Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 5.3

по курсу «Разработка мобильных приложений»

«WebSockets»

Студент группы ИУ9-72Б Шемякин В.А.

Преподаватель Посевин Д. П.

Москва 2025

1 Задача

На стороне мобильного приложения должна выводиться форма ввода прарметров, которые передаются на WbSocket-сервер, на котором в свою очередь происходят вычисления, приведенные в вариантах. Результат вычисления должен асинхронно выводиться в виджет приложения.

Вариант 4: подсчёт слов в предложении, заданном последовательностью символов. Слова разделяются одним или несколькими пробелами.

2 Практическая реализация

Код представлен в Листингах 1-2.

Листинг 1 - main.dart

```dart
import 'dart:convert';
import 'package:flutter/material.dart';
import 'package:web_socket_channel/web_socket_channel.dart';
import 'package:shared_preferences/shared_preferences.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: '                          (WS)',
      theme: ThemeData(colorSchemeSeed: Colors.indigo, useMaterial3: true),
      home: const HomePage(),
    );
  }
}

class HomePage extends StatefulWidget {
  const HomePage({super.key});
  @override
  State<HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  static const _kSavedTextKey = 'saved_input_text';
  final _textController = TextEditingController();
  final _serverController = TextEditingController(text: 'ws://127.0.0.1:8765'
      );
  SharedPreferences? _prefs;
  WebSocketChannel? _channel;
  int? _wordCount;
  List<String> _words = [];
  String? _error;
  bool _connecting = false;
```

```
@override
void initState() {
  super.initState();
  _initPrefsAndRestore();
  _textController.addListener(_persistText);
}

Future<void> _initPrefsAndRestore() async {
  _prefs = await SharedPreferences.getInstance();
  final saved = _prefs!.getString(_kSavedTextKey);
  if (saved != null && saved.isNotEmpty) {
    _textController.value = TextEditingValue(
      text: saved,
      selection: TextSelection.collapsed(offset: saved.length),
    );
  }
}

void _persistText() {
  final p = _prefs;
  if (p != null) {
    p.setString(_kSavedTextKey, _textController.text);
  }
}

@override
void dispose() {
  _textController.removeListener(_persistText);
  _textController.dispose();
  _serverController.dispose();
  _channel?.sink.close();
  super.dispose();
}

void _connect() {
  _channel?.sink.close();
  setState(() {
    _connecting = true;
    _error = null;
  });
  try {
    _channel = WebSocketChannel.connect(Uri.parse(_serverController.text));
    _channel!.stream.listen((event) async {
      final data = jsonDecode(event);
      void _applyResultLike(Map<String, dynamic> data) {
        final original = (data['original'] as String?) ?? '';
```

```
          final words = original.trim().split(RegExp(r'\s+')).where((w) => w.
              isNotEmpty).toList();
          final count = data['word_count'] is int ? data['word_count'] as int
               : words.length;
          setState(() {
            _wordCount = count;
            _words = words;
            _error = null;
          });
          if (_textController.text.trim().isEmpty && original.isNotEmpty) {
            _textController.value = TextEditingValue(
              text: original,
              selection: TextSelection.collapsed(offset: original.length),
            );
          }
        }
        if (data['type'] == 'result') {
          _applyResultLike(data);
          final p = _prefs;
          if (p != null) {
            await p.setString(_kSavedTextKey, data['original'] ?? '');
          }
        } else if (data['type'] == 'restore') {
          _applyResultLike(data);
          final p = _prefs;
          if (p != null) {
            await p.setString(_kSavedTextKey, data['original'] ?? '');
          }
        } else if (data['type'] == 'error') {
          setState(() {
            _error = data['message']?.toString();
          });
        }
      }, onError: (e) {
        setState(() {
          _error = e.toString();
        });
      }, onDone: () {
        setState(() {
          _error ??= '                                        ';
        });
      });
    } catch (e) {
      setState(() {
        _error = e.toString();
      });
```

```
    } finally {
      setState(() {
        _connecting = false;
      });
    }
  }

  void _send() {
    final ch = _channel;
    if (ch == null) {
      setState(() {
        _error = '                                                    ';
      });
      return;
    }
    final payload = jsonEncode({'text': _text_controller_text});
    ch.sink.add(payload);
    _persistText();
  }

  String get _text_controller_text => _textController.text;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('
          WebSocket')),
      body: ListView(
        padding: const EdgeInsets.all(16),
        children: [
          Row(
            children: [
              Expanded(
                child: TextField(
                  controller: _serverController,
                  decoration: const InputDecoration(
                    labelText: 'WS              ',
                    hintText: 'ws://185.102.139.168:8765',
                    border: OutlineInputBorder(),
                  ),
                ),
              ),
              const SizedBox(width: 12),
              FilledButton.icon(
                onPressed: _connecting ? null : _connect,
                icon: const Icon(Icons.link),
```

```
            label: Text(_channel == null ? '                              ' : '
                                        .'),
          ),
        ],
      ),
      const SizedBox(height: 16),
      TextField(
        controller: _textController,
        minLines: 3,
        maxLines: 6,
        decoration: const InputDecoration(
          labelText: '                                          ',
          hintText: '                :
                      ',
          border: OutlineInputBorder(),
        ),
      ),
      const SizedBox(height: 12),
      Row(
        children: [
          FilledButton.icon(
            onPressed: _send,
            icon: const Icon(Icons.send),
            label: const Text('                    '),
          ),
          const SizedBox(width: 12),
          if (_error != null)
            Flexible(
              child: Text(
                _error!,
                style: TextStyle(color: Theme.of(context).colorScheme.
                    error),
              ),
            ),
        ],
      ),
      const SizedBox(height: 24),
      Center(
        child: _wordCount == null
            ? const Text('                                          ')
            : TweenAnimationBuilder<double>(
                duration: const Duration(milliseconds: 400),
                tween: Tween(begin: 0, end: _wordCount!.toDouble()),
                builder: (context, value, _) => Column(
                  children: [
```

```
                    Text('                    ', style: Theme.of(context).
                        textTheme.titleMedium),
                    Text(
                      value.toStringAsFixed(0),
                      style: Theme.of(context).textTheme.displayLarge?.
                          copyWith(fontWeight: FontWeight.w600),
                    ),
                  ],
                ),
              ),
            ),
          ),
          const SizedBox(height: 16),
          if (_words.isNotEmpty) ...[
            Text('          ', style: Theme.of(context).textTheme.titleMedium
                ),
            const SizedBox(height: 8),
            Wrap(
              spacing: 8,
              runSpacing: 8,
              children: _words.map((w) {
                return Chip(
                  label: Text(w),
                  padding: const EdgeInsets.symmetric(horizontal: 8, vertical
                      : 4),
                  side: BorderSide(color: Theme.of(context).colorScheme.
                      outlineVariant),
                );
              }).toList(),
            ),
            const SizedBox(height: 12),
            Card(
              elevation: 0,
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(12),
                side: BorderSide(color: Theme.of(context).colorScheme.
                    outlineVariant),
              ),
              child: ListView.separated(
                shrinkWrap: true,
                physics: const NeverScrollableScrollPhysics(),
                itemCount: _words.length,
                separatorBuilder: (_, __) => const Divider(height: 1),
                itemBuilder: (context, i) {
                  final w = _words[i];
                  return ListTile(
                    leading: CircleAvatar(child: Text('${i + 1}')),
```

```
                    title: Text(w),
                    subtitle: Text('          : ${w.length}'),
                  );
                },
              ),
            ),
          ],
        ],
      ),
    );
  }
}
```

Листинг 2 - server.py

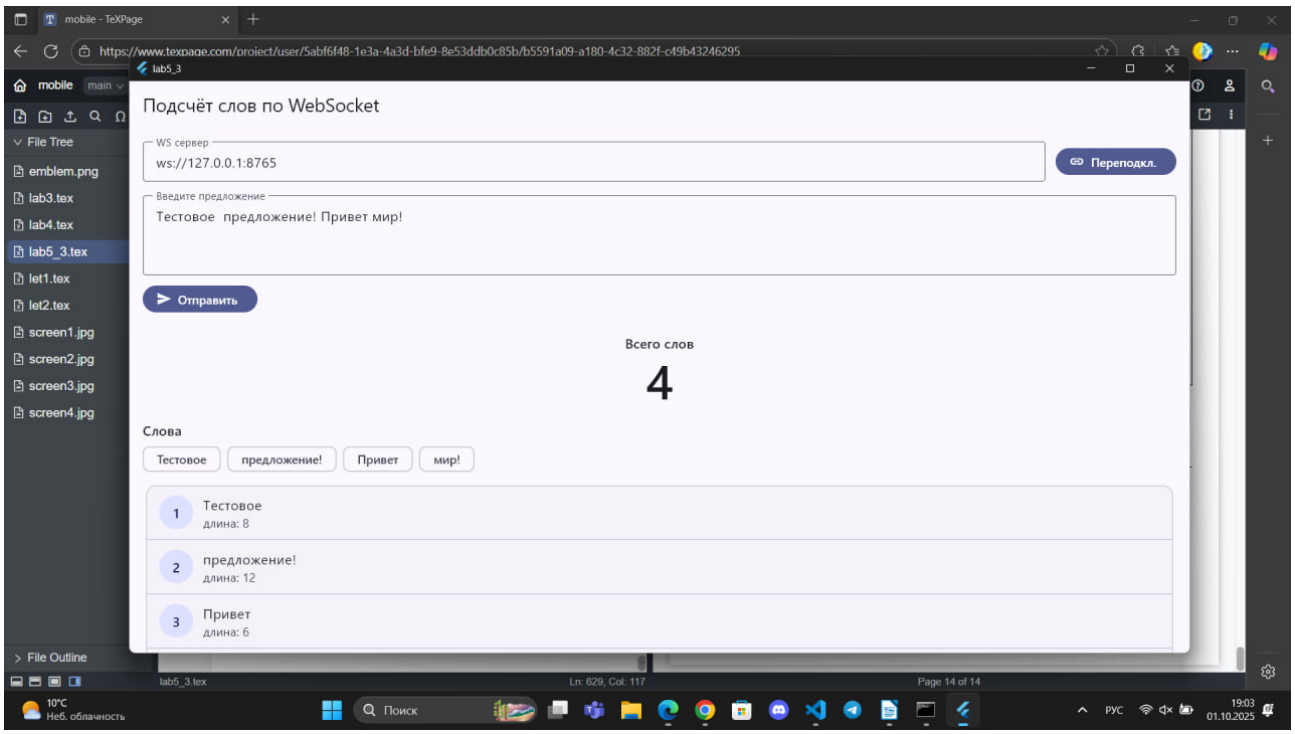
```
import asyncio
import json
import re
from datetime import datetime
from pathlib import Path
import websockets
from websockets.exceptions import ConnectionClosedError

HOST = "0.0.0.0"
PORT = 8765
STATE_PATH = Path("state.json")

def analyze_text(text: str):
    tokens = [t for t in re.split(r"\s+", text.strip()) if t]
    count = len(tokens)
    length_hist = {}
    for t in tokens:
        l = len(t)
        length_hist[l] = length_hist.get(l, 0) + 1
    return {
        "type": "result",
        "original": text,
        "word_count": count,
        "length_hist": length_hist,
        "tokens": tokens,
        "updated_at": datetime.now().isoformat(timespec="seconds"),
    }

def log_request(client, text: str, result: dict):
    ts = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    ip = None
```

```
    port = None
    if client is not None and isinstance(client, tuple) and len(client) >= 2:
        ip, port = client[0], client[1]
    tokens = result.get("tokens", [])
    word_count = result.get("word_count", 0)
    length_hist = result.get("length_hist", {})
    preview = text.replace("\n", "\\n")
    if len(preview) > 200:
        preview = preview[:200] + "    "
    hist_lines = []
    for l in sorted(length_hist.keys()):
        hist_lines.append(f"              {l}: {length_hist[l]}")
    print(
        "\n".join(
            [
                "    " * 72,
                f"[{ts}]                    {ip or '?'}:{port or '?'}",
                f"                              ({len(text)}          .): \"{
                    preview}\"",
                f"         : {word_count}",
                f"                        : {tokens}" if tokens else "
                                              : []",
                "                                    :" if hist_lines else "
                                                 : (          )",
                *hist_lines,
                "    " * 72,
            ]
        )
    )

def save_state(result: dict):
    payload = {
        "type": "restore",
        "original": result.get("original", ""),
        "word_count": result.get("word_count", 0),
        "length_hist": result.get("length_hist", {}),
        "updated_at": result.get("updated_at"),
    }
    tokens = result.get("tokens")
    if tokens is not None:
        payload["tokens"] = tokens
    STATE_PATH.write_text(json.dumps(payload, ensure_ascii=False, indent=2),
        encoding="utf-8")

def load_state():
    if not STATE_PATH.exists():
```

```
        return None
    try:
        data = json.loads(STATE_PATH.read_text(encoding="utf-8"))
        data["type"] = "restore"
        return data
    except Exception as e:
        print(f"[WARN]                                          {STATE_PATH}: {
            e}")
        return None

async def handler(websocket):
    snapshot = load_state()
    if snapshot:
        try:
            await websocket.send(json.dumps(snapshot, ensure_ascii=False))
        except ConnectionClosedError:
            return
    try:
        async for message in websocket:
            try:
                data = json.loads(message)
            except json.JSONDecodeError:
                await websocket.send(
                    json.dumps(
                        {
                            "type": "error",
                            "message": "Expected JSON with {\"text\":
                                \"...\"}",
                        },
                        ensure_ascii=False,
                    )
                )
                continue
            if data.get("type") == "restore":
                snap = load_state()
                if snap:
                    await websocket.send(json.dumps(snap, ensure_ascii=False)
                        )
                else:
                    await websocket.send(
                        json.dumps(
                            {
                                "type": "restore",
                                "original": "",
                                "word_count": 0,
                                "length_hist": {},
```

```
                                "updated_at": None,
                            },
                            ensure_ascii=False,
                        )
                    )
                continue
            text = data.get("text", "")
            result = analyze_text(text)
            log_request(websocket.remote_address, text, result)
            save_state(result)
            result_to_client = {
                "type": "result",
                "original": result["original"],
                "word_count": result["word_count"],
                "length_hist": result["length_hist"],
                "updated_at": result["updated_at"],
            }
            await websocket.send(json.dumps(result_to_client, ensure_ascii=
                False))
    except ConnectionClosedError:
        pass

async def main():
    print(f"Starting WebSocket server on ws://{HOST}:{PORT}")
    async with websockets.serve(
        handler, HOST, PORT, ping_interval=20, ping_timeout=20
    ):
        await asyncio.Future()

if __name__ == "__main__":
    asyncio.run(main())
```

В результате работы программы получился следующий вывод:



3 Заключение

В ходе лабораторной работы удалось работу с WebSocket сервером на примере с подсчетом количества строк в предложении