ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»
ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И ТЕХНОЛОГИЙ
ВЫСШАЯ ШКОЛА ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ И СУПЕРКОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Отчет о прохождении производственной технологической (проектно-технологической) практики на тему: «Разработка системы учета рабочего времени для сотрудников»

Родионова Ильи Алексеевича, гр. 3530903/90301

Направление подготовки: 09.03.03 Прикладная информатика.
Место прохождения практики: СПбПУ, ИКНТ, ВШИСиСТ.
Сроки практики: с 11.06.2022 по 09.07.2022.
Руководитель практики от ФГАОУ ВО «СПбПУ»: Туральчук Константин Анатольевич, к. т. н., доцент ВШИСиСТ.
Консультант практики от ФГАОУ ВО «СПбПУ»: Пархоменко Владимир Андреевич, ассистент ВШИСиСТ.
Руководитель практической подготовки от профильной организации: Карпушин Василий Борисович, технический директор Rem&Coil.
Оценка: _____

Руководитель практической подготовки от ФГАОУ ВО «СПбПУ»	Туральчук К.А
Консультант практической подготовки от ФГАОУ ВО «СПбПУ»	Пархоменко В.А.
Руководитель практической подготовки от профильной организации:	Карпушин В.Б.
Обучающийся	И.А. Родионов

Дата: 09.07.2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В качестве темы практической работы была предложена разработка системы учета рабочего времени для сотрудников *Rem&Coil*. Сотрудники технического отдела данной компании одновременно работают над несколькими проектами и переключаются с одного проекта на другой. На данный момент вычисление затраченного времени на каждый проект производится вручную в конце месяца. Поэтому была предложена идея разработать систему которая позволит централизовать и автоматизировать сбор необходимой информации.

Актуальность работы заключается в том, что данная система позволит производить анализ продуктивности коллектива технического отдела на основе собранных данных. Также, она освободит руководителя от необходимости самостоятельно собирать информацию и производить расчеты.

Цель работы разработать систему (которая включает в себя серверное и web приложения) учета рабочего времени для сотрудников.

Задачи работы:

A. Изучить технологии, позволяющие разрабатывать серверные и web приложения.
B. Разработать приложения, которые необходимы для корректной работы системы.
C. Развернуть разработанную систему.
D. Произвести ручное тестирование всего функционала.

Предполагаемые результаты:

A. Обзор изученных технологий и библиотек;
B. Разработка и внедрение системы;
C. Результаты тестирования функционал.

ГЛАВА 1. ОПИСАНИЕ ОБЩЕЙ АРХИТЕКТУРЫ СИСТЕМЫ

Согласно «Чистой Архитектуре» Роберта Мартина [1], основная архитектура системы не должна зависеть от конкретных технологий (а скорее наоборот, технологии должны служить для реализации выбранной архитектуры), поэтому начать проектирование я решил именно с неё. Это позволит сразу определиться с форматом общения приложений между собой, а так же сформировать набор технологий которые позволят реализовать выбранную архитектуру.

1.1. Описание предложенного задания

Руководитель технического отдела сформулировал следующие критерии, которым должна соответствовать разработанная система:

1 Для взаимодействия с системой необходим веб-интерфейс, который должен быть адаптирован как для мобильных телефонов, так и для экрана монитора.
2 Поддерживаться две роли: администратор и обычный пользователь.
3 Админ создает проекты или архивирует их. На данном этапе для проектов храниться только название. Проекты доступны всем существующим пользователям (в том числе и администратору).
4 На странице проектов для каждого из них доступны следующие операции: пуск/пауза.
5 По нажатию на Пуск начинается отсчет времени. Если он уже был запущен ранее - он продолжается. Одновременно для одного пользователя может идти отсчет только по одному проекту. При активации проекта другой запущенный встает на паузу.
6 Должен быть экран с статистикой по текущему месяцу. Статистика должна отображаться как по проекту в целом, так и по пользователям отдельно. Пример текущего формата вывода статистики представлен на рис.1.1 и рис.1.2

Наименование	%	% кор
РЭМ	22	23
НЭМ	0	0
ПЭМ	0	0
ИС2000М	23	23
ГП4000	49	49
СФ2000М	5	5
	99	77

Рис.1.1. Пример текущего формата общей статистики

Василий								
Дата	День	РЭМ	НЭМ	ПЭМ	ИС2000М	ГП4000	СФ2000М	Check
01.дек	Вт	1			1	6		8
02.дек	Ср	3			1	4		8
03.дек	Чт	2			3	3		8
04.дек	Пт	2			3	3		8
05.дек	Сб							0
06.дек	Вс							0
07.дек	Пн	1			1	6		8
08.дек	Вт	3			1	4		8
09.дек	Ср	2			3	3		8
10.дек	Чт	2			3	3		8
11.дек	Пт	1			3	4		8
12.дек	Сб							0
13.дек	Вс							0
14.дек	Пн	1			3	3	1	8
15.дек	Вт	3			2	2	1	8
16.дек	Ср	1			2	4	1	8
17.дек	Чт	4				4		8
18.дек	Пт	1			2	4	1	8
19.дек	Сб							0
20.дек	Вс							0
21.дек	Пн	1			1	5	1	8
22.дек	Вт				1	5	2	8
23.дек	Ср							0
24.дек	Чт							0
25.дек	Пт							0
26.дек	Сб							0
27.дек	Вс							0
28.дек	Пн							0
29.дек	Вт							0
30.дек	Ср							0
31.дек	Чт							0
	ВСЕГО	28	0	0	30	63	7	128
		РЭМ	НЭМ	ПЭМ	ИС2000М	ГП4000	СФ2000М	

Рис.1.2. Пример текущего формата общей статистики

1.2. Описание предполагаемой архитектуры

После анализа предложенного задания и исходя из предметной области, можно сделать вывод, что наиболее предпочтительной архитектурой системы является архитектура *«Клиент — сервер»*.

Архитектура «Клиент — сервер» - сборное понятие, состоящее из двух взаимодополняющих компонентов: сервера и клиента.

Клиент — это программа, с которой работает пользователь. Он работает в браузере или с desktop-приложением. В разрабатываемой системе в качестве клиента будет использоваться веб-интерфейс.

Сервер - это компьютер, на котором хранится само приложение. Весь код, вся логика, все дополнительные материалы и справочники. Так же сервер обычно разделяют на две сущности: сервер с основной логикой приложения и сервер с базой данных.

Простейшая схема клиент-серверной архитектуры представлен на рисунке рис.1.3

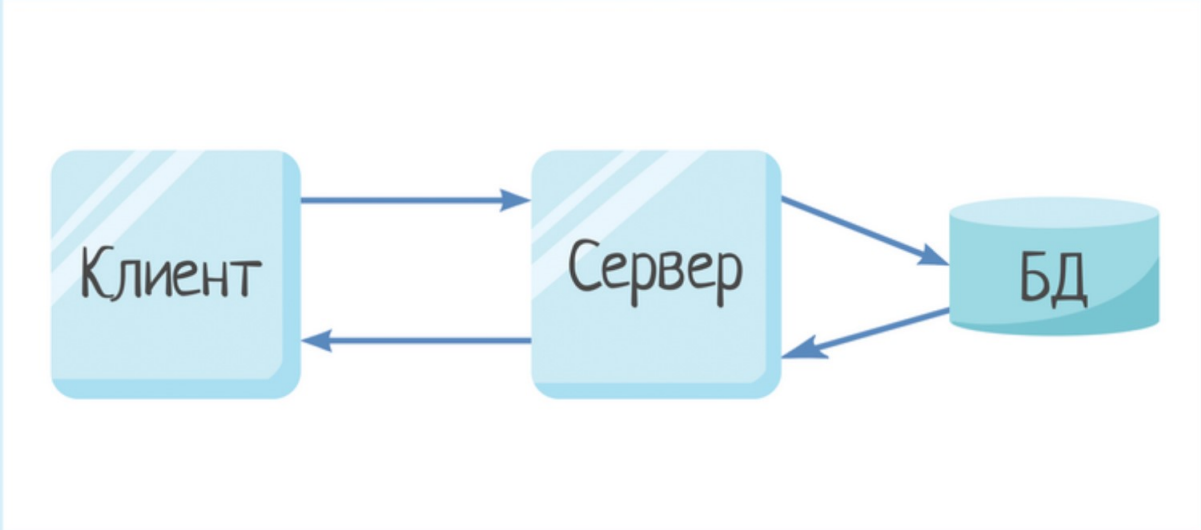

Рис.1.3. Архитектура «Клиент — сервер»

Основной причиной использования данной архитектуры является **централизация обработки входящих запросов**: запросы от всех клиентов обрабатываются и хранятся в одном месте, что позволяет упросить синхронизацию состояний между клиентами.

В клиент-серверной архитектуре клиент и сервер находятся на разных компьютерах и являются разными приложениями, поэтому им необходим интерфейс для общения. Наиболее популярным на данный момент *RESTful API*.

RESTful API[2] — это интерфейс, используемые двумя компьютерными системами для безопасного обмена информацией через Интернет. Он определяет правила, которым необходимо следовать для связи с другими программными системами. Разработчики внедряют или создают API-интерфейсы, чтобы другие приложения могли программно взаимодействовать с их приложениями.

Основной концепцией REST является то, что все данные представляются в виде ресурсов , которые хранятся в определенных местах. Клиент связывается с сервером с помощью API, когда ему требуется какой-либо ресурс.

Основные этапы запроса REST API:

1 Клиент отправляет запрос на сервер. Руководствуясь документацией API, клиент форматирует запрос таким образом, чтобы его понимал сервер.

2 Сервер аутентифицирует клиента и подтверждает, что клиент имеет право сделать этот запрос.

3 Сервер получает запрос и внутренне обрабатывает его.

4 Сервер возвращает ответ клиенту. Ответ содержит информацию, которая сообщает клиенту, был ли запрос успешным. Также запрос включает сведения, запрошенные клиентом.

Традиционно, для передачи запросов от клиента к серверу используется протокол передачи гипертекста (*HTTP*). В качестве адреса доступа к ресурсу используется URL. Также, протокол HTTP поддерживает большой набор методов обращения к серверу, наиболее используемые это GET, POST, PUT, DELETE. Метод HTTP сообщает серверу, что ему необходимо сделать с ресурсом. Например, если метод у запроса GET то серверу необходимо вернуть информацию о данном ресурсе, а если DELETE - то удалить данный ресурс.

После обработки пришедшего запроса сервер должен сформировать ответ. Каждый ответ сервера должен содержать код ответа, по которому клиент понимает результат выполнения запроса. Например, коды 2XX указывают на успешное выполнение, а коды 4XX и 5XX — на ошибки. Также большинство ответов сервера могут содержать тело ответа, в котором храниться ответ сервера, например определенное представление ресурса.

Так как клиент и сервер должны пересылать представления ресурсов, необходимо выбрать определенный формат этого представления. В качестве такого формата я выбрал JSON, ввиду его простоты и гибкости. На рис.1.4 представлен пример ресурса в формате JSON.

```
[
  {
    "Id": 0,
    "FirstName": "string",
    "LastName": "string",
    "Name": "string",
    "EmailAddress": "string",
    "TerritoryId": 0
  }
]
```

Рис.1.4. JSON

Я выбрал RESTful API как основной интерфейс обращения к серверу в силу следующих его преимуществ:

– Системы, реализующие REST API, могут эффективно масштабироваться благодаря оптимизации взаимодействия между сервером и клиентом по REST.

- Веб-службы RESTful поддерживают полное разделение клиента и сервера. Они упрощают и разделяют различные серверные компоненты, чтобы каждая часть могла развиваться независимо. Изменения платформы или технологии в серверном приложении не влияют на клиентское приложение. Возможность разделения функций приложения на уровни еще больше повышает гибкость. Например, разработчики могут вносить изменения в уровень базы данных, не переписывая логику приложения.
- REST API не зависит от используемой технологии. Вы можете создавать как клиентские, так и серверные приложения на разных языках программирования, не затрагивая структуру API. Также можно изменить базовую технологию на любой стороне, не влияя на обмен данными.

1.3. Резюме

После анализа поставленной задачи, я решил разбить систему на два приложения: клиент и сервер. В качестве протокола взаимодействия между ними был выбран протокол HTTP, а также в качестве интерфейса обращения к серверному приложению был выбран RESTful API.

Данная архитектура простая в реализации, а так же позволяет в будущем совершенствовать и масштабировать разработанную систему.

ГЛАВА 2. ОБЗОР ТЕХНОЛОГИЙ

Как уже было сказано в 1.2 в системе будет два приложения: клиентское и серверное. Поэтому данная глава тоже будет разделена на два пункта: технологии для разработки клиентского приложения и для серверного.

2.1. Обзор технологий разработки серверных приложений]

2.1.1. Выбор языка программирования

Прежде чем приступить к рассмотрению технологий, необходимо определиться с языком программирования. Так как почти для каждого языка программирования есть библиотеки или фреймворки, поэтому при выборе языка необходимо я решил основываться на следующих критериях:

- Популярность. Если у языка крупное сообщество программистов, то для него написано большое количество библиотек, а также будет проще найти ответ на форуме на возникающие вопросы, например при возникновении ошибки.
- Строгая/статическая типизация. У языков со строгой типизацией на этапе компиляции происходит проверка типов, и поэтому это позволяет избежать множества ошибок, связанных с типизацией, а так же ускоряет разработку.
- Простота. Так же сильно ускоряет обучение и разработку приложения.

Основываясь на один из самых известных рейтингов языков программирования *«TIOBE Index»*[7] можно выделить 5 самых популярных языков: Python, C, Java, C++, C#. Из данного списка под упомянутые ранее критерии подходят два языка: Java и C#. Ввиду того, что с языком **Java** я знаком уже более 4 лет, мой выбор пал на него.

2.1.2. Выбор основного фреймворка

Определившись с языком программирования, можно перейти к выбору основной технологии. В мире Java самым популярным фреймворком является уже более 10 лет **Spring**[6]. Основные преимущества данной технологии:

- Популярность. Данной технологии доверяют множество компаний по всему миру, например Яндекс или Сбербанк.

– Гибкость. Гибкий и всеобъемлющий набор расширений и сторонних библиотек Spring позволяет разработчикам создавать практически любые приложения, которые только можно вообразить.
– Большая экосистема. Сам по себе Spring содержит только базовые функции. Остальные вы можете подключить найдя соответствующею библиотеку, например для работы с базой данных или управлением безопасностью. Из этого следует то, что вы можете подключить только то, что вам действительно необходимо.

Основой Spring являются две принципа: Inversion of Control (IoC) and Dependency Injection (DI).

Inversion of Control (IoC) - это принцип состоит в том что большую часть объектов создаем не мы, а Spring. Мы лишь конфигурируем классы (с помощью аннотаций либо в конфигурационном XML), чтобы «объяснить» фреймворку Spring, какие именно объекты он должен создать за нас, и полями каких объектов их сделать. Spring управляет созданием объектов и потому его контейнер называется IoC-контейнер. А объекты, которые создаются контейнером и находятся под его управлением, называются бинами. Иллюстрация данного принципа представлена на рис.2.1.

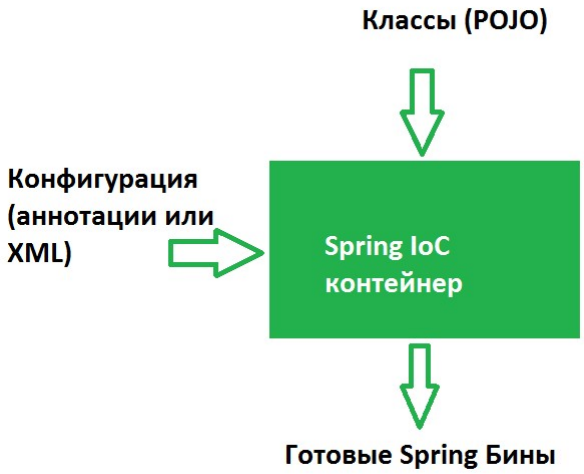

Рис.2.1. Схема Inversion of Control

Dependency Injection (DI) - процесс предоставления внешней зависимости программному компоненту. Это означает что объект не сам создает другие объекты, необходимые для его работы, а получает их снаружи, например через конструктор класса.

На основе этих двух принципов и построена вся экосистема библиотек фреймворка.

2.1.3. Выбор дополнительных библиотек

Как было сказано выше, фреймворк Spring позволяет подключить множество библиотек, поэтому необходимо было определиться с библиотеками, которые будут подключены дополнительно. Исходя из задания, можно было определить следующие библиотеки:

- Для упрощения конфигурации и запуска приложения - Spring Boot.
- Для обработки входящих запросов - Spring Web.
- Для работы с базой данных - Spring Data Jpa. (а также flyway для применения миграции баз данных)
- Для работы с безопасностью (ограничение доступа) - Spring Security. (а io.jsonwebtoken.jwt для формирования JWT токенов).
- Для автогенерации спецификации OpenAPI - SpringDoc.

Рассмотрим основные.

Spring Boot - это проект, целью которого является упрощение создания приложений на основе Spring. Он позволяет наиболее простым способом создать web-приложение, требуя от разработчиков минимум усилий по его настройке и написанию кода. Основными его преимуществами являются: простота управления зависимостями (существуют специальные starter библиотеки для более простой интеграции с Spring Boot) и автоматическая конфигурация (конфигурация всего приложения находится в специальном файле: /resources/application.yml, где описываются все необходимые параметры, например подключение к базе данных).

Spring Web - это библиотека из экосистемы Spring, которая добавляет все необходимое для обработки входящих HTTP запросов. Для этого в библиотеке есть специальные аннотации. На рис.2.2 представлен пример метода, обрабатывающего GET запрос.

```
@GetMapping
public @ResponseBody List<FullUser> getAll() {
    List<User> users = userService.getAll();
    return users.stream().map(FullUser::build).collect(Collectors.toList());
}
```

Рис.2.2. Пример обработчика входящего запроса

В данном примере наша функция принимает входящий запрос, выполняет какую-то бизнес логику, и отдает определенные модели. Остальную работу делает за нас Spring: он настраивает сервер, чтобы после приема запроса он отдавался нужной функции, самостоятельно превращает модели полученные из функции в JSON объекты и формирует ответ клиенту.

Также библиотека позволяет проверять заголовки запроса, получать тело, настраивать динамические URL и обрабатывать их, оправлять настраиваемые коды ответов и многое другое. Все это описано в официальной документации[6].

Spring Data Jpa - это библиотека предоставляющая необходимые абстракции для работы с базой данных, из состава Spring. Основной принцип этой библиотеки заключается в том, что мы таблицы базы данных представляем как обычные классы.

Необходимые шаги:

1 Создаем сущность. Для этого надо создать класс (имя совпадает с именем таблицы в БД) и пометить его аннотацией *Entity*. Поля описанные в данном классе должны совпадать с колонками в базе.

2 Наследоваться от одного из интерфейсов Spring Data, например от CrudRepository. Данный интерфейс представляет базовые методы для работы с базой (есть и другие интерфейсы). Если нет необходимого методы, запросы к сущности можно строить прямо из имени метода. Для этого используется механизм префиксов find. . . By, read. . . By, и так далее, далее от префикса метода начинает разбор остальной части.

3 Далее уже можно использовать интерфейс в бизнес логике. Spring сам создаст необходимый класс со всеми реализациями и передаст его туда, где используется интерфейс.

2.2. Обзор технологий разработки клиентских приложений

2.2.1. Обзор основного фреймворка

В качестве основного фреймворка я решил выбрать *Flutter*[5]. Это технология разработанная компанией Google, позволяющая разрабатывать мультиплатформенные приложения. На данный момент она поддерживает 6 платформ: Web, Android, IOS, Windows, Linux, MacOS. Как раз из-за этой особенности Flutter и был выбран как главная технология.

В качестве языка программирования используется Dart. Данный язык задумывался как типизированная замена JavaScript но в веб среде язык не взлетел, но Google придумали ему другое применение.

Преимущества FLutter:

- Быстрый. Код Flutter компилируется в машинный код ARM или Intel, а также в JavaScript для быстрой работы на любом устройстве.
- Поддерживает горячую перезагрузку. Данная возможность позволяет перекомпилировать только те участки кода, которые поменялись и не перезапускать приложение, что в десятки раз увеличивает скорость разработки.
- Гибкий. Flutter позволяет управлять каждым пикселем, чтобы создавать индивидуальные адаптивные дизайны. А так же он позволяет использовать особенности платформы, например работать с камерой.
- Open Source. Код Flutter распространяется под меткой Open Source, что означает что он полностью открытый и каждый желающий может помочь в развитии продукта.

Весь пользовательский интерфейс во Flutter строиться из так называемых виджетов, которые составляются в дерево. Все кнопки, тексты, картинки и прочее, что видит пользователь, является виджетом. Видимые виджеты разделяются две категории: stateless и statefull.

Stateless виджеты, как понятно из названия, не имеют состояния. Они нужны только для отображения какой либо части пользовательского интерфейса. Из-за отсутствия состояния, эти виджеты намного легче и производительнее.

Statefull виджеты наоборот содержат состояние, и позволяют вызывать обновление и перересовку экрана по изменению состояния. Во время перерисовки, обновляется как сам виджет, так и все его потомки. Для изменения состояния используется метод setState().

Чтобы создать свой виджет, необходимо создать класс и унаследовать его от базового класс (либо StatelessWidget, либо StatefullWidget). Также обязательно необходимо переопределить метод build(BuildContext context) который и отвечает за отрисовку. Данный метод вызывается на каждый отрисованный кадр. На рис.2.3 представлен пример простейшего виджета.

В данном примере отрисовывается экран на котором представлен текст по центру и кнопка. По нажатию кнопки происходит смена текста и перерисовка экрана.

```dart
class SampleAppPage extends StatefulWidget {
  SampleAppPage({Key? key}) : super(key: key);

  @override
  _SampleAppPageState createState() => _SampleAppPageState();
}

class _SampleAppPageState extends State<SampleAppPage> {
  // Default placeholder text.
  String textToShow = 'I Like Flutter';

  void _updateText() {
    setState(() {
      // Update the text.
      textToShow = 'Flutter is Awesome!';
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Sample App'),
      ),
      body: Center(child: Text(textToShow)),
      floatingActionButton: FloatingActionButton(
        onPressed: _updateText,
        tooltip: 'Update Text',
        child: Icon(Icons.update),
      ),
    );
  }
}
```

Рис.2.3. Пример виджета

2.2.2. Обзор дополнительных библиотек

Приложения на Flutter можно разрабатывать и без каких либо библиотек, но для удобства, гибкости и скорости разработки необходимо добавить некоторые библиотеки.

Bloc[3] - это шаблон проектирования, компонент бизнес-логики, отвечающий за управление состоянием. Этот шаблон проектирования помогает отделить представление от бизнес-логики. Следование шаблону BLoC облегчает тестирование и повторное использование. Этот пакет абстрагирует реактивные аспекты шаблона, позволяя разработчикам сосредоточиться на написании бизнес-логики. На рис.2.4 представлена схема взаимодействия с блоком.

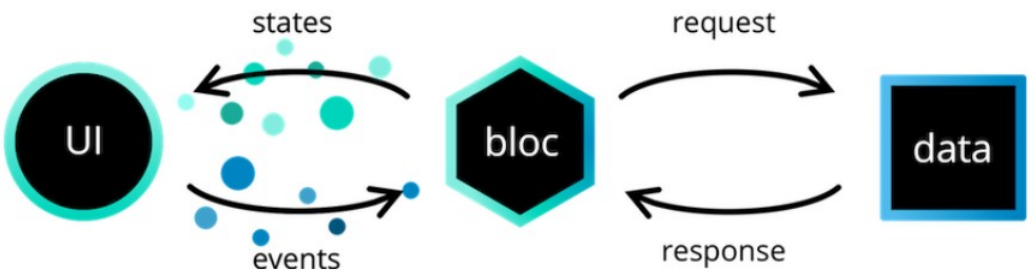

Рис.2.4. Bloc

Основная концепция данного шаблона в том, что создается класс у которого есть два потока: событий и состояний. После определенного действия (например пользователь нажал на кнопку) в блок добавляется новые событие, и блок начинает его обработку. Обработкой события и является бизнес логика. После необходимых действий, блок выдает наружу через соответствующий поток новое состояние которое отображает виджет. Пример простейшего блока представлен на рис.2.5.

```dart
/// The events which `CounterBloc` will react to.
abstract class CounterEvent {}

/// Notifies bloc to increment state.
class CounterIncrementPressed extends CounterEvent {}

/// A `CounterBloc` which handles converting `CounterEvent`s into `int`s.
class CounterBloc extends Bloc<CounterEvent, int> {
  /// The initial state of the `CounterBloc` is 0.
  CounterBloc() : super(0) {
    /// When a `CounterIncrementPressed` event is added,
    /// the current `state` of the bloc is accessed via the `state` property
    /// and a new state is emitted via `emit`.
    on<CounterIncrementPressed>((event, emit) => emit(state + 1));
  }
}
```

Рис.2.5. Пример Bloc

Так же в библиотеке поставляются специальные виджеты, которые упрощают работу с блоком. Так например, есть виджет BlocBuilder позволяющий обновлять экран на каждое изменение состояния блока. Про остальные виджеты можно прочитать в официальной документации[4]

2.3. Выводы

В данной главе мной были рассмотрены технологии и библиотеки которые понадобятся для разработки системы. Итоговый список получился следующий:

- Серверное приложение
    A. Java
    B. Spring, Spring Boot, Spring Web, SpringDoc.
    C. Spring Data Jpa, flyway, Spring Security.
- Клиентское приложение
    A. Dart
    B. Flutter
    C. Bloc, Flutter_Bloc

ГЛАВА 3. РАЗРАБОТКА ПРОГРАММНОГО КОДА

3.1. Название параграфа

3.2. Название параграфа

3.3. Выводы

ГЛАВА 4. РУЧНОЕ ТЕСТИРОВАНИЕ СИСТЕМЫ

4.1. Название параграфа

4.2. Название параграфа

4.3. Выводы

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Мартин Р.* Чистая архитектура. — Питер, 2021. — 352 с.

2. Что такое RESTful API? — URL: https://aws.amazon.com/ru/what-is/restful-api/ (дата обращения: 28.06.2022).

3. Documentation for bloc. — URL: https://pub.dev/packages/bloc (visited on 28.06.2022).

4. Documentation for flutter bloc. — URL: https://pub.dev/packages/flutter_bloc (visited on 28.06.2022).

5. Flutter official site. — URL: https://flutter.dev/ (visited on 28.06.2022).

6. Spring official site. — URL: https://spring.io/ (visited on 28.06.2022).

7. TIOBE Index. — URL: https://www.tiobe.com/tiobe-index/ (visited on 28.06.2022).

Приложение 1

UML диаграммы

Приложение 2

Разработанный код