МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«УДМУРТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(Филиал ФГБОУ ВО «УдГУ» в г. Воткинске)

ОТЧЕТ ПО ПРАКТИКЕ

Вид практики: производственная практика, преддипломная

(Вид производственной практики, ненужное убрать)

Специальность 09.02.07 Информационные системы и программирование

(Код, наименование специальности)

Профессиональный модуль: **нет**

Обучающегося 4 курса, группы СПО-09-Вт-090207-41(к)

Форма обучения \_\_\_\_\_\_\_\_очная\_\_\_\_\_\_\_\_\_\_

(очная, заочная)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество)

Место практики \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Название организации)

Срок практики с 05.05.2025 по 01.06.2025

Руководители практики

от организации \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

(должность) (подпись) (ФИО)

от филиала УдГУ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ Мамрыкин О. В.

(должность) (подпись) (ФИО)

Итоговая оценка по практике \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Воткинск 2025

[Введение 3](#_Toc197934102)

[Общая характеристика предприятия (организации) 3](#_Toc197934103)

[Анализ делопроизводства организации ООО «Рога и копыта», построение схемы бизнес-процессов делоппроизводства 3](#_Toc197934104)

[1 Доступные способы сохранения состояний во flame 4](#_Toc197934105)

[1.1 Использование SharedPreferences 4](#_Toc197934106)

[1.2 Сохранение данных в локальные файлы во Flutter (dart:io + path\_provider) 4](#_Toc197934107)

[1.3 Использование Hive (NoSQL-база данных) 5](#_Toc197934108)

[Описывается основная суть вашей работы, преимущества, которые получит организация и пользователи системы после модификации системы 7](#_Toc197934109)

[Заключение 7](#_Toc197934110)

[Список использованных источников 7](#_Toc197934111)

[Приложения 8](#_Toc197934112)

# Введение

В современной разработке мобильных и десктопных приложений на Flutter важную роль играет игровой движок Flame, который предоставляет удобные инструменты для создания 2D-игр. Одной из ключевых задач при разработке игр является сохранение и восстановление состояния игры, что позволяет пользователям продолжить игровой процесс с того же места после перезапуска приложения.

Актуальность данной темы обусловлена необходимостью обеспечения удобства пользователей и повышения стабильности работы приложений. Реализация механизмов сохранения и загрузки состояния в играх на Flutter Flame требует понимания работы движка, а также использования подходящих методов сериализации и хранения данных.

Целью данной практики является изучение и реализация механизмов сохранения и загрузки состояния в игровом движке Flame. В ходе работы предстоит рассмотреть следующие задачи:

- Изучение возможностей Flame для управления состоянием игры.

- Анализ методов сериализации данных (JSON, binary, Hive и др.).

- Реализация сохранения и восстановления игрового прогресса.

- Тестирование работоспособности решения на различных платформах.

Практическая значимость работы заключается в применении полученных знаний для создания устойчивого и удобного механизма сохранения состояния, что может быть использовано в дальнейшей разработке игровых проектов на Flutter.

# Общая характеристика предприятия (организации)

# Анализ делопроизводства организации ООО «Рога и копыта», построение схемы бизнес-процессов делоппроизводства

# 1 Доступные способы сохранения состояний во flame

## 1.1 Использование SharedPreferences

SharedPreferences — это легковесное решение для хранения простых данных в формате ключ-значение. Он использует механизмы платформы (Android — SharedPreferences, iOS — NSUserDefaults), поэтому работает быстро и подходит для небольших данных (настройки, рекорды, флаги).

Как указано на странице самого расширения - обертывает платформу-специфическое постоянное хранилище для простых данных Данные могут сохраняться для диска асинхронно, и нет никакой гарантии, что записи будут сохраняться на диске после выполнения, поэтому этот плагин не должен использоваться для хранения критически важных данных.

Поддерживаемые типы: int, double, bool, String, List<String>.

Пример:

import 'package:shared\_preferences/shared\_preferences.dart';

// Сохранение

final prefs = await SharedPreferences.getInstance();

await prefs.setInt('highScore', 100);

// Загрузка

final highScore = prefs.getInt('highScore') ?? 0;

Несмотря на простоту использования к сожалению не подходит для сохранения сложных структур данных(нужно сериализовать например в json). Не гарантирует сохранения данных при записи.

## 1.2 Сохранение данных в локальные файлы во Flutter (dart:io + path\_provider)

Этот метод подходит для хранения сложных данных (JSON, бинарные файлы, игровые уровни) в файловой системе устройства. В отличие от SharedPreferences, он позволяет работать с большими объемами данных и структурированными объектами.

Этот метод позволяет сохранять любые данные(json, csv, бинарные данные, изображения и аудио) в локальные файлы. Можно сохранять объекты предварительно обработанные dart:convert.

Пример:

Допустим, у нас есть объект Player:

class Player {

  final String name;

  final int level;

  final List<String> inventory;

  Player(this.name, this.level, this.inventory);

  // Преобразование в Map для JSON

  Map<String, dynamic> toJson() => {

'name': name,

'level': level,

'inventory': inventory,

  };

  // Создание из Map

  factory Player.fromJson(Map<String, dynamic> json) => Player(

json['name'],

json['level'],

List<String>.from(json['inventory']),

  );

}

Мы можем сохранить его в файл следующим образом:

import 'dart:io';

import 'dart:convert';

import 'package:path\_provider/path\_provider.dart';

Future<void> savePlayer(Player player) async {

  final directory = await getApplicationDocumentsDirectory();

  final file = File('${directory.path}/player.json');

  await file.writeAsString(jsonEncode(player.toJson()));

}

И восстановить из файла как показано ниже:

Future<Player?> loadPlayer() async {

  try {

final directory = await getApplicationDocumentsDirectory();

final file = File('${directory.path}/player.json');

if (!await file.exists()) return null;

final jsonData = await file.readAsString();

return Player.fromJson(jsonDecode(jsonData));

  } catch (e) {

print('Ошибка загрузки: $e');

return null;

  }

}

Использование этого способа позволяет сохранять любые данные, но есть определенные минусы, необходимо самостоятельно сериализовать данные.

## 1.3 Использование Hive (NoSQL-база данных)

Hive — это легковесная, быстрая и удобная NoSQL-база данных, написанная на чистом Dart. Она отлично подходит для хранения структурированных данных (игровые объекты, настройки, инвентарь) и работает быстрее SQLite и SharedPreferences в большинстве сценариев.

Hive хранит данные в Box — это аналог таблиц в SQL. Каждый Box может содержать: примитивы (int, String, bool, List, Map), кастомные объекты (если зарегистрирован адаптер). Данные хранятся в формате ключ → значение. Ключи могут быть int или String. Значения — любые поддерживаемые типы. Для работы с кастомными классами (например, Player, Item) нужно создать адаптер.

Пример:

Инициализация Hive:

import 'package:hive/hive.dart';

import 'package:hive\_flutter/hive\_flutter.dart';

void main() async {

  await Hive.initFlutter(); // Инициализация для Flutter

  await Hive.openBox('settings'); // Открываем Box

  runApp(MyApp());

}

Работа с примитивами:

final settingsBox = Hive.box('settings');

// Запись данных

settingsBox.put('soundEnabled', true);

settingsBox.put('playerName', 'Hero');

settingsBox.put('highScore', 1000);

// Чтение данных

final isSoundOn = settingsBox.get('soundEnabled', defaultValue: true);

final playerName = settingsBox.get('playerName', defaultValue: 'Guest');

final highScore = settingsBox.get('highScore', defaultValue: 0);

// Удаление данных

settingsBox.delete('highScore');

Пример работы с кастомными объектами:

Подключение адаптера для класса Player:

void main() async {

  await Hive.initFlutter();

  Hive.registerAdapter(PlayerAdapter()); // Регистрация адаптера

  await Hive.openBox<Player>('players'); // Box для Player

  runApp(MyApp());

}

Сохранение и загрузка объектов:

final playersBox = Hive.box<Player>('players');

// Сохранение объекта

playersBox.put('currentPlayer', Player('Hero', 10, ['Sword', 'Potion']));

// Чтение объекта

final player = playersBox.get('currentPlayer');

// Удаление

playersBox.delete('currentPlayer');

Данный метод сохранения данных подходит для задач сохранения информации о игровом процессе благодаря возможности сохранять кастомные классы через адаптеры. Но есть в нем и недостатки, это отсутствие поддержки какого либо языка запросов.

# Описывается основная суть вашей работы, преимущества, которые получит организация и пользователи системы после модификации системы

Дается ссылка на приложение 1, в котором будет ваше техническое задание

# Заключение

# Список использованных источников

Никаких интернет-источников, только ссылки на литературу библиотеки, нормативные документы организации или живые книги. Интернет-источники можете указать в качестве дополнительных

# Приложения