реализуемых ПМ АТВ, подробно описан в пояснительной записке. В данном приложении представлены наиболее важные из них:

- Генерация тестовых данных;
- Управление временными email-адресами;
- Автозаполнение веб-форм;
- Обработка входящих писем;
- Установка и поддержание WebSocket-соединения.

СОДЕРЖАНИЕ

1. СТРУКТУРА ПРОГРАММНОГО КОДА

Проект ПМ АТВ состоит из двух основных частей: клиентской (браузерное расширение) и серверной. Клиентская часть реализована на JavaScript, а серверная на Python с использованием FastAPI и Uvicorn.

1.1 Основные файлы клиентской части

Клиентская часть реализована в виде браузерного расширения и включает следующие ключевые файлы:

- manifest.json – манифест расширения, содержащий метаданные и разрешения;
- popup.js – скрипт, отвечающий за генерацию тестовых данных и временных email-адресов;
- content.js – скрипт для автозаполнения веб-форм;
- sidepanel.js – скрипт для отображения входящих писем в Side Panel;
- data-generator.js – модуль для генерации тестовых данных с исопльзованием библиотеки Faker.js;
- background.js – фоновый скрипт для обработки событий и связи с сервером

1.2 Основные файлы серверной части

Серверная часть реализована на Python с использованием фреймворка FastAPI и включает следующие ключевые файлы:

- main.py – основной файл приложения, содержащий конфигурацию и маршрутизацию;
- models.py – модели базы данных;
- shemas.py – схемы данных для валидации;
- email_service.py – модуль генерации временных адресов;
- email_parser.py – модуль для парсинга писем;
- smtp_server.py – реализация SMTP-сервера для приема входящих писем;
- websocket.py – модуль для обработки WebSocket-соединений.

2. ТЕКСТ ПРОГРАММЫ НА ИСХОДНОМ ЯЗЫКЕ

2.1 Клиентская часть

2.1.1 Файл manifest.json

```
{
  "manifest_version": 3,
  "name": "AutoTest Web Forms",
  "version": "1.0",
  "description": "Браузерное расширение для автозаполнения веб-форм и
генерации временных email-адресов",
  "permissions": [
    "activeTab",
    "storage",
    "webRequest",
    "webRequestBlocking"
  ],
  "action": {
    "default_popup": "popup.html",
    "default_icon": {
      "16": "icons/icon16.png",
      "48": "icons/icon48.png",
      "128": "icons/icon128.png"
    }
  },
  "background": {
    "service_worker": "background.js"
  },
  "content_scripts": [
    {
      "matches": ["<all_urls>"],
      "js": ["content.js"]
    }
  ],
  "side_panel": {
    "default_path": "sidepanel.html"
  }
}
```

2.1.2 Файл popup.js

```
document.addEventListener('DOMContentLoaded', async () => {
  const generateDataBtn = document.getElementById('generateDataBtn');
  const generateEmailBtn =
document.getElementById('generateEmailBtn');
  const dataOutput = document.getElementById('dataOutput');
  const emailOutput = document.getElementById('emailOutput');

  // Генерация тестовых данных
  generateDataBtn.addEventListener('click', () => {
    const fakeData = generateFakeData();
    dataOutput.textContent = JSON.stringify(fakeData, null, 2);
  });

  // Генерация временного email-адреса
```

```
  generateEmailBtn.addEventListener('click', async () => {
    const response = await fetch('http://localhost:8000/api/generate-
email');
    const data = await response.json();
    emailOutput.textContent = data.email;
  });
});

// Функция для генерации тестовых данных
function generateFakeData() {
  return {
    name: faker.name.findName(),
    email: faker.internet.email(),
    phone: faker.phone.phoneNumber(),
    address: faker.address.streetAddress()
  };
}
```

2.1.3 Файл content.js

```
// Автозаполнение веб-форм
document.addEventListener('DOMContentLoaded', () => {
  chrome.runtime.onMessage.addListener((request, sender, sendResponse)
=> {
    if (request.action === 'autofill') {
      const fakeData = generateFakeData();
      const inputs = document.querySelectorAll('input, textarea,
select');

      inputs.forEach(input => {
        const fieldName = input.name.toLowerCase();

        if (fieldName.includes('name')) {
          input.value = fakeData.name;
        } else if (fieldName.includes('email')) {
          input.value = fakeData.email;
        } else if (fieldName.includes('phone')) {
          input.value = fakeData.phone;
        } else if (fieldName.includes('address')) {
          input.value = fakeData.address;
        }
      });
    }
  });
});
```

2.1.4 Файл sidepanel.js

```
// Отображение входящих писем
document.addEventListener('DOMContentLoaded', async () => {
  const emailsList = document.getElementById('emailsList');

  // Подключение к WebSocket для получения уведомл
ений
  const socket = new WebSocket('ws://localhost:8000/ws/emails');

  socket.onmessage = (event) => {
    const email = JSON.parse(event.data);
```

```
    const emailElement = document.createElement('div');
    emailElement.textContent = `От: ${email.from}, Тема:
${email.subject}`;
    emailsList.appendChild(emailElement);
  };
});
```

2.2 Серверная часть

2.2.1 Файл main.py

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from app.api.v1 import router as api_router
from app.core.config import settings

app = FastAPI(title=settings.PROJECT_NAME)

# Настройка CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Подключение маршрутов API
app.include_router(api_router, prefix="/api/v1")
```

2.2.2 Файл models.py

```
from sqlalchemy import Column, String, DateTime, Text
from app.database import Base

class TemporaryEmail(Base):
    __tablename__ = "temporary_emails"

    id = Column(String, primary_key=True, index=True)
    email = Column(String, unique=True, index=True)
    created_at = Column(DateTime)
    expires_at = Column(DateTime)

class EmailMessage(Base):
    __tablename__ = "email_messages"

    id = Column(String, primary_key=True, index=True)
    email_id = Column(String, index=True)
    from_address = Column(String)
    to_address = Column(String)
    subject = Column(String)
    body = Column(Text)
    received_at = Column(DateTime)
```

2.2.3 Файл email_service.py

```
import uuid
from datetime import datetime, timedelta
from app.database import SessionLocal
```

```
from app.models import TemporaryEmail

def generate_temporary_email():
    db = SessionLocal()
    email = f"temp_{uuid.uuid4().hex}@example.com"

    # Сохранение временного email в базе данных
    db_email = TemporaryEmail(
        id=str(uuid.uuid4()),
        email=email,
        created_at=datetime.utcnow(),
        expires_at=datetime.utcnow() + timedelta(days=1)
    )
    db.add(db_email)
    db.commit()
    db.refresh(db_email)

    return {"email": email}
```

2.2.4 Файл smtp_service.py

```
from aiosmtpd.controller import Controller
from aiosmtpd.handlers import Sink
from app.email_parser import parse_email
from app.database import SessionLocal
from app.models import EmailMessage

class CustomHandler(Sink):
    async def handle_DATA(self, server, session, envelope):
        email_data = parse_email(envelope.content.decode('utf-8'))
        db = SessionLocal()

        # Сохранение письма в базе данных
        db_email = EmailMessage(
            id=str(uuid.uuid4()),
            email_id=email_data["to"],
            from_address=email_data["from"],
            to_address=email_data["to"],
            subject=email_data["subject"],
            body=email_data["body"],
            received_at=datetime.utcnow()
        )
        db.add(db_email)
        db.commit()
        db.refresh(db_email)

        return "250 OK"

# Запуск SMTP-сервера
controller = Controller(CustomHandler(), hostname='0.0.0.0',
port=8025)
controller.start()
```

2.2.5 Файл websocket.py

```
from fastapi import WebSocket, WebSocketDisconnect
from fastapi.websockets import WebSocketState
from app.database import SessionLocal
from app.models import EmailMessage
```

```
import json
import asyncio

class ConnectionManager:
    def __init__(self):
        self.active_connections = []

    async def connect(self, websocket: WebSocket):
        await websocket.accept()
        self.active_connections.append(websocket)

    def disconnect(self, websocket: WebSocket):
        self.active_connections.remove(websocket)

    async def send_email_notification(self, email: EmailMessage):
        notification = {
            "from": email.from_address,
            "subject": email.subject,
            "body": email.body
        }
        for connection in self.active_connections:
            if connection.client_state == WebSocketState.CONNECTED:
                await connection.send_text(json.dumps(notification))

manager = ConnectionManager()

@app.websocket("/ws/emails")
async def websocket_endpoint(websocket: WebSocket):
    await manager.connect(websocket)
    try:
        while True:
            db = SessionLocal()
            emails =
db.query(EmailMessage).order_by(EmailMessage.received_at.desc()).all()
            for email in emails:
                await manager.send_email_notification(email)
            await asyncio.sleep(5)
    except WebSocketDisconnect:
        manager.disconnect(websocket)
```