Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра программного обеспечения информационных технологий
Дисциплина: Разработка программного обеспечения для мобильных платформ

ОТЧЕТ

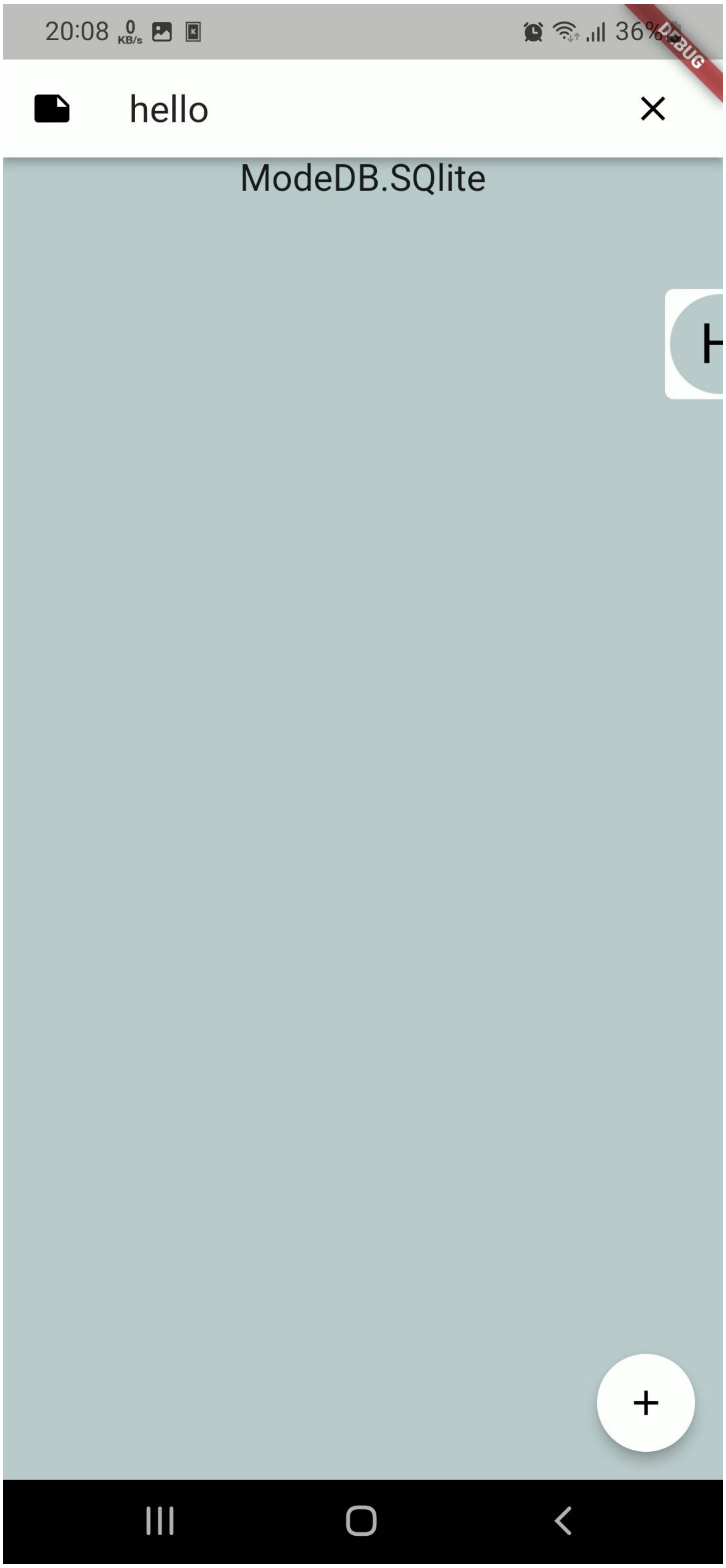
По лабораторной работе №1

Тема работы: «Записная книжка»

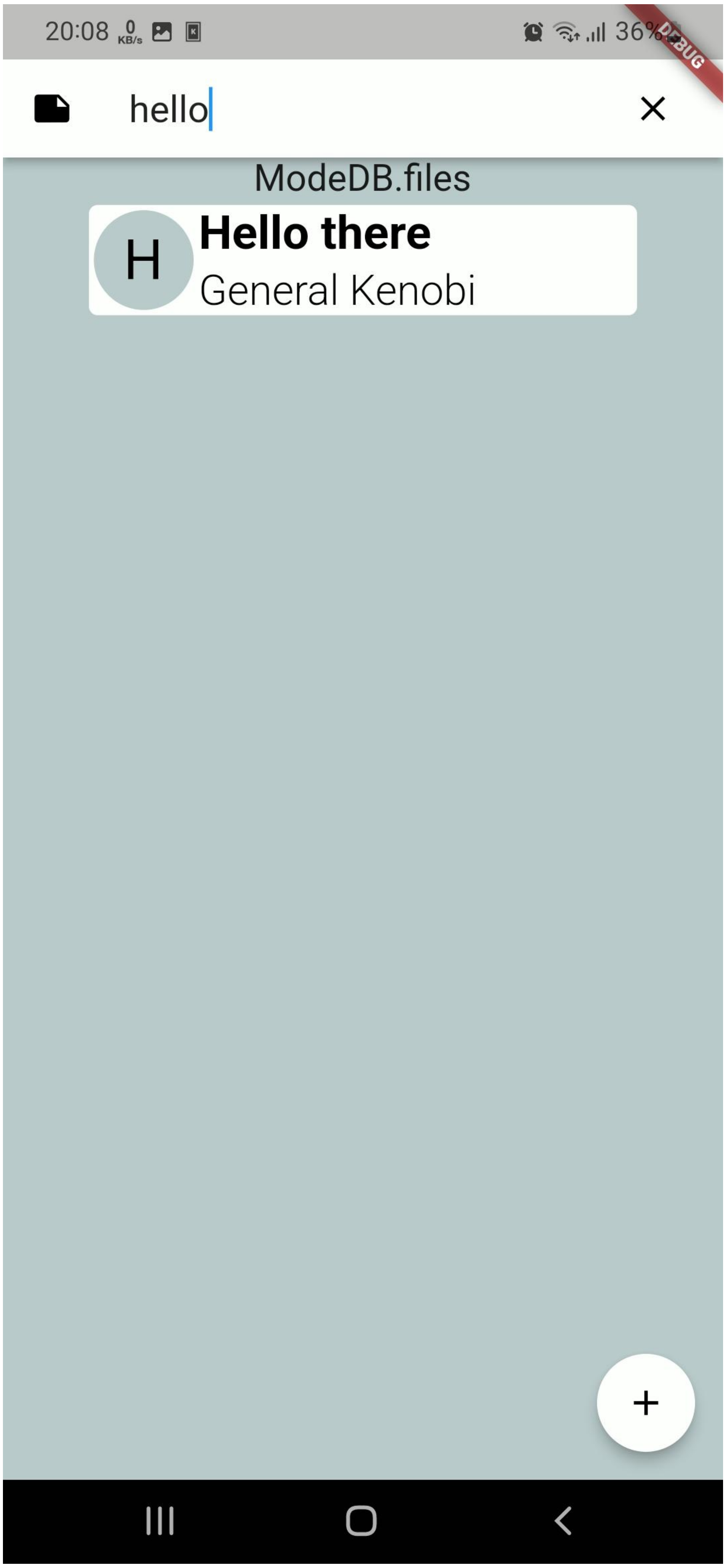
Выполнил:
студент: гр. 051006 Шуляк А.В.

Проверил: Коловайтис Н. А.

Минск 2023

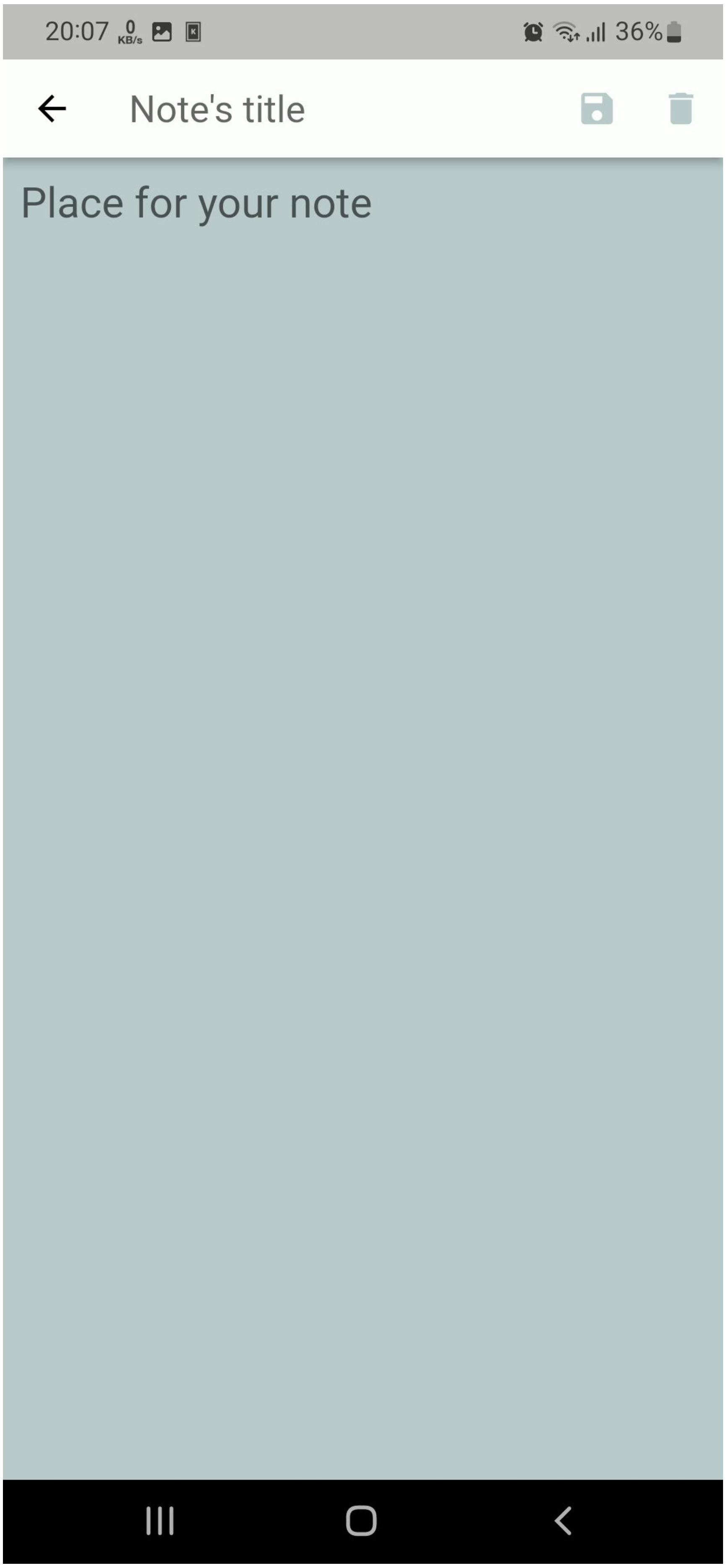


20:08
36%
DEBUG
hello
ModeDB.SQlite
H

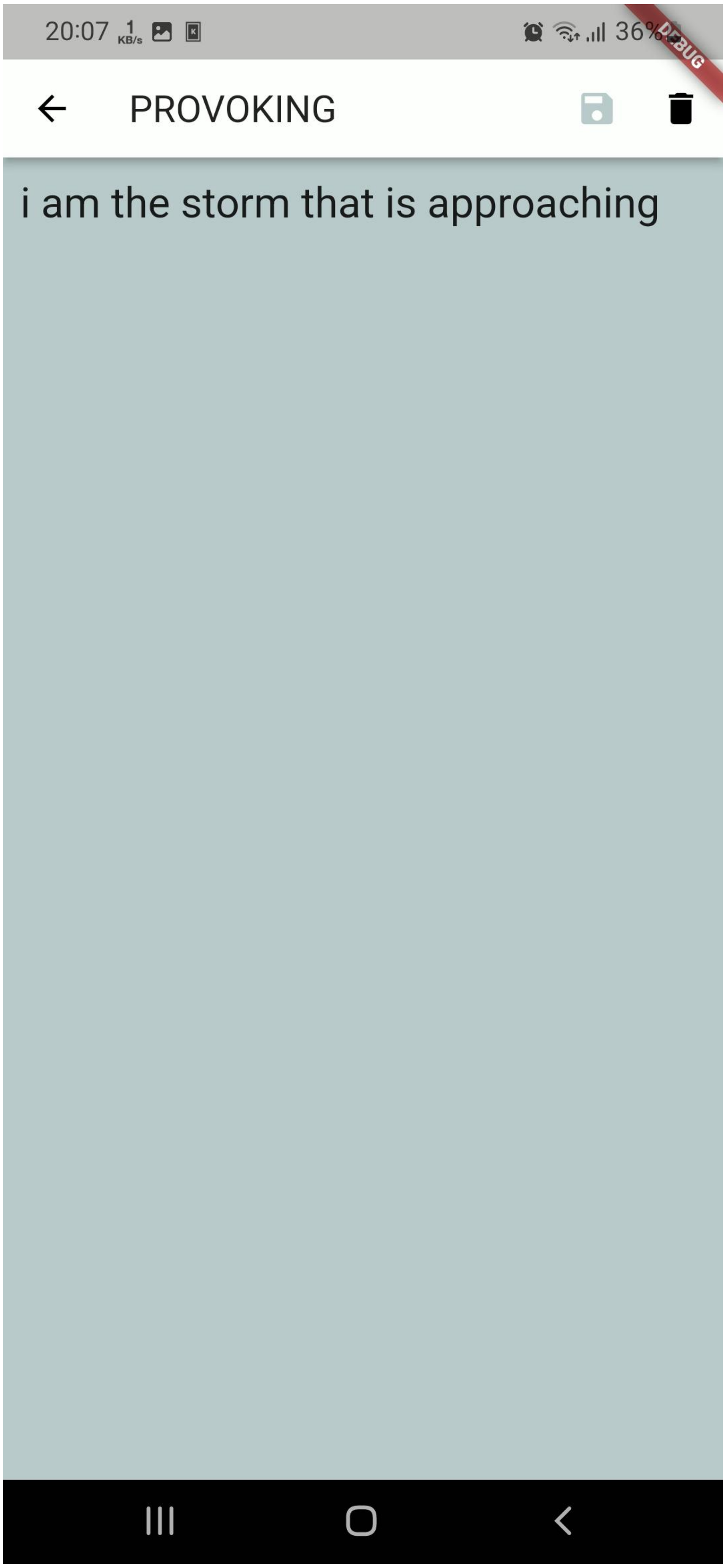


ModeDB.files

H
Hello there
General Kenobi



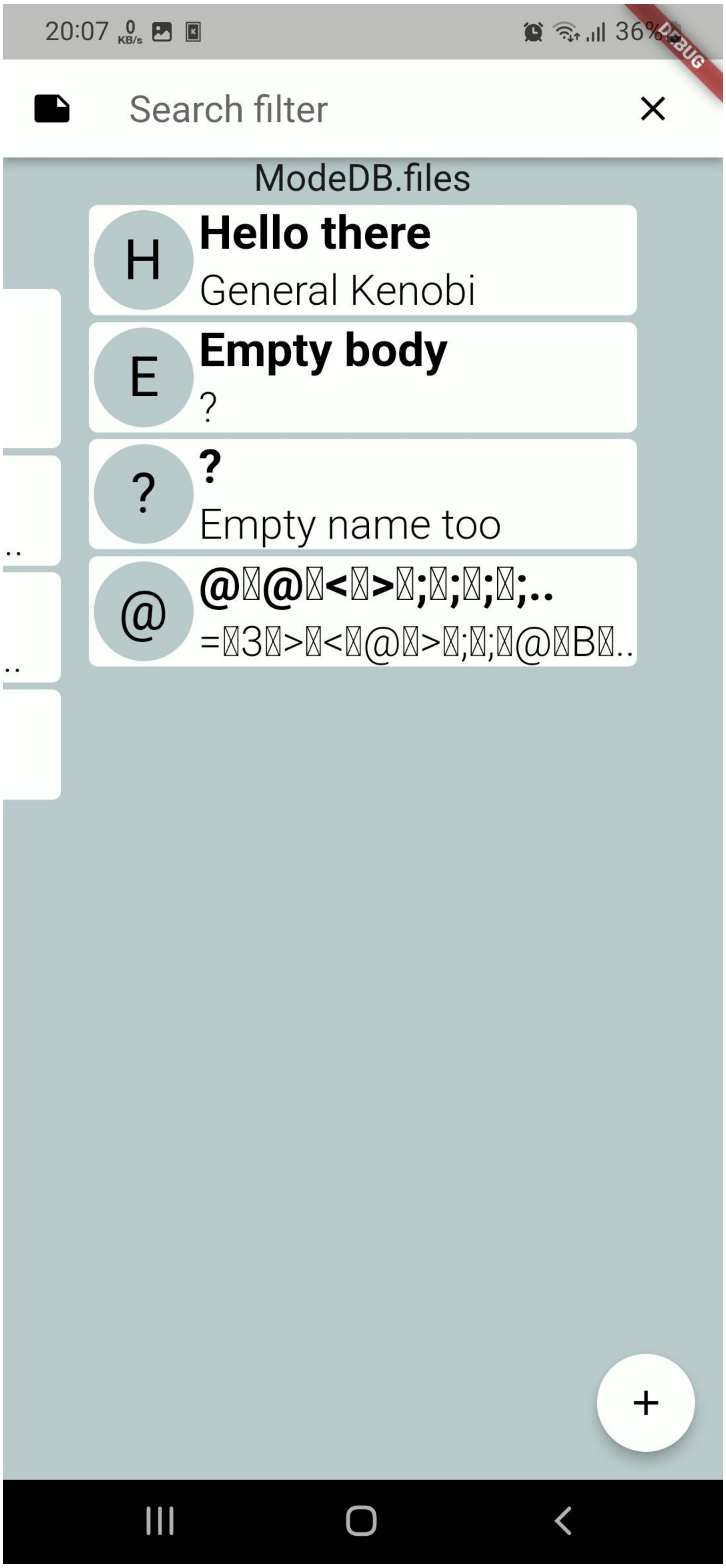
Place for your note



20:07 1 KB/s 36% DEBUG

PROVOKING

i am the storm that is approaching



20:07 36%

DEBUG

Search filter

ModeDB.files

H **Hello there**
General Kenobi

E **Empty body**
?

? **?**
Empty name too

@ **@�@�<�>�;�;�;�;..**
=�3�>�<�@�>�;�;�@�B�..

Исходный код:

main.dart

```
import 'package:flutter/material.dart';

import './screens/home.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Home(),
    );
  }
}
```

IDatabase.dart:

```
import 'Note.dart';

import 'Note.dart';

abstract class IDatabase {
  initDB() async {
    throw UnimplementedError();
  }

  Future<int> insertNote(Note note) async {
    throw UnimplementedError();
  }

  Future<int> updateNote(Note note) async {
    throw UnimplementedError();
  }

  Future<List<Map<String, dynamic>>> getAllNotes() async {
    throw UnimplementedError();
  }

  Future<Map<String, dynamic>?> getNote(int id) async {
    throw UnimplementedError();
  }

  Future<int> deleteNote(int id) async {
    throw UnimplementedError();
  }

  closeDB() async {
    throw UnimplementedError();
  }
}
```

DbHelper.dart:

```
import 'package:flutter_test_app/models/AppDataDB.dart';
import 'package:flutter_test_app/models/IDatabase.dart';
import 'package:flutter_test_app/models/SQLiteDB.dart';
```

```
enum ModeDB { SQlite, files }

mixin dbHelper {
  static IDatabase getDbViaMode(ModeDB mode) {
    late IDatabase db;
    switch (mode) {
      case ModeDB.SQlite:
        db = SQLiteDB();
        break;
      case ModeDB.files:
        db = AppDataDB();
        break;
      default:
        throw Exception("No db info provided for note editor");
    }
    return db;
  }

  static String getShowableDbName(ModeDB mode) {
    String res = "";
    switch (mode) {
      case ModeDB.SQlite:
        res = "SQLite";
        break;
      case ModeDB.files:
        res = "Files";
        break;
      default:
        throw Exception("No db info provided for note editor");
    }
    return res;
  }
}
```

AppDataDB.dart:

```
import 'dart:ffi';
import 'dart:io';
import 'dart:convert';
import 'package:flutter_test_app/models/IDatabase.dart';
import 'package:flutter_test_app/models/Note.dart';
import 'package:path_provider/path_provider.dart';

class AppDataDB implements IDatabase {
  static List<int> ids = List.empty(growable: true);
  static String? path;
  static late Directory folder;

  static String getPath(String file) {
    return "$path/$file";
  }

  static File getFile(String file) {
    return File(getPath(file));
  }

  @override
  closeDB() {}
```

```
@override
initDB() async {
  if (path == null) {
    String appdir = (await getApplicationDocumentsDirectory()).path;
    path = "$appdir/notes";
    folder = Directory(path!);
    if (!await folder.exists()) {
      await folder.create(recursive: true);
    }
  }
}

Future<List<int>> getIds() async {
  var res = List<int>.empty(growable: true);
  var items = folder.listSync(followLinks: false, recursive: false);
  for (var item in items) {
    var str = item.path.split('/').last.split('.').first;
    res.add(int.parse(str));
  }
  return res;
}

@override
Future<int> deleteNote(int id) async {
  var ids = await getIds();
  if (ids.contains(id)) {
    var file = getFile("$id.txt");
    await file.delete();
  }
  return 0;
}

@override
Future<List<Map<String, dynamic>>> getAllNotes() async {
  var ids = await getIds();
  var res = List<Map<String, dynamic>>.empty(growable: true);
  for (int i = 0; i < ids.length; i++) {
    var item = await getNote(i);
    if (item != null) {
      res.add(item);
    }
  }
  return res;
}

@override
Future<Map<String, dynamic>?> getNote(int id) async {
  var ids = await getIds();
  if (!ids.contains(id)) {
    return <String, dynamic>{};
  } else {
    var file = getFile("$id.txt");
    String str = await file.readAsString();
    var obj = jsonDecode(str);
    var note = Note();
    note.id = obj['id'];

    var tmp = obj['name'].cast<int>();
    tmp.removeWhere((int x) => x == 0);
    note.name = String.fromCharCodes(tmp);
```

```
      tmp = obj['content'].cast<int>();
      tmp.removeWhere((int x) => x == 0);
      note.content = String.fromCharCodes(tmp);

      note.tsCreated =
          DateTime.fromMillisecondsSinceEpoch(obj['tsCreated'] * 1000);
      note.tsUpdated =
          DateTime.fromMillisecondsSinceEpoch(obj['tsUpdated'] * 1000);
      return note.toMap();
    }
  }

  @override
  Future<int> insertNote(Note note) async {
    var ids = await getIds();
    int newId = 0;
    if (ids.isNotEmpty) {
      newId = ids[0];
      for (int i = 0; i < ids.length; i++) {
        if (ids[i] > newId) {
          newId = ids[i];
        }
        newId++;
      }
    }
    note.id = newId;
    String str = jsonEncode(note.toMap());
    var file = getFile("$newId.txt");
    await file.writeAsString(str);
    return 0;
  }

  @override
  Future<int> updateNote(Note note) async {
    return insertNote(note);
  }
}
```

sqllitedb.dart

```
import 'package:flutter_test_app/models/Note.dart';
import 'package:sqflite/sqflite.dart';
import 'IDatabase.dart';
import 'package:mutex/mutex.dart';

class SQLiteDB implements IDatabase {
  static const _name = "SQLiteNotesDatabase.db";
  static const _version = 1;

  late Database database;
  static const tableName = 'notes';

  static int cnt = 0;
  final m = Mutex();

  @override
  initDB() async {
    await m.acquire();
  try {
    cnt++;
    database = await openDatabase(_name, version: _version,
        onCreate: (Database db, int version) async {
      await db.execute('''CREATE TABLE $tableName (
              id INTEGER PRIMARY KEY AUTOINCREMENT,
              name TEXT,
              content TEXT,
              tsCreated INTEGER,
              tsUpdated INTEGER
              )''');
    });
  } finally {
    m.release();
  }
}

@override
closeDB() async {
  await m.acquire();
  try {
    cnt--;
    if (cnt <= 0) {
      await database.close();
    }
  } finally {
    m.release();
  }
}

@override
Future<int> deleteNote(int id) async {
  return await database.delete(tableName, where: 'id = ?', whereArgs: [id]);
}

@override
Future<List<Map<String, dynamic>>> getAllNotes() async {
  return await database.query(tableName);
}

@override
Future<Map<String, dynamic>?> getNote(int id) async {
  var result =
      await database.query(tableName, where: 'id = ?', whereArgs: [id]);

  if (result.isNotEmpty) {
    return result.first;
  }

  return null;
}

@override
Future<int> insertNote(Note note) async {
  return await database.insert(tableName, note.toMap(),
      conflictAlgorithm: ConflictAlgorithm.replace);
}

@override
Future<int> updateNote(Note note) async {
    return await database.update(tableName, note.toMap(),
        where: 'id = ?',
        whereArgs: [note.id],
        conflictAlgorithm: ConflictAlgorithm.replace);
  }
}
```

Note.dart

```
import 'dart:convert';

import
'package:zooper_flutter_encoding_utf16/zooper_flutter_encoding_utf16.dart';

class Note {
  int? id;
  String name;
  String content;
  DateTime? tsCreated;
  DateTime? tsUpdated;

  final encoder = UTF16LE();

  // Note(this.id, this.name, this.content, this.tsCreated, this.tsUpdated);
  Note({
    this.name = "New Note",
    this.content = "",
  }) {
    tsCreated = DateTime.now();
    tsUpdated = tsCreated;
  }

  Map<String, dynamic> toMap() {
    var data = {
      'id': id,
      'name': encoder.encode(name),
      'content': encoder.encode(content),
      'tsCreated': tsCreated!.millisecondsSinceEpoch ~/ 1000,
      'tsUpdated': tsUpdated!.millisecondsSinceEpoch ~/ 1000,
    };

    return data;
  }

  @override
  String toString() {
    return {
      'id': id,
      'name': utf8.encode(name),
      'content': utf8.encode(content),
      'tsCreated': tsCreated!.millisecondsSinceEpoch ~/ 1000,
      'tsUpdated': tsUpdated!.millisecondsSinceEpoch ~/ 1000,
    }.toString();
  }
}
```

colors.dart:

```
import 'package:flutter/material.dart';
```

```
const clBackground = Color(0xFFB9CACA);
const clMain = Color(0xFFFDFFFC);
const clMainContrast = Colors.black;

//old colors
const c1 = 0xFFFDFFFC, c2 = 0xFFFF595E, c3 = 0xFF374B4A, c4 = 0xFF00B1CC, c5 =
0xFFFFD65C, c6 = 0xFFB9CACA,
    c7 = 0x80374B4A, c8 = 0x3300B1CC, c9 = 0xCCFF595E;
```

home.dart:

```
import 'dart:async';
import 'dart:developer' as dlp;
import 'dart:math';
import 'package:dart_numerics/dart_numerics.dart';
import 'package:flutter/material.dart';
import 'package:carousel_slider/carousel_slider.dart';
import 'package:flutter_test_app/models/IDatabase.dart';
import 'package:flutter_test_app/models/dbHelper.dart';
import 'editor.dart';
import 'colors.dart';
import
'package:zooper_flutter_encoding_utf16/zooper_flutter_encoding_utf16.dart';

class Home extends StatefulWidget {
  const Home({super.key});
  @override
  _HomeState createState() => _HomeState();
}

class _HomeState extends State<Home> {
  Map<ModeDB, List<Map<String, dynamic>>> notes = {};

  static late _HomeState self;

  _HomeState() {
    ModeDB.values.forEach((mode) {
      notes[mode] = <Map<String, dynamic>>[];
    });
    self = this;
  }

  ModeDB curMode = ModeDB.SQlite;

  void onUpdate() {
    setState(() {});
  }

  final TextEditingController filterController = TextEditingController();
  String filter = "";

  void handleFilterChange() {
    setState(() {
      filter = filterController.text.trim();
      dlp.log(filterController.text.trim());
    });
  }

  @override
```

```
void initState() {
  super.initState();
  filterController.addListener(handleFilterChange);
}

@override
void dispose() {
  filterController.dispose();
  super.dispose();
}

@override
Widget build(BuildContext context) {
  final decoder = UTF16LE();
  return MaterialApp(
    title: 'Notes',
    home: Scaffold(
      backgroundColor: clBackground,
      appBar: AppBar(
          automaticallyImplyLeading: false,
          backgroundColor: clMain,
          leading: const Icon(
            Icons.note,
            color: clMainContrast,
          ),
          title: TextField(
            maxLines: 1,
            decoration: InputDecoration(
              hintText: 'Search filter',
              border: InputBorder.none,
              suffixIcon: IconButton(
                onPressed: () {
                  filterController.clear();
                },
                icon: const Icon(
                  Icons.clear,
                  color: clMainContrast,
                ),
              ),
            ),
            controller: filterController,
          )),
      body: CarouselSlider.builder(
        itemCount: ModeDB.values.length,
        itemBuilder: (BuildContext context, int itemIndex, int pageIndex) =>
            Column(
          mainAxisSize: MainAxisSize.max,
          children: [
            Text(ModeDB.values.elementAt(itemIndex).toString()),
            Expanded(
                child: FutureBuilder<List<Map<String, dynamic>>>(
              future: getAllNotes(ModeDB.values.elementAt(itemIndex)),
              builder: (context, snapshot) {
                if (snapshot.hasData) {
                  return ListView.builder(
                    itemCount: snapshot.data?.length,
                    itemBuilder: (context, index) {
                      dynamic item = snapshot.data?[index];
                      String name =
                          decoder.decode(item['name']).toLowerCase();
                      if (name.contains(filter.toLowerCase())) {
                        return DisplayNote(
                          note: item,
                          modeDB: curMode,
                        );
                      } else {
                        return Container();
                      }
                    },
                    shrinkWrap: true,
                  );
                } else if (snapshot.hasError) {
                  return const Text(
                      "Some errors occured while loading notes..");
                } else {
                  return const Center(
                    child: CircularProgressIndicator(backgroundColor: clMain),
                  );
                }
              },
            ))
          ],
        ),
        options: CarouselOptions(
          height: MediaQuery.of(context).size.height - 100.0,
          enlargeStrategy: CenterPageEnlargeStrategy.height,
          enlargeCenterPage: true,
          animateToClosest: true,
          autoPlay: false,
          initialPage: 0,
          enableInfiniteScroll: false,
          scrollDirection: Axis.horizontal,
          onPageChanged: (index, reason) {
            curMode = ModeDB.values[index];
          },
        ),
      ),
      floatingActionButton: FloatingActionButton(
        tooltip: 'New Note',
        backgroundColor: clMain,
        onPressed: () async {
          await Navigator.push(
              context,
              MaterialPageRoute(
                  builder: (context) => Editor(
                        modeDB: curMode,
                      ))).then(
            (value) {
              setState(() {});
            },
          );
        },
        child: const Icon(
          Icons.add,
          color: clMainContrast,
        ),
      ),
    ),
  );
}
```

```
  Future<List<Map<String, dynamic>>> getAllNotes(ModeDB modeDB) async {
    IDatabase db = dbHelper.getDbViaMode(modeDB);
    int id = Random().nextInt(0xFFFFFFFF);
    try {
      dlp.log(
          "$id: Trying to get all notes in db: 
${dbHelper.getShowableDbName(modeDB)}");
      await db.initDB();
      List<Map> notesList = await db.getAllNotes();
      List<Map<String, dynamic>> notesData =
          List<Map<String, dynamic>>.from(notesList);
      dlp.log("$id: ${notesData.length} note(s) got succesfully!");
      return notesData;
    } catch (e) {
      dlp.log("$id: Caught exception while trying access db: ${e.toString()}");
    }
    return List<Map<String, dynamic>>.empty();
  }
}

class DisplayNote extends StatelessWidget {
  final dynamic note;
  final ModeDB modeDB;
  const DisplayNote({Key? key, this.note, required this.modeDB})
      : super(key: key);

  @override
  Widget build(BuildContext context) {
    final decoder = UTF16LE();
    return Padding(
        padding: const EdgeInsets.symmetric(horizontal: 8.0, vertical: 2.0),
        child: Material(
            color: clMain,
            clipBehavior: Clip.hardEdge,
            borderRadius: BorderRadius.circular(5.0),
            child: InkWell(
              onTap: () async {
                await Navigator.push(
                    context,
                    MaterialPageRoute(
                        builder: (context) => Editor(
                              modeDB: modeDB,
                              noteItem: note,
                            ))).then((value) {
                  _HomeState.self.onUpdate();
                });
              },
              onLongPress: () async {
                var db = dbHelper.getDbViaMode(modeDB);
                await db.initDB();
                await db.deleteNote(note['id']).then((value) {
                  _HomeState.self.onUpdate();
                });
              },
              child: Row(
                children: [
                  Expanded(
                    flex: 1,
                    child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          crossAxisAlignment: CrossAxisAlignment.center,
          mainAxisSize: MainAxisSize.min,
          children: [
           Container(
            alignment: Alignment.center,
            decoration: const BoxDecoration(
             color: clBackground,
             shape: BoxShape.circle,
             // border: Border.all(),
            ),
            child: Padding(
             padding: const EdgeInsets.all(10),
             child: Text(
               getPreview((decoder.decode(note['name'])), 1)
                 .toUpperCase(),
               style: const TextStyle(
                color: clMainContrast,
                fontSize: 21,
               )),
            ),
           )
          ],
         ),
        ),
        Expanded(
          flex: 4,
          child: Column(
           mainAxisAlignment: MainAxisAlignment.spaceAround,
           crossAxisAlignment: CrossAxisAlignment.start,
           mainAxisSize: MainAxisSize.min,
           children: [
            Text(
             getPreview(decoder.decode(note['name']), 15,
               end: ".."),
             style: const TextStyle(
              color: clMainContrast,
              fontSize: 18,
              fontWeight: FontWeight.bold,
             ),
            ),
            Container(
             // decoration: BoxDecoration(border: Border.all()),
             height: 3,
            ),
            Text(
             getPreview(decoder.decode(note['content']), 20,
               end: ".."),
             style: const TextStyle(
              color: clMainContrast,
              fontSize: 16,
              fontWeight: FontWeight.w300,
             ),
            )
           ],
          ))
       ],
      ),
     )));
}
```

```
  String getPreview(String src, int count, {String end = ""}) {
    String res = "";
    String tmp = src.split("\n")[0];
    if (tmp.length < count) {
      res = tmp;
      if (tmp.length != src.length) res += end;
    } else {
      res = tmp.substring(0, count) + end;
    }
    if (res == "") res = "?";
    return res;
  }
}
```

editor.dart:

```
import 'dart:developer';

import 'package:flutter/material.dart';
import 'package:flutter_test_app/models/IDatabase.dart';
import 'package:flutter_test_app/models/SQLiteDB.dart';
import 'package:flutter_test_app/models/Note.dart';
import 'package:flutter_test_app/models/dbHelper.dart';
import 'colors.dart';
import
'package:zooper_flutter_encoding_utf16/zooper_flutter_encoding_utf16.dart';

class Editor extends StatefulWidget {
  ModeDB modeDB;
  dynamic noteItem;

  Editor({required this.modeDB, this.noteItem}) : super();

  _Editor createState() => _Editor(modeDB: modeDB, noteItem: noteItem);
}

class _Editor extends State<Editor> {
  ModeDB modeDB;
  late IDatabase db;

  dynamic noteItem;

  _Editor({required this.modeDB, this.noteItem}) : super() {
    db = dbHelper.getDbViaMode(modeDB);
    noteContent = "";
    noteTitle = "";
    _isHaveToSave = false;
  }

  final coder = UTF16LE();

  late String noteTitle;
  late String noteContent;

  final TextEditingController titleController = TextEditingController();
  final TextEditingController contentController = TextEditingController();

  void handleTitleChange() {
```

```
  setState(() {
    _isHaveToSave = true;
    noteTitle = titleController.text.trim();
  });
}

void handleContentChange() {
  setState(() {
    _isHaveToSave = true;
    noteContent = contentController.text.trim();
  });
}

saveNote() async {
  setState(() {
    _isHaveToSave = false;
  });
  if (noteTitle.length + noteContent.length != 0) {
    Note note = Note(
      name: noteTitle,
      content: noteContent,
    );
    if (noteItem != null) {
      note.id = noteItem['id'];
      note.tsCreated =
          DateTime.fromMillisecondsSinceEpoch(noteItem['tsCreated'] * 1000);
      note.tsUpdated = DateTime.now();
    }
    try {
      log("Trying save note..");
      await db.initDB();
      log("Db is opened");
      await db.insertNote(note);
      log("Note saved succesfully");
    } catch (e) {
      log("Exception while saving note: ${e.toString()}");
    }
  }
}

deleteNote() async {
  try {
    log("Trying delete note..");
    await db.initDB();
    log("Db is opened");
    await db.deleteNote(noteItem['id']);
    log("Note deleted succesfully");
  } catch (e) {
    log("Exception while deleting note: ${e.toString()}");
  }
}

handleDeleteAction() {
  deleteNote();
  Navigator.of(context).pop(true);
  setState(() {});
}

void handleBackArrow() {
  Navigator.pop(context);
```

```
}

bool _isHaveToSave = false;

@override
Widget build(BuildContext context) {
  return Scaffold(
      backgroundColor: clBackground,
      appBar: AppBar(
        backgroundColor: clMain,
        leading: IconButton(
          icon: const Icon(
            Icons.arrow_back,
            color: clMainContrast,
          ),
          tooltip: 'Back',
          onPressed: () => {handleBackArrow()},
        ),
        actions: [
          IconButton(
              onPressed: () async =>
                  {_isHaveToSave ? await saveNote() : null},
              icon: Icon(
                Icons.save,
                color: (_isHaveToSave ? clMainContrast : clBackground),
              )),
          IconButton(
            onPressed: () => {noteItem != null ? handleDeleteAction() : null},
            icon: Icon(
              Icons.delete,
              color: (noteItem != null ? clMainContrast : clBackground),
            ),
          )
        ],
        title: TextField(
          maxLines: 1,
          decoration: const InputDecoration(
              hintText: 'Note\'s title', border: InputBorder.none),
          controller: titleController,
        ),
      ),
      body: Padding(
        padding: const EdgeInsets.symmetric(vertical: 0.0, horizontal: 10.0),
        child: Stack(children: [
          TextField(
            maxLines: null,
            keyboardType: TextInputType.multiline,
            decoration: const InputDecoration(
                hintText: 'Place for your note', border: InputBorder.none),
            controller: contentController,
          ),
        ]),
      ));
}

@override
void initState() {
  super.initState();
  if (noteItem != null) {
    titleController.text = coder.decode(noteItem["name"]);
```

```
      contentController.text = coder.decode(noteItem["content"]);
      noteContent = contentController.text.trim();
      noteTitle = titleController.text.trim();
    } else {
      titleController.text = "";
      contentController.text = "";
    }
    titleController.addListener(handleTitleChange);
    contentController.addListener(handleContentChange);
  }

  @override
  void dispose() {
    titleController.dispose();
    contentController.dispose();
    super.dispose();
  }
}
```