Федеральное государственное автономное
Образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Гуманитарный институт
институт

информационных технологий в креативных и культурных индустриях
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ А. В. Усачев
«____»_____ 2022 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.03.14 «Прикладная информатика в области искусства и гуманитарных наук»
код – наименование направления

Использование программного интерфейса приложения (API)) для
Тема

исследования коллекций цифровой библиотеки «Европеана»

Руководитель	_____	ст. преподаватель	Е.Р. Брюханова
	подпись, дата	должность, ученая степень	инициалы, фамилия
Консультант	_____	ст. преподаватель	И.А. Кижнер
	подпись, дата	должность, ученая степень	инициалы, фамилия
Выпускник	_____		Д.М. Изосимов
	подпись, дата		инициалы, фамилия

Красноярск 2022

Продолжение титульного листа бакалаврской работы по теме Использование программного интерфейса приложения (API) для исследования коллекций цифровой библиотеки Europeana

Нормоконтролер _____ Е. Р. Брюханова
подпись, дата

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На сегодняшний день в разных сферах научной деятельности используются большие массивы данных. Одним из крупных агрегаторов таких данных является Europeana.

Europeana – это цифровая библиотека, содержащая более 61 миллиона оцифрованных экспонатов. Среди них 36 миллионов составляют изображения. Такое огромное количество данных, а также системы поиска и фильтрации потенциально позволяют использовать Europeana как ресурс для гуманитарных исследований.

Однако для того, чтобы результаты исследований на базе Europeana были корректными и достоверными, необходимо иметь возможность проводить анализ по всем данным. Проблема исследования заключается в том, что Europeana, не предоставляет пользователям возможности проводить выборку по некоторым параметрам.

Выпускная квалификационная работа посвящена исследованию программного интерфейса приложения цифровой библиотеки Europeana.

Актуальность настоящей работы заключается в необходимости иметь возможность проводить анализ на основе всех данных, это влияет на качество исследований и наши представления о мировой культуре.

Объект исследования – программный интерфейс приложения (API).

Предмет исследования – принципы использования программного интерфейса приложения (API) для исследования коллекций цифровой библиотеки Europeana.

Цель настоящей работы – выявить принципы использования программного интерфейса приложения (API) для исследования коллекций цифровой библиотеки Europeana.

Задачи исследования:

- провести анализ документации Europeana;
- изучить понятие API;
- найти способы получения данных через API;
- спроектировать и реализовать веб-сервис для количественного анализа полученных данных.

Научная новизна данного исследования заключается в отсутствии сервисов для количественного анализа Europeana.

Существование сервиса, позволяющего проводить анализ по всем данным, позволило бы проводить различные исследования для широкого спектра гуманитарных задач.

Выпускная квалификационная работа состоит из введения, трех глав, заключения, списка использованной литературы.

Первая глава посвящена анализу документации Europeana, определению понятия API, изучению работы Europeana API.

Вторая глава «Разработка веб-сервиса» описывает процесс разработки.

Третья глава «Результаты исследования» включает описание реализованного веб-сервиса и демонстрацию его работы.

Заключение подводит итоги исследования, список литературы содержит библиографические описания цитируемых источников, в приложении представлены примеры собранных данных в виде таблиц, диаграмм, карт и прочих изображений.

1 API как инструмент исследования

1.1 Понятие API и его задачи

Программный интерфейс приложения (Application Programming Interface, API) – это набор способов и правил, по которым различные программы общаются между собой и обмениваются данными. Все эти коммуникации происходят с помощью функций, классов, методов, структур, а иногда констант одной программы, к которым могут обращаться другие [1].

API — это набор определенных правил, которые объясняют, как компьютеры или приложения взаимодействуют друг с другом. API находятся между приложением и веб-сервером, выступая в качестве промежуточного уровня, который обрабатывает передачу данных между системами.

Механизм работы API:

Клиентское приложение инициирует вызов API для получения информации, также известный как запрос. Этот запрос обрабатывается от приложения к веб-серверу через универсальный код ресурса (URI) API и включает в себя команду запроса, заголовки, а иногда и текст запроса.

После получения действительного запроса API делает вызов внешней программе или веб-серверу, после чего сервер отправляет API ответ с запрошенной информацией. Затем API передает данные исходному запрашивающему приложению.

Хотя передача данных будет отличаться в зависимости от используемой веб-службы, этот процесс запросов и ответов происходит через API. В то время как пользовательский интерфейс предназначен для использования людьми, API предназначены для использования компьютером или приложением.

1.1.1 Типы API

В настоящее время большинство интерфейсов прикладного программирования представляют собой веб-API, которые предоставляют данные и функции приложения через Интернет. Вот четыре основных типа веб-API:

Открытые API — это интерфейсы прикладного программирования с открытым исходным кодом, к которым можно получить доступ с помощью протокола HTTP. Также известные как общедоступные API, они определяют конечные точки API и форматы запросов и ответов.

Партнерские API — это интерфейсы прикладного программирования, предоставляемые или предоставляемые стратегическими деловыми партнерами. Как правило, разработчики могут получить доступ к этим API в режиме самообслуживания через общедоступный портал разработчика API. Тем не менее, им нужно будет завершить процесс адаптации и получить учетные данные для входа в систему для доступа к партнерским API.

Внутренние API — это интерфейсы прикладного программирования, которые остаются скрытыми от внешних пользователей. Эти частные API недоступны для пользователей за пределами компании и вместо этого предназначены для повышения производительности и коммуникации между различными внутренними командами разработчиков.

Составные API объединяют несколько API данных или служб. Эти службы позволяют разработчикам получать доступ к нескольким конечным точкам за один вызов. Составные API полезны в архитектуре микрослужб, где для выполнения одной задачи может потребоваться информация из нескольких источников.

1.1.2 Типы протоколов API

По мере расширения использования веб-API были разработаны определенные протоколы, предоставляющие пользователям набор определенных правил, определяющих принимаемые типы данных и команды. По сути, эти протоколы API облегчают стандартизированный обмен информацией:

SOAP (Simple Object Access Protocol) — это протокол API, построенный на основе XML, позволяющий пользователям отправлять и получать данные через SMTP и HTTP. С помощью API SOAP проще обмениваться информацией между приложениями или программными компонентами, работающими в разных средах или написанными на разных языках.

XML-RPC — это протокол, который использует определенный формат XML для передачи данных, в то время как SOAP использует проприетарный формат XML. XML-RPC старше, чем SOAP, но намного проще и относительно легок в том смысле, что использует минимальную пропускную способность.

JSON-RPC - это протокол, похожий на XML-RPC, поскольку они оба являются удаленными вызовами процедур (RPC), но этот использует JSON вместо формата XML для передачи данных. Оба протокола просты. Хотя вызовы могут содержать несколько параметров, они ожидают только одного результата.

REST (Representational State Transfer) — это набор принципов архитектуры веб-API, что означает, что нет официальных стандартов (в отличие от стандартов с протоколом). Чтобы быть REST API (также известным как RESTful API), интерфейс должен соответствовать определенным архитектурным ограничениям. Можно создавать RESTful API с протоколами SOAP, но эти два стандарта обычно рассматриваются как конкурирующие спецификации.

1.2 Цифровая библиотека Europeana

Europeana [2] – электронная библиотека, цель которой – обеспечить доступ к оцифрованным коллекциям и экспонатам культурного наследия, находящиеся в разных культурных организациях: в музеях, архивах, библиотеках и т.д.

Экспонаты могут представлять собой отсканированные страницы книг, произведения искусства, предметы материальной культуры, фото, видео, звукозаписи и многое другое. Данный ресурс позволяет ознакомиться и изучить миллионы экспонатов из множества культурных учреждений разных стран мира. Главная страница цифровой библиотеки представлена на Рисунке 1.

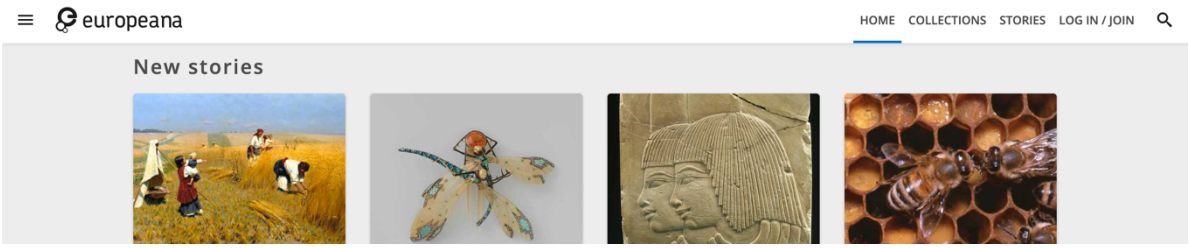


Рисунок 1 – Главная страница. Интерфейс веб-ресурса Europeana

Europeana была основана в 2008 году. На момент старта проекта в ней было представлено около 2 миллионов оцифрованных экспонатов, на состояние 2022 года библиотека содержит более 61 милллиона оцифрованных экспонатов из разных областей культурного наследия.

Своей миссией сотрудники проекта Europeana называют облегчение использования культурного наследия для образования, научных исследований, творчества и отдыха, что способствует развитию открытого и творческого общества. [3]

Проект Europeana стремится дать пользователю возможность организовать информацию по разным параметрам: поиск по сайту позволяет отсортировывать экспонаты по типу носителя (изображение, текст, видео, звукозапись или 3D), языку, организации, а также возможности дальнейшего использования. Также

Europeana содержит несколько крупных тематических коллекций, посвященных искусству, моде, музыке, Первой мировой войне, археологии и т.д.

Визуальный контент занимает в Europeana более половины всего объема экспонатов. Из 61 миллиона 36 миллиона экспонатов являются изображениями. Опираясь на удобную систему поиска и фильтрации, потенциально проект

Europeana уделяет много внимания стандартизации коллекций, размещенных на ее страницах. Для того, чтобы разместить свою коллекцию в Europeana, культурному учреждению необходимо привести оцифрованные экспонаты к принятому стандарту. [4] В эти стандарты входят формат метаданных, лицензирование коллекций для дальнейшего использования, доступность коллекции в Сети и соблюдение необходимого объема коллекций, в соответствии с контентной стратегией Europeana. [5]

Помимо этого, Europeana уделяет внимание авторским правам на экспонаты и их метаданные. В систему лицензирования Europeana включено 14 вариантов лицензий, в которые включены и лицензии Creative Commons. Обязательным условием является то, что все метаданные коллекций должны быть доступны для повторного использования без ограничений.

1.2.1 Документация по использованию Europeana API

API Europeana позволяет создавать приложения, которые используют данные коллекций, взятых из крупных музеев и галерей по всей Европе. На главной странице Europeana pro предлагается ознакомиться в Пользовательским соглашением и получить API ключ для дальнейшей работы Рисунок 2, Рисунок 3.

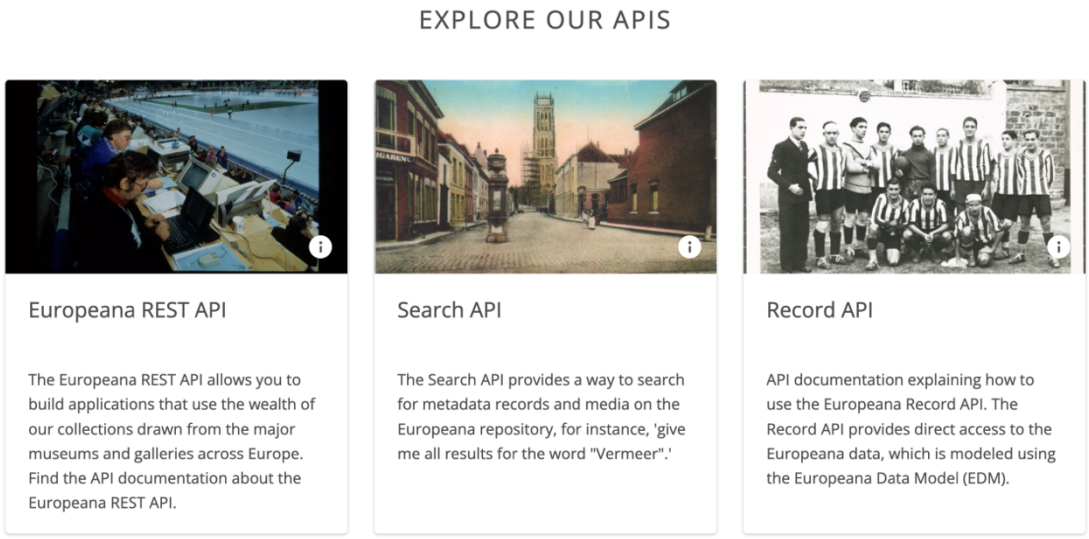


Рисунок 2 Главная страница. Интерфейс веб-ресурса Europeana pro

Рисунок 3 Главная страница. Подключение API ключа, Пользовательское соглашение.

Условия использования Europeana API:

Все Метаданные, доступные на europeana.eu, публикуются без ограничений в соответствии с условиями Creative Commons CC0 1.0 Universal Public Domain Dedication. Текст лицензии – «Лицо, связавшее произведение с этим документом, сделало произведение общественным достоянием, отказавшись от всех своих прав на произведение во всем мире в соответствии с законом об авторском праве, включая все смежные права, в той мере, в какой это разрешено законом. Вы можете копировать, изменять, распространять и выполнять работу даже в коммерческих целях, не спрашивая разрешения.»

Получая доступ к Europeana API, Пользователь принимает и соглашается соблюдать настоящие Условия использования и Политику конфиденциальности и файлов cookie Europeana. Кроме того, Пользователь должен, прежде чем использовать Europeana API, принять настоящие Условия использования, если эта опция доступна в форме регистрации для API.

API-ключи являются конфиденциальными и предназначены только для использования Пользователем и не могут быть переданы третьим лицам, кроме как с явного согласия Europeana. API-ключ применяется к одному программному решению. Пользователю необходимо создать отдельный API-ключ для каждого нового Программного решения.

1. Europeana оставляет за собой право прекратить действие отдельных API-ключей Europeana, которые используются для отображения данных, полученных через Europeana API, в контекстах, которые являются незаконными, порнографическими, дискредитирующими или могут нанести ущерб репутации Europeana любым другим способом.

2. Europeana предоставит Пользователю публичную техническую документацию Europeana API, но не будет предоставлять техническую поддержку Пользователям, если явно не оговорено иное.

3. Europeana будет регистрировать все виды использования Europeana API Пользователем с целью мониторинга загрузки сервиса и с целью улучшения Europeana API. Собранная информация будет храниться в течение ограниченного периода времени.

4. Europeana может ограничить количество запросов, которые Пользователь может отправлять и получать через Europeana API. Такое ограничение должно быть разумным и может быть установлено Europeana в любое время для обеспечения равномерной работы Europeana API для всех Пользователей.

5. Europeana будет постоянно обновлять Europeana API, добавляя новые функции. Europeana будет своевременно уведомлять Пользователя об обновлениях и будет стараться поддерживать обратную совместимость новых

версий Europeana API. Тем не менее, Europeana не гарантирует, что Программное решение Пользователя будет продолжать функционировать после обновления.

6. Пользователям API предлагается использовать логотип «Сделано с Помощью Europeana» в непосредственной близости от Метаданных всякий раз, когда такие Метаданные отображаются, чтобы указать, что они получены из API Europeana.

7. Пользователю рекомендуется следовать Руководству по использованию метаданных Europeana и указывать источники данных, включая, где это возможно, ссылки на Контент на веб-сайтах первоначальных поставщиков данных.

8. В случае, если у Europeana сложится впечатление, что она публикует метаданные или миниатюры, на которые она не имеет права, она удалит эти метаданные или миниатюры из API Europeana. Пользователь должен прекратить использование таких Метаданных и Миниатюр по первому запросу от Europeana.

1.2.2 Europeana API поиска

API поиска предоставляет способ поиска записей метаданных и мультимедиа в репозитории Europeana, например, дает все результаты по слову «Vermeer». Помимо возможности прямого поиска на Europeana, этот API также предоставляет вспомогательный метод для перевода запросов и поддержку протокола OpenSearch.RSS.

API поиска — это самый простой API для использования и понимания. Он взаимодействует с данными Europeana во многом так же, как и веб-сайт Europeana. Дает возможность искать ключевые слова, и API вернет все записи, соответствующие этому ключевому слову. Можно также уточнить свой поиск с помощью более сложных запросов, таких как логические поиски, или

отфильтровать части расширенной фильтрации результатов. Предлагает выбрать возврат только тех объектов, которые имеют определенные заявления об авторских правах, и выбрать возврат результатов на языке по выбору.

1. Начало работы.

Каждый вызов API поиска является HTTPS-запросом в следующей подписи URL-адреса: https://api.europeana.eu/record/v2/search.json

2. Ответ

Ответ на API поиска всегда форматируется в JSON и будет содержать ряд полей, представляющих информацию об обработке запроса, в то время как конкретная информация о записи представлена в поле «элементы».

3. Ответы на ошибки

Об ошибке, возникающей во время обработки метода API, сообщается (1) соответствующим кодом состояния HTTP, (2) значением поля успеха и (3) значимым сообщением об ошибке в поле ошибки.

Таблица 1 демонстрирует типы ошибок API записей.

Таблица 1 - Типы ошибок.

Код состояния HTTP	**Описание**
200	Запрос выполнен успешно.
401	Учетные данные проверки подлинности отсутствовали или не удалось выполнить проверку подлинности.
429	Запрос может быть обслужен, поскольку приложение достигло предела использования.
500	На сервере произошла ошибка, которая не была должным образом обработана. Если вы получили эту ошибку, это означает, что что-то пошло не так, поэтому, пожалуйста, сообщите нам об этом!

Пример: запрос к API поиска, предоставляющий недопустимый (неизвестный) ключ API

https://api.europeana.eu/record/v2/search.json?query=*&wskey=test


```
{
  "apikey": "test",
```

```
  "success": false,
  "error": "Invalid API key"
}
```

4. Поля поиска вне модели EDM

Модель EDM - это набор основных понятий, которые описывают структуру данных независимо от формы хранения. Модель EDM заимствует свойства модели «сущность-связь»[6].

В дополнение к полям, определенным в EDM, несколько других полей были определены по административным причинам, которые также могут быть использованы для поиска.

5. Поля поиска, зависящие от языка

В модели EDM большинство свойств, принимающих литерал, могут быть помечены тегами языка, что означает наличие языкового кода, определяющего язык текста с использованием стандарта ISO 639-2. Чтобы обеспечить поиск по таким свойствам для конкретного языка, API поиска определяет поле для каждого из языковых вариантов, которые отображаются в нашем репозитории, сохраняя базовое поле со всеми значениями во всех языковых вариациях. В отличие от базового поля, которое обычно имеет тип данных Text (некоторые поля также могут быть определены как String), поля, специфичные для языка, всегда имеют тип String, чтобы обеспечить огранку с полным значением (без токенизации). Если поле, зависящее от языка, является частью набора метаданных, оно также может быть выведено в ответе.

В следующей таблице показаны базовые и языковые поля поиска для свойства dc:creator:

Таблица 2 - Базовые и языковые поля поиска для свойства dc:creator

Поле поиска	**Тип данных поиска**	**Поле результата**
proxy_dc_creator	Text	dcCreator

proxy_dc_creator.en proxy_dc_creator.fr ... proxy_dc_creator.*	String	dcCreatorLangAware

6. Поля поиска, определенные в модели EDM

Модель EDM определяет обширный список классов и свойств. В API поиска только подмножество из них, соответствующее наиболее часто используемым, может использоваться для поиска в репозитории.

7. Агрегированные поля

Europeana агрегирует свои данные от культурных учреждений, которые могут использовать разнообразные, мелкозернистые системы и методологии. В результате связь между, например, объектом и человеком может храниться в различных специализированных областях. Чтобы упростить представление об этих данных, Europeana ввела несколько общих агрегированных полей, таких как: название, кто, что, когда и где. В этих полях мы собираем вместе информацию из разных полей записи, чтобы облегчить обнаружение объектов. Title, например, агрегирует данные из полей dc:title и dcterms:alternative, которые являются частью Dublin Core, популярного общего стандарта для описания различных типов ресурсов.

8. Поиск СМИ

API поиска позволяет не только искать и извлекать метаданные, но и предлагает мощные функции, основанные на технических метаданных. Технические метаданные - это метаданные, которые извлекаются из медиафайлов, таких как изображения и видео, которые связаны с записями, такими как ширина и высота изображения. Эти функции дают возможность искать и фильтровать записи Europeana по информации о мультимедиа, например, искать только записи, которые имеют очень большие изображения, высококачественные аудиофайлы или какие изображения соответствуют определенному цвету. Помимо поиска и фильтрации, огранка также возможна с

использованием технических метаданных и, по сути, является частью аспектов по умолчанию, предоставляемых фасетным профилем. Эти функции были разработаны в рамках Фреймворка повторного использования контента в рамках проекта Europeana Creative.

Запись метаданных Europeana может содержать ссылку на ноль, один или несколько медиафайлов, это означает, что при поиске по свойству или аспекту технических метаданных (например, размеру изображения) запись возвращается, если один из медиафайлов, присутствующих в записи, соответствует поисковому запросу.

9. Цветовая палитра

Из всех записей с изображениями извлекаются шесть наиболее заметных цветов. Затем эти цвета сопоставляются с одним из 120 цветов, которые можно найти в листинге. Для поиска записей, где одно из изображений соответствует определенному цвету, вы можете использовать параметр цветовой палитры, вы можете предоставить его несколько раз. В качестве значения необходимо указать шестнадцатеричный RGB-код, например #8A2BE2 или #FFE4C4.

10. Типы данных для полей поиска

Следующие типы данных определены для полей поиска, используемых для запросов, фильтрации и огранки.

Таблица 3 - Типы данных определены для полей поиска

Тип данных	**Описание**
Boolean	Значение true или false.
Number	Числовое значение, обычно с целочисленной точностью.
Date	Момент времени с миллисекундной точностью. Дополнительные сведения о запросах полей даты см. в разделе X.
String	Значения сохраняются в том виде, в каком они присутствуют в данных, без дополнительной обработки НЛП. Этот тип данных, как правило, более полезен для огранки.
Text	Слово, токенизированное с помощью пунктуационной фильтрации и значения, чувствительного к регистру, с дополнительной стечением слов. Этот тип данных обычно полезен для запросов и фильтрации.

11. Профили

Профиль обычно определяет, насколько обширным будет ответ, либо диктуя поля метаданных, которые будут присутствовать (т.е. минимальные, стандартные и богатые), либо добавляя дополнительные элементы данных, такие как грани или хлебные крошки. Большинство аспектов можно комбинировать, за исключением аспектов метаданных или комбинированных аспектов, таких как rich. В таблице 4 перечислены профили, поддерживаемые API.

Таблица 4 - Профили, поддерживаемые API

Профиль	**Описание**
minimal	Возвращает минимальный набор метаданных.
standard	Возвращает более широкий набор метаданных.
rich	Возвращает самый широкий набор метаданных.
facets	Добавлена информация о гранях. Для записей используется стандартный профиль.
breadcrumbs	Информация о запросе добавляется в виде хлебных крошек. Также добавляются грани; для записей используется стандартный профиль.
params	Заголовок ответа будет содержать ключ params, в котором перечислены запрошенные параметры и параметры по умолчанию вызова API.
portal	Стандартные, грани и панировочные сухари вместе взятые, а также дополнительные поля.

1.3 Языки программирования и библиотеки

PHP — интерпретируемый язык программирования с динамической типизацией. Является языком программирования общего назначения, реализует парадигму объектно-ориентированного программирования. Его основными преимуществами, в рамках разработки системы для количественного анализа, являются:

- простота использования;
- большое количество уже реализованных библиотек и инструментов;
- скорость реализации алгоритмов за счет динамической типизации;

- большое количество реализованных библиотек с открытым исходным кодом для работы с анализом данных.

Также, наиболее весомым аргументом в пользу PHP является то, что все инструменты для разработки непосредственно информационной системы и алгоритмов интеллектуального анализа данных могут быть беспрепятственно интегрированы между собой благодаря единому окружению.

Объектно-ориентированное программирование (ООП) – это методология программирования, основанная на представлении программы в виде совокупности взаимодействующих объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

К плюсам ООП подхода относится:

- скорость исполнения команд и простота их исполнения;
- обеспечение четкой структуры программы;
- упрощение модификации, отладки и сопровождения кода;
- упрощение повторного использования кода.

Также, при разработке были использованы:

- операционная система macOS Monterey;
- symfony — фреймворк для создания веб-сайтов и веб-приложений;
- PhpStorm — кроссплатформенная интегрированная среда разработки для PHP.

1.4 База данных для приложения

Так как система проектируется для работы с большими объемами данных, требуется особенно тщательно подойти к выбору хранилища данных приложения. В таблице 5 приведено сравнение некоторых систем управления базами данных (далее СУБД).

Таблица 5 — Сравнение некоторых популярных СУБД

	PostgreSQL	**MySQL**	**SQLite**	**MongoDB**

Расширяемость	+	+	-	+
Скорость	+	-	-	+
Отказоустойчивость	+	-	-	+
Безопасность	+	+	-	+
Реляционная	+	+	+	-

Под расширяемостью понимается наличие у СУБД внутренних механизмов и инструментов, позволяющих расширить базовый функционал СУБД, например — механизмы для поддержки хранимых процедур, новых типов и т.д.. Безопасность подразумевает наличие и надежность встроенных механизмов защиты доступа к данным. Отказоустойчивость отвечает за возможности интеграции инструментов мониторинга за состоянием и нагрузкой СУБД и наличие внутренних механизмов для организации таковой.

Таблица 5 позволяет определить наиболее эффективную СУБД - PostgreSQL. Немаловажным фактом является то, что PostgreSQL — реляционная СУБД, а значит, при проектировании можно воспользоваться инструментом реляционной алгебры для наиболее эффективного использования информации.

Помимо приведенных выше преимуществ PostgreSQL, можно отдельно выделить ещё следующие:

- свободно распространяемое ПО;
- мощные инструменты индексации данных;
- надежные механизмы для работы с большими объемами данных.

2 Разработка веб-сервиса

2.1 Подготовка веб-сервера

Для серверной части необходимо установить веб-сервер, который имеет все необходимое для работы. Устанавливаем Symfony CLI.

Symfony CLI — это инструмент разработчика, который позволяет создавать, запускать и управлять приложениями Symfony прямо с терминала.

Открываем терминал и вводим команду brew install symfony-cli/tap/symfony-cli. После установки symfony в системе, создаем проект с помощью команды symfony new my_project_directory --version=5.4 –webapp.

Получим следующую структуру проекта, показана на рисунке 4

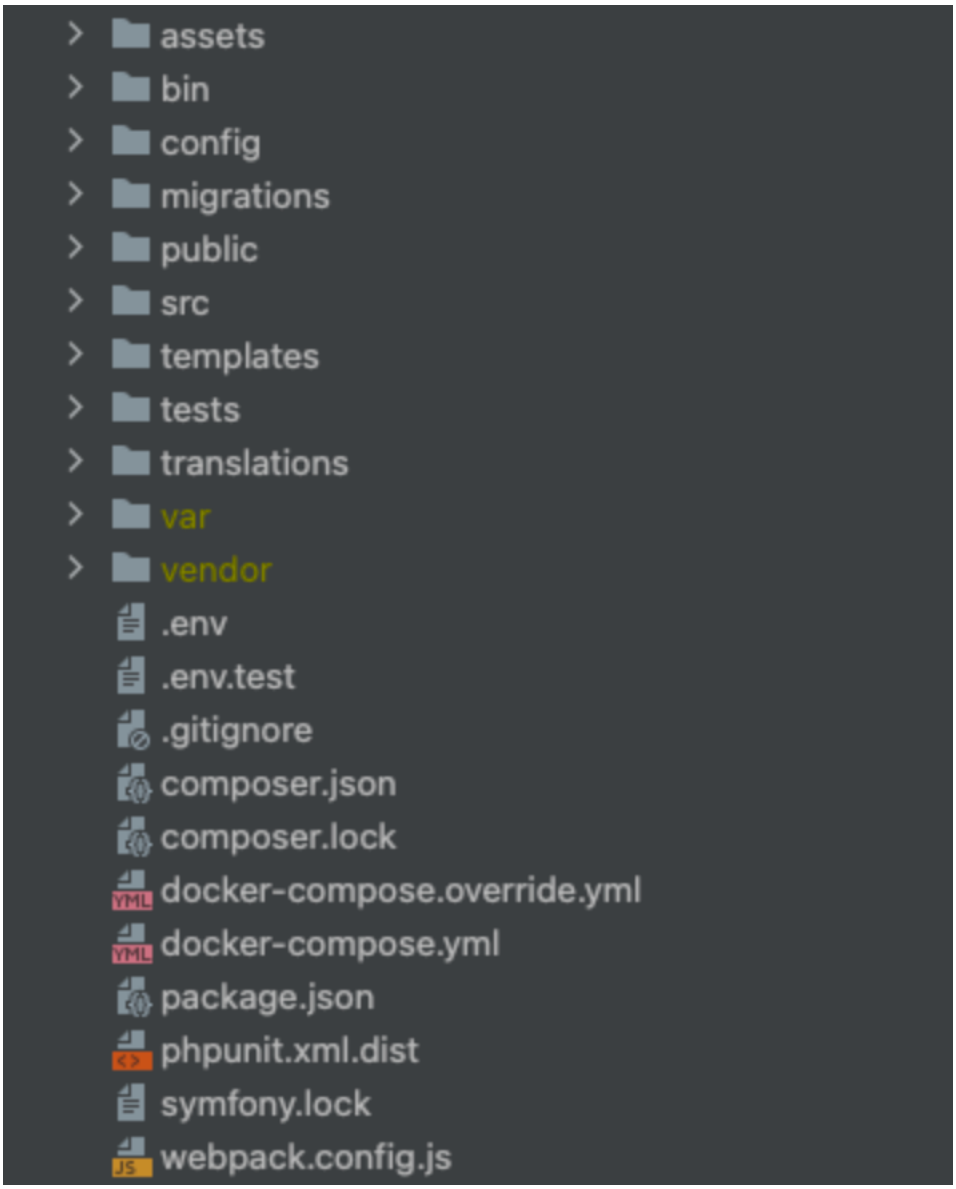


Рисунок 4 – Структура проекта

Точка входа, иначе говоря, index.php, располагается в каталоге public. В папке public по умолчанию хранятся файлы, к которым разрешен публичный доступ.

В каталоге src распологается основная логика приложения, например контроллеры, сущности, модели и сервисы и так далее.

В каталоге migrations находятся файлы миграций, для работы с базой данных.

В каталоге templates находятся виды (view), которые используют шаблонизатор Twig. Все шаблоны кэшируются до следующих изменений, что экономит ресурсы системы.

В каталоге config располагаются конфигурационные файлы, в которых описываются подключения к базам данных, пути для контроллеров, конфигурация контейнера зависимостей [7]. Пример структуры файла конфигурации контейнера зависимостей показан на рисунке 5.

Рисунок 5 – Файл конфигурации контейнера зависимостей

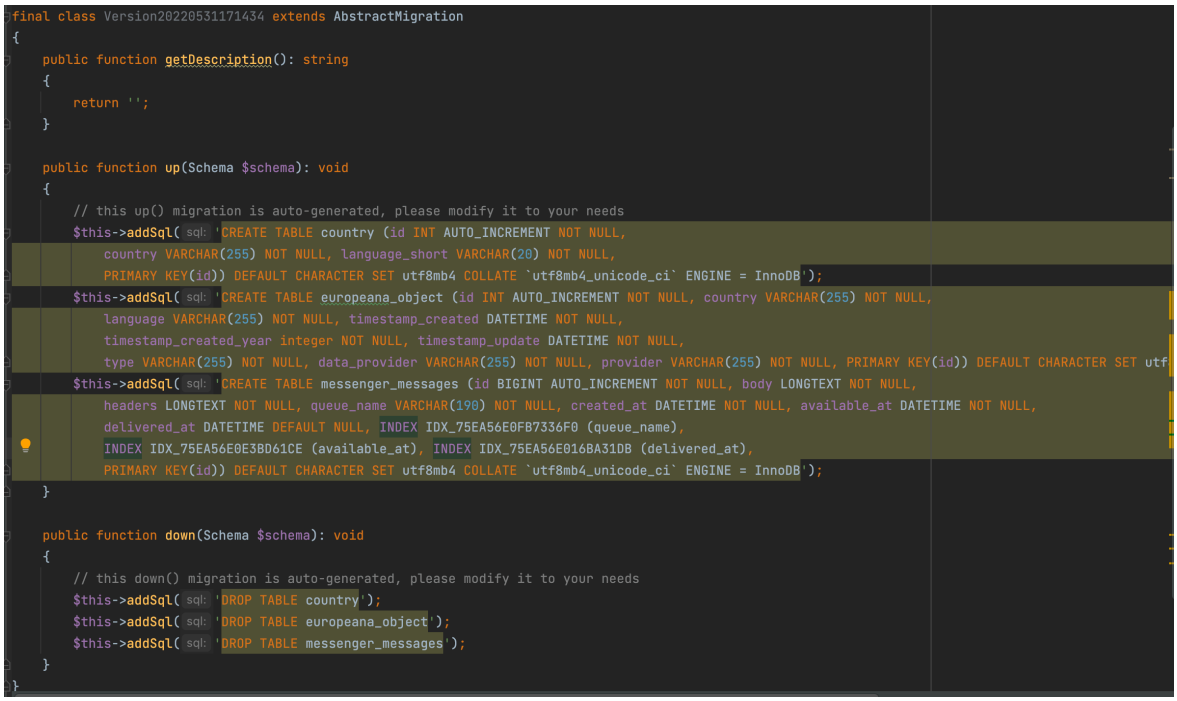
2.2 Проектирование базы данных

Грамотное проектирование базы данных очень важно для работы приложения. Цель проектирования состоит в получении моделей, отражающих предметную область и информационные потребности пользователей. [8]

PostgreSQL – это реляционная база данных. Важная особенность реляционных систем, их отличие от одноуровневых баз данных – возможность располагать данные в нескольких таблицах. Взаимосвязанные данные можно хранить в отдельных таблицах и объединять по ключу, общему для обеих таблиц. Ключ – это отношение (relation) между таблицами.

Миграция базы данных — это процедура перемещения данных с одной или нескольких исходных платформ в другую целевую базу данных.

После анализа возможностей API Europeana, была определена структура базы данных и необходимые метаданные. Для добавления данных в базу данных (далее БД), была написана миграция, скриншот кода которой демонстрируется на рисунке 6. В результате получилась готовая к наполнению база данных, пример структуры которой изображен на рисунке 7.



```
final class Version20220531171434 extends AbstractMigration
{
    public function getDescription(): string
    {
        return '';
    }

    public function up(Schema $schema): void
    {
        // this up() migration is auto-generated, please modify it to your needs
        $this->addSql( sql: 'CREATE TABLE country (id INT AUTO_INCREMENT NOT NULL,
            country VARCHAR(255) NOT NULL, language_short VARCHAR(20) NOT NULL,
            PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE `utf8mb4_unicode_ci` ENGINE = InnoDB');
        $this->addSql( sql: 'CREATE TABLE europeana_object (id INT AUTO_INCREMENT NOT NULL, country VARCHAR(255) NOT NULL,
            language VARCHAR(255) NOT NULL, timestamp_created DATETIME NOT NULL,
            timestamp_created_year integer NOT NULL, timestamp_update DATETIME NOT NULL,
            type VARCHAR(255) NOT NULL, data_provider VARCHAR(255) NOT NULL, provider VARCHAR(255) NOT NULL, PRIMARY KEY(id)) DEFAULT CHARACTER SET utf
        $this->addSql( sql: 'CREATE TABLE messenger_messages (id BIGINT AUTO_INCREMENT NOT NULL, body LONGTEXT NOT NULL,
            headers LONGTEXT NOT NULL, queue_name VARCHAR(190) NOT NULL, created_at DATETIME NOT NULL, available_at DATETIME NOT NULL,
            delivered_at DATETIME DEFAULT NULL, INDEX IDX_75EA56E0FB7336F0 (queue_name),
            INDEX IDX_75EA56E0E3BD61CE (available_at), INDEX IDX_75EA56E016BA31DB (delivered_at),
            PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE `utf8mb4_unicode_ci` ENGINE = InnoDB');
    }

    public function down(Schema $schema): void
    {
        // this down() migration is auto-generated, please modify it to your needs
        $this->addSql( sql: 'DROP TABLE country');
        $this->addSql( sql: 'DROP TABLE europeana_object');
        $this->addSql( sql: 'DROP TABLE messenger_messages');
    }
}
```

Рисунок 6 – Файл миграции базы данных.

```
id                      int auto_increment
    primary key,
country                 varchar(255) not null,
language                varchar(255) not null,
timestamp_created       datetime     not null,
timestamp_created_year  int          not null,
timestamp_update        datetime     not null,
type                    varchar(255) not null,
data_provider           varchar(255) not null,
provider                varchar(255) not null
```

Рисунок 7 – Пример структуры таблицы базы данных.

После подготовки базы данных, была написана консольная команда, заполняющая БД, данными с Europeana, она изображена на рисунке 8, рисунке 9. Команда по API обращается к Europeana и получает данные, после чего сохраняет нужную часть в БД. Также была предусмотрена ситуация, если возникнет ошибка, команда остановит выполнение и запишет сообщение об ошибке в логи.

```php
public function execute(InputInterface $input, OutputInterface $output): int
{
    echo 'Fetching start' . PHP_EOL;

    try {
        $this->fetchAll();
    } catch (\Exception $e) {
        echo 'Fetching failed' . PHP_EOL;
        $this->logger->error( message: 'Fetching failed: ' . $e->getMessage(), ['error' => $e->getTrace()]);
    }
    echo 'Fetching end' . PHP_EOL;

    return 1;
}
```

Рисунок 8 – Метод команды, запускающий метод получения данных с Europeana

```php
private function fetchAll(): void
{
    $params = [
        'query' => [
            'wskey' => self::API_KEY,
            'query' => '*',
            'rows' => '100',
            'cursor' => '*',
        ],
    ];

    while (true) {
        sleep( seconds: 1);
        $response = $this->client->request(
            Request::METHOD_GET,
            'https://api.europeana.eu/record/v2/search.json',
            $params
        );

        $data = $response->toArray();

        $dto = EuropeanaObject::getCollectionFromArray($this->presentDto($data['items']));
        $this->europeanaObjectRepository->addBatch($dto);

        $params['query']['cursor'] = $data['nextCursor'];

        if (!$params['query']['cursor']) {
            break;
        }
    }
}
```

Рисунок 9 – Метод команды, получающий данные с Europeana

2.3 Создание классов для работы с объектами из базы данных

Для получения и работы с объектами из БД был создан класс сущности объекта, представленный на рисунке 10, этот класс позволяет работать с объектом базы данных, как с объектом php класса, метод для создания объектов класса из данных Europeana показан на рисунке 11. Также был создан класс репозитория, позволяющий обращаться к базе данных из php кода, он продемонстрирован на рисунке 12.

```php
public function __construct(
    $country,
    $language,
    $timestamp_created,
    $timestamp_created_year,
    $timestamp_update,
    $type,
    $data_provider,
    $provider)
{
    $this->country = $country;
    $this->language = $language;
    $this->timestamp_created = $timestamp_created;
    $this->timestamp_created_year = $timestamp_created_year;
    $this->timestamp_update = $timestamp_update;
    $this->type = $type;
    $this->data_provider = $data_provider;
    $this->provider = $provider;
}
```

Рисунок 10 –Конструктор класса сущности, позволяющий работать с объектом базы данных, как с объектом php класса.

```php
public static function getCollectionFromArray(array $rows): array
{
    return \array_map(
        static fn (array $row) => new self(
            (string)$row['country'],
            (string)$row['language'],
            \DateTimeImmutable::createFromFormat( format: 'Y-m-d H:i:s', $row['timestamp_created']->format('Y-m-d H:i:s')),
            (int)$row['timestamp_created']->format('Y'),
            \DateTimeImmutable::createFromFormat( format: 'Y-m-d H:i:s', $row['timestamp_update']->format('Y-m-d H:i:s')),
            (string)$row['type'],
            (string)$row['dataProvider'],
            (string)$row['provider'],
        ),
        $rows
    );
}
```

Рисунок 11 – Код класса сущности. Он позволяет создать объект из данных Europeana.

```php
/**
 * @extends ServiceEntityRepository<EuropeanaObject>
 *
 * @method EuropeanaObject|null find($id, $lockMode = null, $lockVersion = null)
 * @method EuropeanaObject|null findOneBy(array $criteria, array $orderBy = null)
 * @method EuropeanaObject[]    findAll()
 * @method EuropeanaObject[]    findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null)
 */
class EuropeanaObjectRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, entityClass: EuropeanaObject::class);
        $this->_em = $this->getEntityManager();
    }

    public function add(EuropeanaObject $entity, bool $flush = false): void
    {
        $this->getEntityManager()->persist($entity);

        if ($flush) {
            $this->getEntityManager()->flush();
        }
    }
```

Рисунок 12 – Код класса репозитория. Он позволяет обращаться к базе данных из php кода.

2.4 Создание контроллера и пользовательского представления

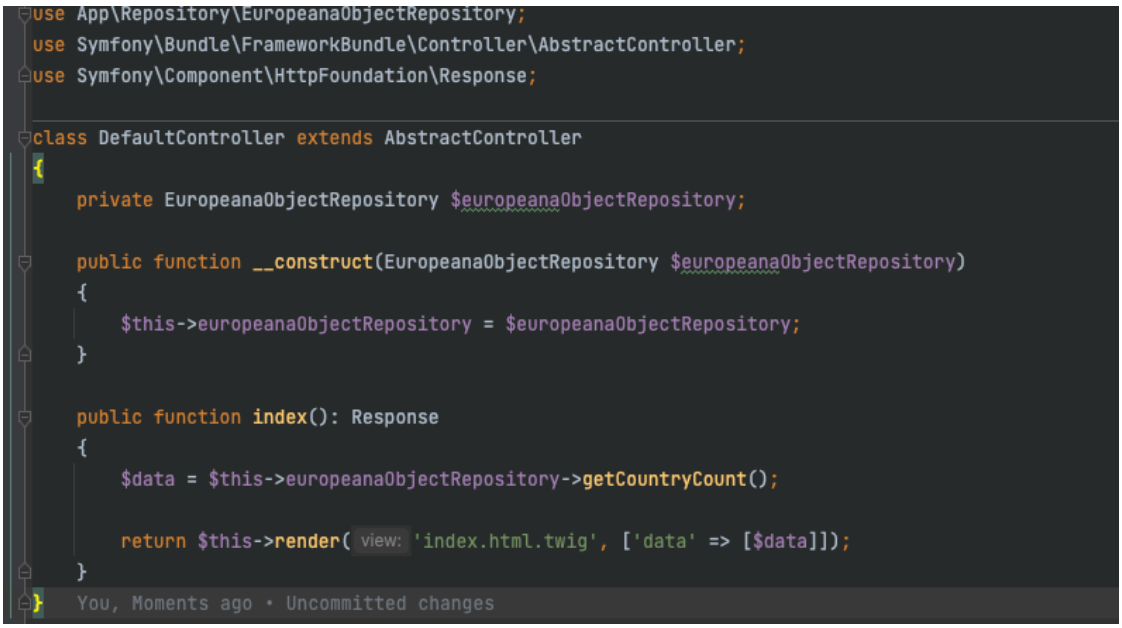
Контроллер управляет запросами пользователя (получаемые в виде запросов HTTP GET или POST, когда пользователь нажимает на элементы интерфейса для выполнения различных действий). Его основная функция — вызывать и координировать действие необходимых ресурсов и объектов, нужных для выполнения действий, задаваемых пользователем. Обычно контроллер вызывает соответствующую модель для обработки данных, перед тем как отдать их во view. [9]

Вид (view) обеспечивает различные способы представления данных, которые получены из модели. Он может быть шаблоном, который заполняется данными. Может быть несколько различных видов, и контроллер выбирает, какой подходит наилучшим образом для текущей ситуации.

Веб приложение обычно состоит из набора контроллеров, моделей и видов. Контроллер может быть устроен как основной, который получает все запросы и вызывает другие контроллеры для выполнения действий в зависимости от ситуации.

Для отображения пользователю данных был создан класс контроллера, изображенный на рисунке 13. Этот класс отвечает за прием данных с view и отправкой view на отрисовку пользователю. Также в контроллере вызывается класс подготавливающий данные для вывода. Далее был создан файл view index.html.twig и main.js, скриншот которого демонстрируется на рисунке 14, рисунке 15.

Эти файлы отвечают за визуализацию графических объектов. Благодаря им строятся диаграммы и графики.



```php
use App\Repository\EuropeanaObjectRepository;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;

class DefaultController extends AbstractController
{
    private EuropeanaObjectRepository $europeanaObjectRepository;

    public function __construct(EuropeanaObjectRepository $europeanaObjectRepository)
    {
        $this->europeanaObjectRepository = $europeanaObjectRepository;
    }

    public function index(): Response
    {
        $data = $this->europeanaObjectRepository->getCountryCount();

        return $this->render( view: 'index.html.twig', ['data' => [$data]]);
    }
}
You, Moments ago • Uncommitted changes
```

Рисунок 13 – Класс контроллера приложения. Он отвечает за прием данных с view и отправкой view на отрисовку пользователю. Также в контроллере вызывается класс подготавливающий данные для вывода

```
<div class="container-fluid pt-4 px-4">
    <div class="row g-4">
        <div class="col-sm-12 col-xl-6">
            <div class="bg-light text-center rounded p-4">
                <div class="d-flex align-items-center justify-content-between mb-4">
                    <h6 class="mb-0">Топ 5 стран по количеству контента</h6>
                </div>
                <canvas id="year_analyze"></canvas>
            </div>
        </div>
        <div class="col-sm-12 col-xl-6">
            <div class="bg-light text-center rounded p-4">
                <div class="d-flex align-items-center justify-content-between mb-4">
                    <h6 class="mb-0">Рост количества контента по типу и году</h6>
                </div>
                <canvas id="types_and_year_analyze"></canvas>
            </div>
        </div>
        <div class="col-sm-12 col-xl-6">
            <div class="bg-light text-center rounded p-4">
                <div class="d-flex align-items-center justify-content-between mb-4">
                    <h6 class="mb-0">Рост количества контента по годам</h6>
                </div>
                <canvas id="types_analyze"></canvas>
            </div>
        </div>
    </div>
</div>
```

Рисунок 14 – Код, отвечающий за вывод диаграмм. Он отвечает за вывод графических объектов.

```
var ctx3 = $("#analyze").get(0).getContext("2d");
var myChart3 = new Chart(ctx3, {
    type: "line",
    data: {
        labels: data['years'],
        datasets: [{
            label: "Текст",
            fill: false,
            backgroundColor: "rgba(151,255,232,0.7)",
            data: data['data']['1']
        },
            {
                label: "Видео",
                fill: false,
                backgroundColor: "rgba(0,166,255,0.7)",
                data: data['data']['2']
            },
            {
                label: "3D",
                fill: false,
                backgroundColor: "rgba(222,149,225,0.7)",
                data: data['data']['3']
            },
            {
                label: "Аудио",
                fill: false,
                backgroundColor: "rgba(255,237,86,0.7)",
                data: data['data']['4']
            },
            {
                label: "Изображения",
                fill: false,
                backgroundColor: "rgba(115,255,154,0.7)",
                data: data['data']['5']
            }]
    },
    options: {
        responsive: true
    }
});
```

Рисунок 15 – Код, отвечающий за вывод диаграмм. Он отвечает за вывод графических объектов.

3 Результаты разработки

В данной главе будет демонстрация работы разработанного веб-сервиса.

Главная страница веб-сервиса показана на рисунке 16.

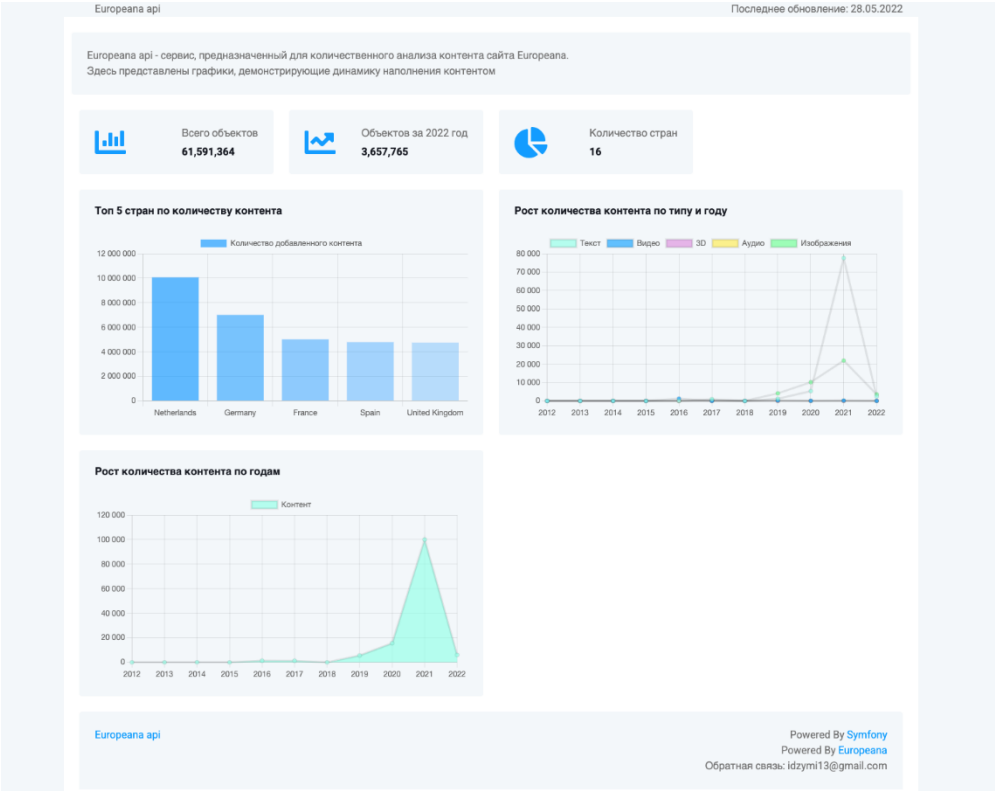


Рисунок 16 – Главная страница веб-сервиса

В верхней части экрана находится название веб-сервиса и дата последнего обновления данных, после чего идет описание сервиса, показано на рисунке 17.

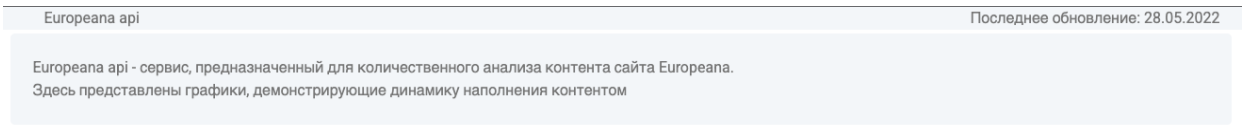


Рисунок 17 – Верхняя часть веб-сервиса

Ниже идут три блока выводящие количественную информацию Europeana, такую, как количество объектов всего, количество объектов добавленных за текущий год, количество представленных стран. Демонстрация на рисунке 18.

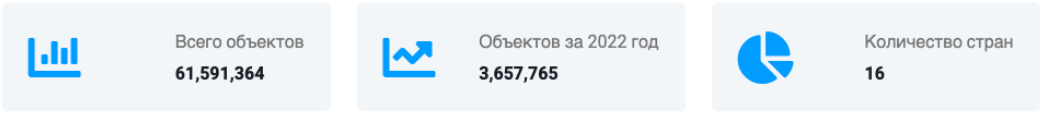


Рисунок 18 – Блоки с информацией

В основном блоке страницы представлены 3 диаграммы.

Первая диаграмма отображает количество объектов в 5 странах, наиболее представленными в Europeana. Показана на рисунке 19.

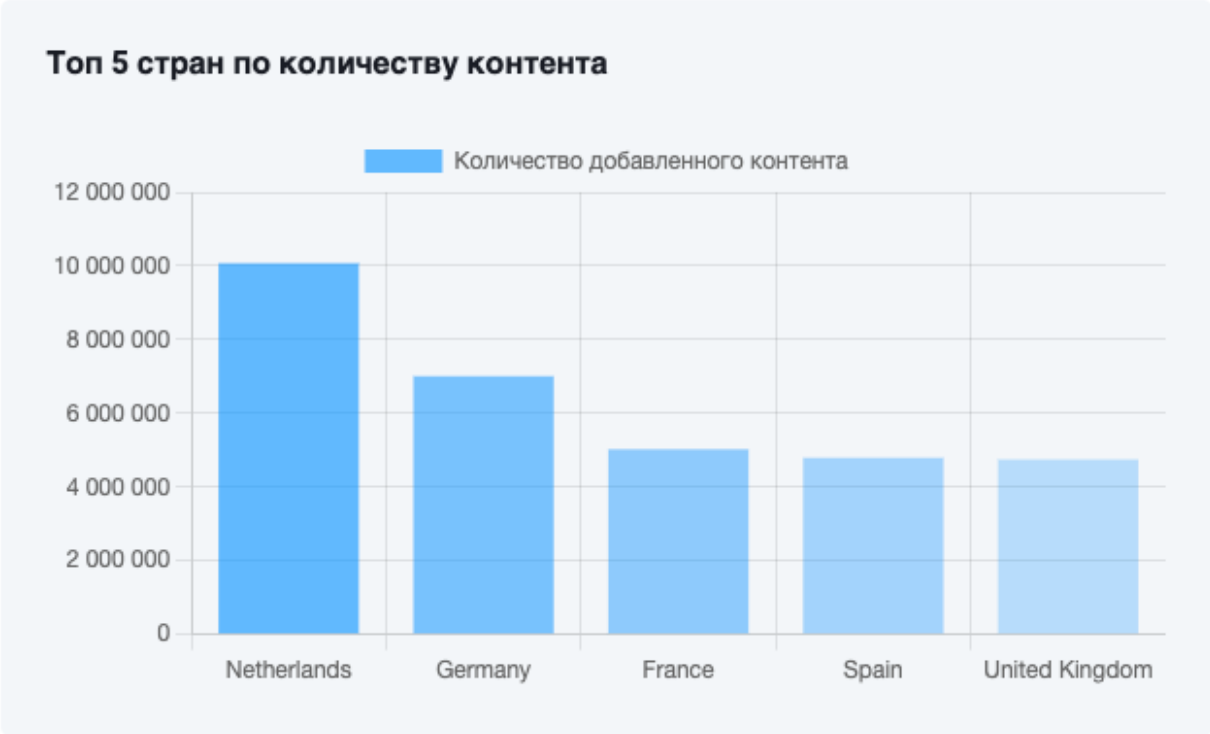


Рисунок 19 – Диаграмма топ 5 стран

Вторая диаграмма отображает рост количества контента по типу и году, вид данной диаграммы показан на рисунке 20. Также это диаграмма может отображать данные по выбранным типам данных и динамически обновлять контент, данная функция показана на рисунке 21.

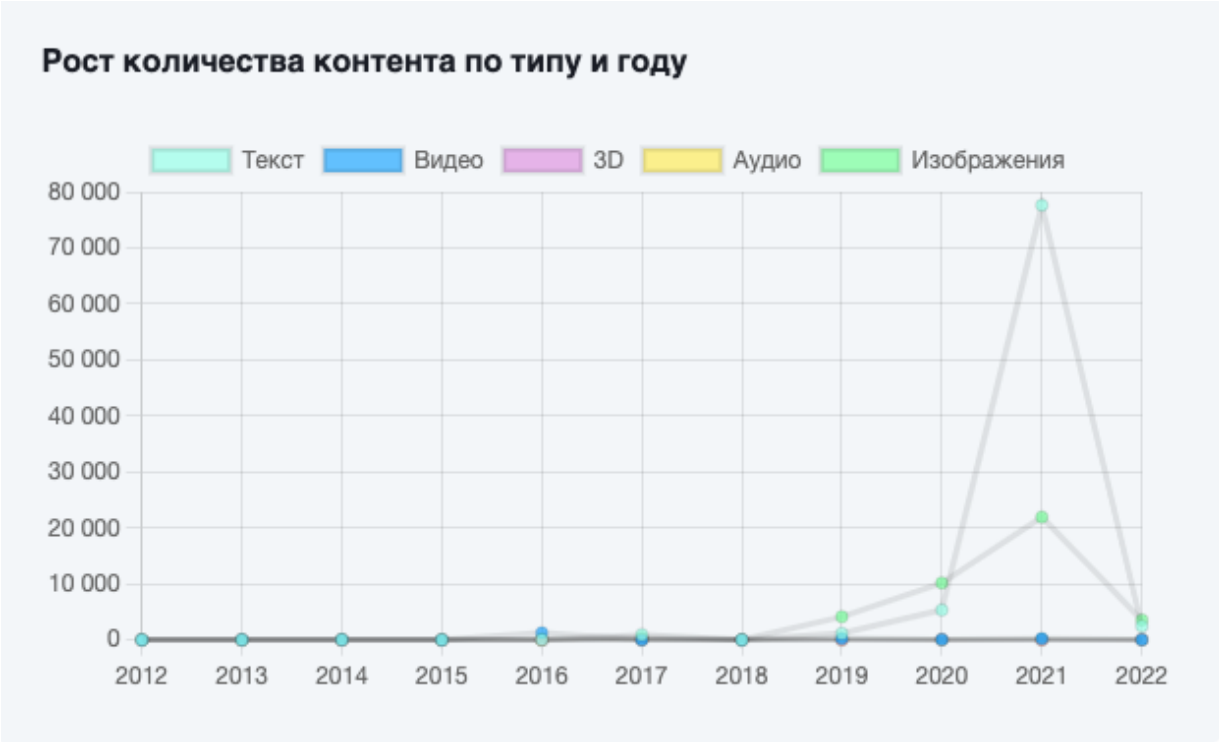


Рисунок 20 – Вид диаграммы количества контента по типу и году

Рисунок 21 – Функция отображения по выбранным типам

Третья диаграмма отображает общий рост количества контента на сайте Europeana по годам, показана на рисунке 22.

Рисунок 22 – Диаграмма роста количества контента по годам

В нижней части веб-сервиса находится информационный блок, в котором содержится информация для обратной связи, ссылка на данный веб-сервис, указание фреймворка использованного при разработке и информация о источнике данных. Данный блок показан на рисунке 23.

Рисунок 23 – Информационный блок в нижней части

Пример работы веб-сервиса на мобильный устройствах показан на рисунке 24, рисунке 25.

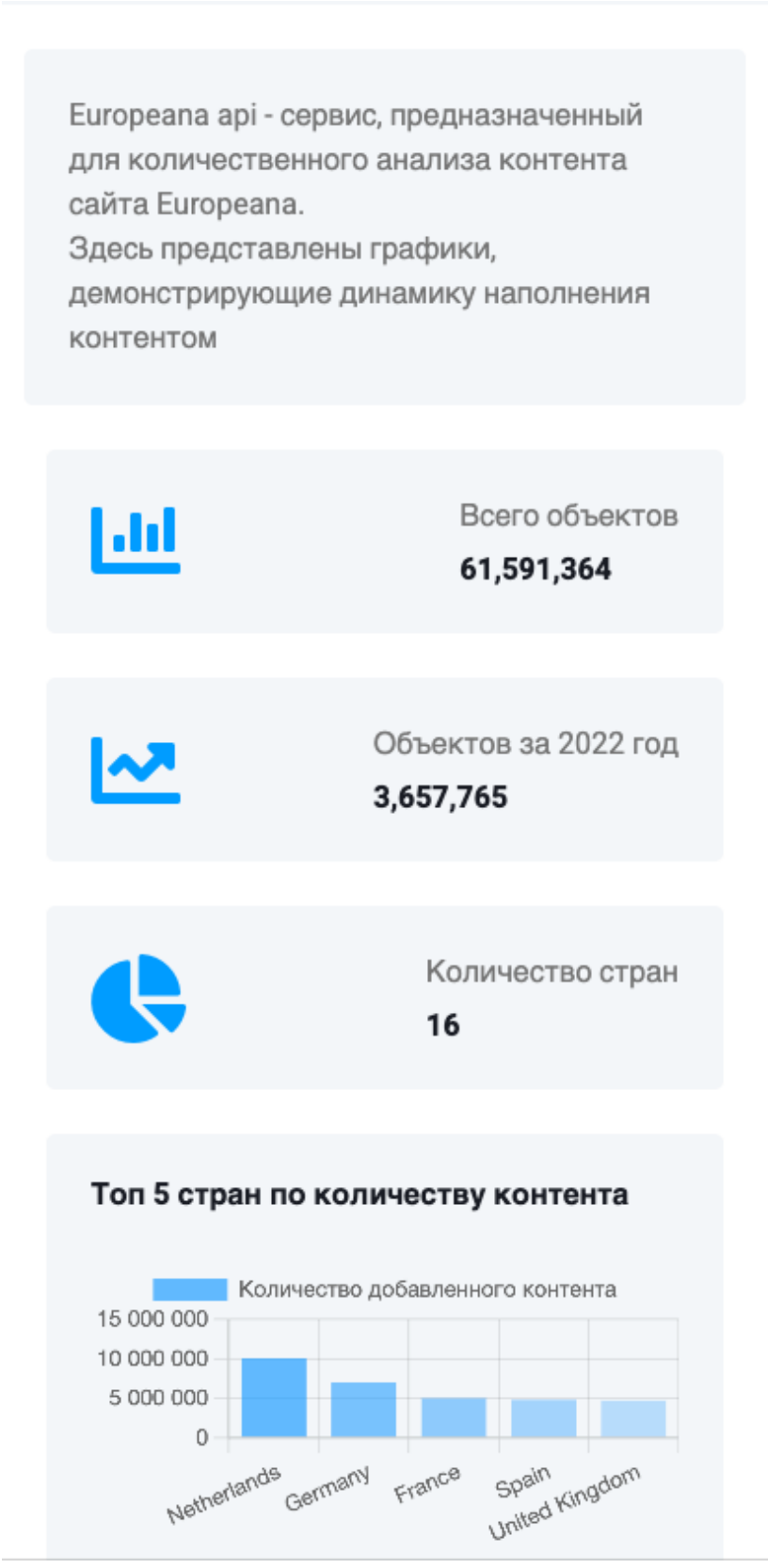


Рисунок 24 – Вид с мобильного устройства (1)

Рисунок 25 – Вид с мобильного устройства (2)

Процесс обработки веб-сервером данных и скорость работы приложения показаны на рисунке 26.

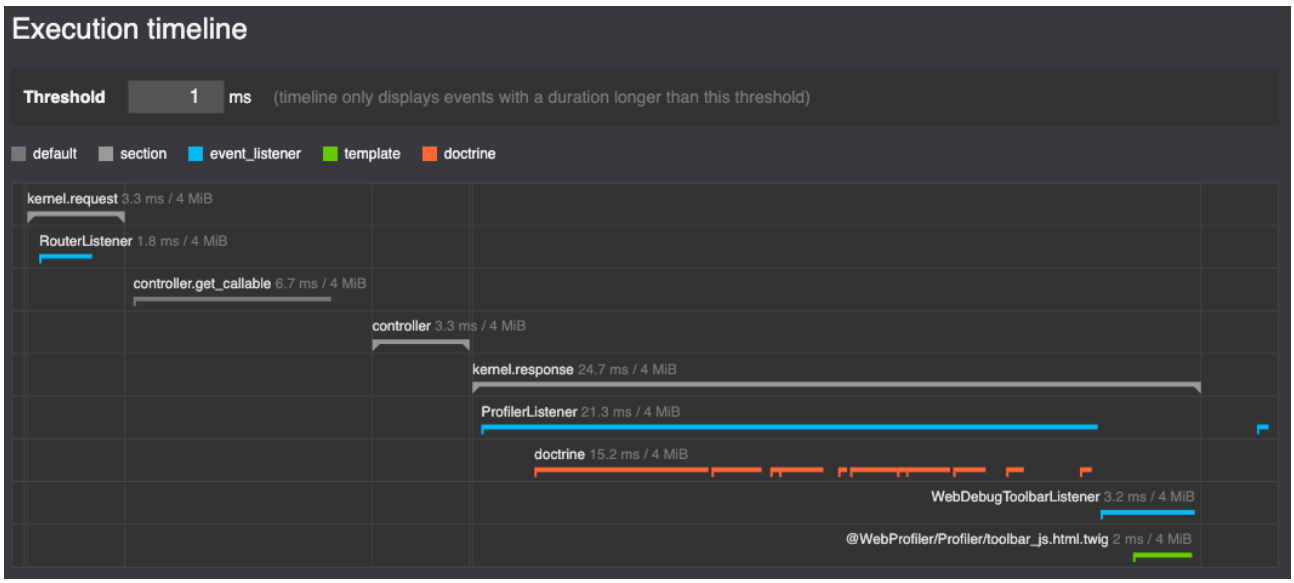


Рисунок 26 – Скорость работы приложения и порядок обработки запросов

ЗАКЛЮЧЕНИЕ

Целью данной работы являлось выявить принципы использования программного интерфейса приложения (API) для исследования коллекций цифровой библиотеки Europeana.

В ходе исследования были изучены принципы работы Europeana API, и найдены способы получения данных от Europeana, в соответствии с правилами работы с API, разобраны преимущества различных баз данных.

Был разработан веб-сервис позволяющий использовать программный интерфейс приложения для количественного исследования коллекций Europeana.

Опираясь на все вышесказанное, можно считать все задачи исследования выполненными, а цель – достигнутой.

Цифровая библиотека Europeana содержит огромный объем разнообразного контента и обладает удобным поисковым интерфейсом. Несмотря на это, в настоящий интерфейс требует существенных улучшений и не представляется удобным для использования в исследованиях.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Кучерявый, Е. Что такое API и как он помогает в создании программных систем. [Электронный ресурс]: образовательная платформа, 2020. – Режим доступа: https://skillbox.ru/media/code/chto_takoe_api/

2. EuropeanaPro. Ourmission. [Электронный ресурс] – Режим доступа: https://pro.europeana.eu/about-us/mission

3. Europeana Pro. Shareyourdata: process [Электронный ресурс] – Режим доступа: https://pro.europeana.eu/share-your-data/process

4. Europeana Pro. EuropeanaContentStrategy [Электронный ресурс] – Режим доступа: https://pro.europeana.eu/post/europeana-content-strategy

5. EDM (модель данных с использованием сущностей), 2022. [Электронный ресурс] – Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/framework/data/adonet/entity-data-model

6. Сервис-контейнер, 2021. [Электронный ресурс] – Режим доступа: https://symfony.ru/doc/current/service_container.html

7. Д.А. Попова-Коварцева, Е.В. Сопченко. Основы проектирования баз данных. – Самара: Самарский университет, 2019. – С. 37–42.

8. Фастунов, С. Концепция MVC. [Электронный ресурс] – Режим доступа: https://ruseller.com/lessons.php?id=666

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Гуманитарный институт
Кафедра информационных технологий в креативных и культурных индустриях

УТВЕРЖДАЮ
И. о. заведующего кафедрой
_____ А. В. Усачёв
подпись
«_5_» _____07_____2022 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.03.14 «Прикладная информатика в искусстве и гуманитарных науках»

Использование программного интерфейса приложения (API) для исследования коллекций цифровой библиотеки «Европеана»

Руководитель _____ ст. преподаватель Е.Р. Брюханова
подпись, дата

Выпускник _____ 05.07.2022 Д.М. Изосимов
подпись, дата

Красноярск 2022

Продолжение титульного листа бакалаврской работы по теме Использование программного интерфейса приложения (API) для исследования коллекций цифровой библиотеки «Европеана»

Нормоконтролер _____ Е. Р. Брюханова
подпись, дата