|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

|  |  |
| --- | --- |
| ФАКУЛЬТЕТ | Робототехника и комплексная автоматизация |
| КАФЕДРА | Системы автоматизированного проектирования |

**ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ**

|  |  |
| --- | --- |
| Студент | Принцев Владислав Сергеевич |
| Группа | РК6-73Б |
| Тип практики | Эксплуатационная |
| Название предприятия | «НИИ Автоматизации Производственных  Процессов МГТУ им. Н.Э. Баумана» |

|  |  |  |
| --- | --- | --- |
| Студент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **Принцев В.С.** |
|  | *подпись, дата* | *фамилия, и.о.* |
| Руководитель практики | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **Оглоблин Д.И.** |
|  | *подпись, дата* | *фамилия, и.о.* |

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Москва, 2023 г.*

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

УТВЕРЖДАЮ

Заведующий кафедрой \_\_\_\_\_\_\_\_

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

« \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_ г.

**ЗАДАНИЕ**

**на прохождение производственной практики**

      эксплуатационная

тип практики

Студент

Принцев Владислав Сергеевич        4        курса группы        РК6-73Б

Фамилия Имя Отчество № курса индекс группы

в период с     03    .    07    .    2023     г. по     31    .    07    .    2023     г.

*Предприятие:* «НИИ Автоматизации Производственных Процессов МГТУ им. Н.Э. Баумана»

*Подразделение:* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(отдел/сектор/цех)

*Руководитель практики от предприятия (наставник):*

                                                 Витюков Фёдор Андреевич

(Фамилия Имя Отчество полностью, должность)

*Руководитель практики от кафедры:*

                                               Оглоблин Дмитрий Игоревич

(Фамилия Имя Отчество полностью, должность)

*Задание:*

1. Используя Svelte, TypeScript и Tailwind CSS, реализовать дизайн шаблона «CardsView», представленного в Figma.
2. Адаптировать внешний вид интерфейса под мониторы различного размера, включая 4k и 8k.
3. Реализовать получение списка игр путем запроса к приложению на Drogon C++, которое берет данные из базы данных на PostgreSQL.
4. Реализовать формирование при первом запросе статической HTML-страницы, сформированной в приложении на Drogon C++, на основе данных из базы данных.
5. Реализовать динамическую загрузку данных из приложения на Drogon C++ при пролистывании вниз или нажатии на кнопку «Show more…».

Дата выдачи задания «    3    »         июля             2023     г.

Руководитель практики от предприятия  **\_\_\_\_\_\_\_\_\_\_ /**   Витюков Ф.А.**/**

Руководитель практики от кафедры **\_\_\_\_\_\_\_\_\_\_ /** Оглоблин Д.И.**/**

Студент **\_\_\_\_\_\_\_\_\_\_ /** Принцев В.С. **/**

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc145180306)

[1. HTML 6](#_Toc145180307)

[2. CSS 8](#_Toc145180308)

[3. JavaScript 10](#_Toc145180309)

[4. Svelte 12](#_Toc145180310)

[5. Tailwind CSS 13](#_Toc145180311)

[6. Drogon C++ 14](#_Toc145180312)

[7. Общие положения о выполнении задания 15](#_Toc145180313)

[8. Frontend-часть 18](#_Toc145180314)

[9. Backend-часть 23](#_Toc145180315)

[ЗАКЛЮЧЕНИЕ 26](#_Toc145180316)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 28](#_Toc145180317)

ВВЕДЕНИЕ

Сложно представить современный мир без веб-технологий. Веб-разработка является одной из самых важных отраслей.

Большинство современных сайтов состоит из двух составляющих: frontend и backend. С frontend взаимодействуют обычные пользователи сайта. Здесь реализуется внешний вид веб-страницы, а также элементы, с которыми может взаимодействовать пользователь, например, кнопки, ссылки, поля для ввода и т.д. В backend реализуется взаимодействие с сервером. Здесь происходит выполнение некоторой логики с данными на сервере, работа с базой данных.

Для обоих составляющих существуют свои подходы и технологии, а также языки программирования и среды разработки.

Базовыми языками для frontend являются HTML (HyperText Markup Language – «язык [гипертекстовой](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%BF%D0%B5%D1%80%D1%82%D0%B5%D0%BA%D1%81%D1%82) разметки»), CSS (Cascading Style Sheets – «каскадные таблицы стилей»)и JavaScript. Именно эти языки «понимает» любой веб-браузер. Любые прочие языки для frontend основаны на них и компилируются в них.

Для упрощения frontend-разработки существуют различные библиотеки и среды разработки. В данном проекте используются среды разработки Svelte, который упрощает реализацию взаимодействия между HTML, JavaScript и CSS, и Tailwind CSS, который предоставляет набор классов стилей, для более удобной настройки дизайна сайта. Для написания скрипт-кода в Svelte используется расширение для JavaScript – TypeScript, который, в отличие от оригинального JavaScript, поддерживает строгую типизацию и полноценное ООП, что упрощает разработку крупных проектов.

Для backend-разработки в данном проекте используется среда разработки Drogon, основанный на языке C++. Он поддерживает работу с HTTP-запросами/ответами, сессией, cookies, формированием динамических шаблонов страниц на сервере, JSON, некоторыми базами данных и т.д. Благодаря тому, что используется язык C++, Drogon является одной из самых быстрых сред разработки для backend.

Для хранения данных в проекте используется реляционная СУБД PostgreSQL. Является одной из самых популярных и быстрых СУБД. Drogon С++ изначально имеет ее поддержку.

1. HTML

HTML (Hypertext Markup Language) – это язык разметки, используемый для создания структуры и представления веб-страниц. Основной синтаксической единицей языка являются тэги, которые определяют содержимое страницы.

Тэги подразделяются на строчные и блочные.

Элементы, идущие друг за другом, определенные строчными тэгами, стремятся выстроиться последовательно в один параграф. Такие элементы имеют размер, позволяющий вместить их содержимое. Например, строчный тэг *<span>* используется для выделения подстроки в параграфе, для которой необходимо задать свой стиль в CSS, тэг *<a>* – для задания ссылки.

Элементы, определенные блочными тэгами, стремятся занять все доступное пространство строк, на которых расположено их содержимое. Сам элемент, а также элемент, идущий далее, будут начинаться с новых строк. К примеру, тэг разделителя *<div>* используется для задания какого-то логически цельного элемента страницы, например карточки игры в данном проекте, тэг *<p>* используется для определения параграфа, а тэги *<h1>*, *<h2>*, …, *<h6>* для заголовков.

Иной классификацией тэгов HTML является разделение на одиночные и двойные.

Одиночные, в отличие от двойных, не имеют закрывающего элемента. К одиночным, например, относится тэг для определения изображения *<img>*, а к двойным – упомянутые выше *<span>*, *<div>*, *<p>*. Двойные тэги позволяют задавать содержимое внутри себя, в том числе и элементы, определенные другими тэгами.

Тэги поддерживают задание параметров. Для упомянутого выше тэга определения картинки *<img>* существует параметр *src*, который задает ссылку на изображение. Для всех тэгов существуют параметры для задания стилей CSS *class* и *style*. В первом указываются CSS-классы, определенные в файле CSS, а во втором CSS-параметры. Еще одним параметром, доступным для всех тэгов, является параметр *id* для задания идентификатора элемента, который может быть полезен для работы с DOM в JavaScript.

В HTML существует несколько десятков тэгов, однако из-за возможности гибкой настройки стиля в CSS, а также создания интерактивности в JavaScript, нет необходимости использования их всех. Поэтому в данном проекте в основном используются тэги *<div>*, *<p>*, *<span>*, *<a>*.

С помощью HTML и связанных технологий можно создавать разнообразные веб-страницы и приложения.

1. CSS

CSS – «каскадные таблицы стилей» (Cascading Style Sheets). Это язык таблиц стилей, используемый для описания внешнего вида и форматирования документов, написанных на языке HTML.

С помощью CSS можно определить различные стили и применять их к элементам HTML. Это включает установку свойств, таких как цвета, шрифты, размеры, отступы, границы и позиционирование.

Каждому стилю, заданному в CSS, соответствует селектор, необходимый для определения элементов в HTML, к которым применяется этот стиль. Селекторы могут задавать стили для элементов, определенных определенным HTML-тэгом (например, *«div»*), а также элементов с определенным классом в параметре *class* (например, *«.myclass»*).

Селектор можно задавать для элементов, находящихся внутри элементов, которым соответствует другой селектор. Например, для элементов с классом *myclass2*, которые находятся только внутри элементов с классом *myclass1*, необходимо написать *«.myclass1 .myclass2»*.

Селекторам можно задавать псевдоэлементы и псевдоклассы. Например, для стиля, который применяется к элементу, на который наведен курсор, используется псевдокласс *:hover* (например, *«.btn:hover»*), а для стиля, который применяется к первой строке элемента, используется псевдоэлемент *::first-line* (например, *«p::first-line»*).

CSS-стили задаются с помощью свойств и параметров к ним. Существует большое количество различных свойств, которые позволяют очень гибко настроить внешний вид страницы.

Для задания размерных параметров существуют различные единицы измерения. К некоторым из них относятся пиксели *px*, задающие абсолютный размер, проценты *%* задающие размер относительно родительского элемента, а также *rem* и *em*, задающие размер относительно размера шрифта коревого и родительского элемента соответственно.

Для создания контейнера элементов в CSS существуют flexbox и grid. Первый используется для размещения объектов в одном измерении (по вертикали или горизонтали), а второй в двумерной сетке. Хоть flexbox и grid работают по-разному, они могут использоваться для одних и тех же задач, поэтому выбор между ними зависит от удобства для разработчика.

CSS позволяет веб-разработчикам отделить представление документа от его структуры, что облегчает поддержку и обновление стилей веб-сайта.

1. JavaScript и TypeScript

JavaScript – это язык программирования, который применяется в основном для разработки frontend. Он известен своей гибкостью и поддерживается всеми основными веб-браузерами.

JavaScript позволяет разработчикам добавлять интерактивность, динамический контент и поведение на веб-сайты. Он часто используется для таких задач, как валидация форм, манипуляции с DOM (изменение структуры и содержимого веб-страниц), обработка событий (например, клики мыши или ввод с клавиатуры) и выполнение асинхронных запросов к серверам (например, получение данных из базы данных).

JavaScript использует Си-подобный синтаксис и основные конструкции языков программирования высокого уровня, такие как циклы, функции, переменные, условные конструкции, массивы, основные операции над переменными. Кроме того, в JavaScript существуют объекты, которые могут включать в себя поля типа ключ-значение, и лямбда-выражения, выполняющие такую же роль, что и функции.

В этом языке отсутствует строгая типизация, однако каждая переменная обладает своим типом: число (number), строка (string), булево значение (boolean) и т.д.

Для работы с динамическим отображением сайта существуют инструменты для работы с DOM (Document Object Module) – иерархической структурой веб-страницы на HTML. JavaScript позволяет добавлять, удалять и изменять элементы веб-страницы. Для реализации взаимодействия с пользователем элементам можно добавлять обработчики событий (например, на нажатие, наведение мыши и т.д.), которые в случае возникновения события вызывают обработчик в виде функции (или лямбда-выражения).

Отсутствие в JavaScript строгой типизации и полноценного ООП затрудняет разработку крупных проектов. Поэтому был разработан TypeScript, который решает основные проблемы JavaScript. TypeScript компилируется в обычный JavaScript, поэтому возможные синтаксические ошибки и ошибки типизации выявляются заранее, а не в момент выполнения.

JavaScript играет ключевую роль в современной веб-разработке и используется в различных областях, включая frontend- и backend-разработку, разработку мобильных приложений и интернет вещей (IoT). Он является неотъемлемой для множества веб-проектов и продолжает развиваться и совершенствоваться с течением времени.

1. Svelte

Svelte – это среда разработки для веб-приложений. Основным отличием Svelte от других подобных сред разработки является то, что он компилируется в чистый JavaScript во время сборки вместо того, чтобы выполняться во время выполнения на стороне клиента, что позволяет создавать более производительные веб-приложения.

Svelte не использует виртуальный DOM, как большинство других сред разработки. Вместо этого он преобразует код в JavaScript, который манипулирует DOM напрямую. Это позволяет избежать накладных расходов на виртуальный DOM и улучшить производительность.

В Svelte существует поддержка многомодульности (возможности разделить проект на несколько файлов – модулей) для удобного разделения блоков кода и возможности их переиспользования. Написание JavaScript- (или TypeScript-), HTML- и CSS-кода происходит в одном файле. При этом переменные в секции скрипта и стили видны только в этом файле, что позволяет избегать возможные конфликты имен.

В секции HTML-кода есть возможность подставлять переменные, объявленные в поле скрипта, использовать условные конструкции и циклы для формирования динамических страниц. Благодаря поддержке реактивности код страницы может автоматически изменяться при изменении данных в секции скрипта.

Svelte становится все более популярным среди разработчиков веб-приложений благодаря своей производительности и простоте использования. Он подходит для разработки как небольших проектов, так и крупных веб-приложений, и может быть хорошим выбором для тех, кто хочет экономить время на оптимизации и улучшении производительности своих приложений.

1. Tailwind CSS

Tailwind CSS – это инструмент для разработки веб-интерфейсов, который предоставляет набор готовых классов для стилизации HTML-элементов.

Одной из основных концепций Tailwind CSS является предоставление классов для каждого стиля, который можно применить к элементам веб-сайта. Например, можно использовать классы *text-red-500* для установки красного цвета текста или *bg-blue-300* для задания голубого фона. Tailwind CSS предоставляет обширный набор классов, которые позволяют управлять макетом, шрифтами, цветами, отступами и многими другими стилями. Это упрощает создание и изменение дизайна сайта.

Tailwind CSS легко настраивается. Можно создать свой собственный файл конфигурации, чтобы настроить цвета, шрифты, отступы и многое другое. Кроме того можно создавать собственные стили и компоненты для удобного дальнейшего использования.

Среда разработки предоставляет классы для удобной настройки адаптивности для различных размеров экрана.

Благодаря своей простоте Tailwind CSS заработал большую популярность в среде веб-разработчиков.

1. Drogon C++

Drogon – это среда разработки на C++ для веб-приложений и веб-серверов. Он является быстрым и эффективным инструментом для backend-разработки.

Drogon разработан с учетом асинхронной и многопоточной обработки запросов, что позволяет обрабатывать большое количество запросов одновременно и эффективно использовать многоядерные процессоры.

Среда разработки предоставляет мощный механизм маршрутизации, который позволяет определить, какие обработчики вызывать для разных URL-адресов. Обработчики принимают HTTP-запрос и высылают HTTP-ответ. В них реализуется логика работы веб-приложения. Для всего этого существуют специальные классы библиотеки Drogon – контроллеры.

Для генерации HTML-страниц Drogon предоставляет механизмы обработки шаблонов (views – отображения), что упрощает создание динамических веб-сайтов.

В Drogon встроена библиотека для удобной работы с JSON, а также существует возможность легко создать HTTP-ответ, содержащий в себе JSON-код.

Drogon включает в себя интеграцию с различными системами управления базами данных, такими как PostgreSQL, MySQL, SQLite и другими.

Drogon – мощный инструмент для разработки веб-приложений на C++, который обеспечивает высокую производительность и гибкость. Он может быть особенно полезен в случаях, когда вам нужно создать быстрое и эффективное веб-приложение на C++.

1. Общие положения о выполнении задания

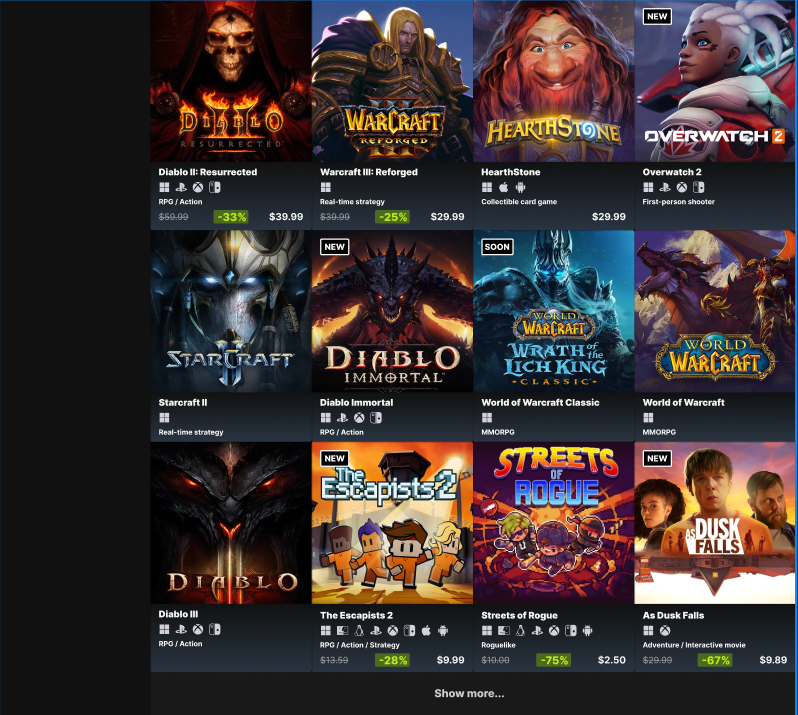


Рисунок 1. Шаблон веб-страницы в Figma

Необходимо разработать проект веб-приложения магазина игр «CardsView». Шаблон внешнего вида сайта для экрана шириной 1920 пикселей представлен в Figma (рисунок 1).

Веб-страница представляет из себя список карт игр, расположенных в ячейках сетки. Изначально загружается ограниченное число игр, затем при прокрутке вниз либо при нажатии на темную кнопку с серым текстом «Show more…» загружается следующее число, и так до тех пор, пока не будут загружены все доступные игры. Загрузка выполняется так, чтобы после нее не возникало пустых ячеек в конце. Страница имеет темный фон.

Карта игры состоит из изображения игры с ее логотипом снизу, поверх которых сверху слева может быть расположено небольшое темное поле со светлой границей с дополнительной информацией («NEW», «SOON» и т.п.), текст которой светлого цвета. При наведении курсора на карту изображение плавно заменяется на видео, логотип и дополнительная информация остаются на месте.

Ниже расположено поле информации об игре с темным градиентом на фоне. Это поле состоит из (по порядку сверху вниз):

1. Названия игры, текст которого светлого цвета;
2. Меню доступных платформ и магазинов для загрузки, подробное описание которого будет дано ниже;
3. Списка жанров игры, текст которого светлого цвета;
4. Поля цены (если она есть), подробное описание которого будет дано ниже.

При первоначальной загрузке на картах отображено меню доступных платформ. Платформы отображены соответствующими значками серого цвета, расположенными в линию по горизонтали. При наведении курсора на значок он плавно светлеет, а ниже него плавно появляется темное поле со светлой границей и светлым текстом – названием платформы. При нажатии на значок платформы меню заменяется на меню доступных магазинов с этой игрой для данной платформы с аналогичным дизайном. При нажатии на значок магазина открывается соответствующая страница. Вместе со значками магазинов в конце расположен значок возврата «Back», при нажатии на который меню вновь заменяется на меню платформ.

Поле цены может отсутствовать, если цена игры не указана. В обратном случае оно состоит из трех секций: старой (без скидки) цены, скидки и текущей (со скидкой, если она есть) цены. Цены указаны в долларах по шаблону *«$<цена>»*, числа имеют два знака после точки. Если отсутствует скидка, первые две секции пусты. Секции расположены по горизонтали. В секции старой цены текст имеет более темный цвет и меньший размер, чем в двух других секциях, а также перечеркнут. Секция скидки имеет зеленый фон и текст скидки в процентах салатового цвета. В последней секции текущей цены текст светлого цвета.

Нажатие на карту в любом месте, кроме иконок меню, переводит пользователя на официальный сайт игры.

Страница сайта должна быть адаптирована под разные размеры экрана. Было решено сделать изменение размера текста линейной интерполяцией в зависимости от ширины экрана. Сетка с картами игр должна изменяться следующим образом:

1. 2 столбца для очень маленьких экранов (меньше 576 пикселей);
2. 3 столбца для маленьких, средних и больших экранов (576-768, 768-992 и 992-1200 пикселей соответственно, не включая вторую границу интервала);
3. 5 столбцов со свободным первым как на изначальном шаблоне для очень больших экранов (1200 и больше пикселей).

Загрузка информации об играх выполняется из базы данных на PostgreSQL. Для этого происходит отправка соответствующего запроса на серверное приложение, которое в ответе возвращает соответствующие данные.

Для корректного распознавания содержимого сайта поисковыми сервисами при изначальном заходе на страницу сервер высылает пользователю статическую страницу, которая содержит в себе список предзагруженных игр. Код этой страницы формируется на сервере только один раз при первом обращении и больше не изменяется.

1. Frontend-часть

Разработка frontend велась на Svelte вместе с Tailwind CSS. Для их установки и работы требуется Node.js.

При разработке использовалась модульная структура проекта.

Базовый размер шрифта на странице задан интерполяцией, в все остальные размеры шрифтов зависят от базового.

**Сетка с играми.** Выше было написано, что карты игр расположены на странице в ячейках сетки. Для возможности отображения левого столбца сетки как пустого в главном модуле *+page.svelte* создан объект с классом *page*, который является grid’ом.

В *page* содержится другой grid-элемент *games*, содержащий карты игр, а также в случае, если загружены не все доступные игры, внизу находится элемент с классом *load-btn* – кнопка для загрузки большего количества игр «Show more…». Этим элементам при помощи классов Tailwind CSS *col-start-x* и *col-span-full* задаются столбец, с которого начинается элемент, и размер до конца сетки.

Параметры *col-start-x* и количества столбцов в сетке (*grid-cols-x*) зависят от размеров экрана. Информация о размерах экрана была внесена в файл *tailwind.config.js*: для *sm*, *md*, *lg* и *xl* – маленького, среднего, большого и очень большого размеров.

Внутри *games* содержатся карточки игр, для хранения информации о которых используется массив *games* с объектами типа интерфейса TypeScript *IGame*, содержащий следующие поля:

1. *link* – ссылка на официальный сайт игры: string;
2. *name* – название игры: string;
3. *image* – ссылка на изображение игры: string;
4. *video* – ссылка на видео игры: string | null;
5. *logo* – ссылка на логотип игры: string;
6. *description* – описание игры (в проекте не используется): string;
7. *additionalMark* – дополнительная информация об игре («NEW», «SOON» и т.п.): string | null;
8. *price* – цена игры: number | null;
9. *discount* – скидка на игру: number | null;
10. *linksToDescriptionStores* – объект ссылок на магазины с игрой типа *«[<магазин>: <ссылка>]…»*: Record<string, string>;
11. *platforms* – массив доступных платформ для игры: string[];
12. *genres* – массив жанров игры: string[].

**Загрузка информации об играх.** Загрузка информации об играх происходит при помощи функции *showMore(needToLoad: number)*, которая, посылая по адресу */getdata* соответствующий POST-запрос (подробнее описано в следующей главе про backend), получает информацию о *needToLoad* играх из базы данных, которые сохраняются в массив *games*. Из backend запрашивается на одну игру больше, чем требуется, для проверки, есть ли еще игры для загрузки (если загружено игр меньше или равно, чем *needToLoad*, то загружены все игры). Идентификатор последней игры, полученной из backend, сохраняется в переменную *fromId* для дальнейшей загрузки следующих игр.

При загрузке страницы пользователем на ней уже содержится информация о части играх, загруженных при формировании статической страницы в backend. В frontend при первой загрузке в случае необходимости загружаются еще игры, чтобы не допустить возникновения пустых мест в конце списка игр (если возможно).

При событиях нажатия на кнопку «Show more…» и прокручивании на нее добавлены обработчики для загрузки еще игр из базы данных. При этом игры также загружаются так, чтобы избежать пустых мест в конце.

**Карты игр.** Для элементов, представляющих из себя карты игр, создан модуль *Card.svelte*, получающий в качестве аргумента объект *game* класса *IGame*, в котором содержится информация об игре. Все прочие модули, которые идут ниже по иерархии от *Card.svelte*, кроме *MenuItem.svelte*,также принимают такой же аргумент *game.*

Элемент карты представляет из себя ссылку (*a*) класса *card*. Ссылка используется для перехода на официальный сайт игры при нажатии на карту (кроме случаев, описанных ниже). Внутри элемента подключены модули *Image.svelte* и *Info.svelte*. Для того, чтобы содержимое *card* занимало все доступное пространство по вертикали (в ячейке сетки карт), *card* задано как flexbox с отображением элементов по вертикали (*flex-col*).

Модуль *Image.svelte* отвечает за отображение изображения, логотипа и дополнительной информации об игре. Представляет из себя элемент с классом *image*, который имеет на фоне изображение (либо видео) игры. Внутри *image* содержатся элементы класса *logo* – логотипа игры, и класса *mark* – дополнительной информации об игре (если она указана). Для задания положения внутренних элементов им задано абсолютное позиционирование (*absolute*).

Модуль *Info.svelte* отображает информацию об игре. Базовый элемент класса *info* содержит элемент названия *name* и модули *Platforms.svelte* (меню платформ и магазинов), *Genre.svelte* (жанры игры) и *Price.svelte* (цена игры и скидка). Для того, чтобы содержимое модуля *Price.svelte* находилось всегда внизу карты (в случае, если на картах на одной строке сетки окажется разный размер занятого пространства по вертикали), элемент является flexbox’ом, а базовому элементу модуля *Price.svelte* задан автоматический внутренний отступ сверху (*mt-auto*).

Модуль *Platforms.svelte* описан в отдельном разделе дальше.

Базовый элемент модуля *Genre.svelte genre* содержит единственную строку, в которой через знак *«/»* перечисляются все указанные жанры игры.

В базовом элементе модуля *Price.svelte price*, представляющий из себя grid из трех столбцов, могут содержаться три элемента классов *old*, *discount* и *now* – старой цены, скидки и новой цены. При помощи класса *col-start-x* каждый из этих элементов находится в соответствующей ячейке сетки. Элементы *old* и *discount* могут отсутствовать при отсутствии скидки, при нулевой цене в *now* будет записано *«Free»*, при отсутствии цены элемент *price* вообще будет отсутствовать и место для цены будет пустое.

При наведении курсора на карту и уведении с ней срабатывают обработчики этих событий, один из которых заменяет фон в элементе *image* на видео игры, а другой возвращает обратно изображение игры.

Для создания особого эффекта появления карты для элемента класса *card* задан параметр перехода *transition:fly* так, чтобы карты плавно возникали немного левее и выше своего места и при этом двигались в свое место.

**Меню платформ и магазинов.** Отображение меню платформ и магазинов реализовано в модуле *Platforms.svelte*. В базовом элементе *main-container* содержатся элементы класса *menu-items*, представляющие из себя соответствующее меню. Каждое такое меню имеет свой идентификатор типа *«main-menu»* для меню платформ и *«<платформа>-menu»* для меню магазинов для соответствующих платформ. Идентификаторы нужны для того, чтобы определять, какое меню открыто у пользователя. Изначально отображается только элемент с идентификатором *«main-menu»*, а остальные скрыты, то есть изначально пользователю видны меню платформ. При переходе в другое меню текущее скрывается, а выбранное показывается.

В меню платформ содержатся иконки платформ, доступных данной игре, при нажатии на которые происходит плавный переход в меню магазинов для соответствующей платформы, где содержатся иконки магазинов, доступных для данной игры, при нажатии на которые происходит переход на соответствующую страницу игры в магазине, а также кнопка «Back» для возврата в первоначальное меню.

Внутренние элементы *menu-items* – кнопки меню, реализованы в модуле *MenuItem.svelte*, принимающий следующие параметры:

1. *name* – название кнопки: string;
2. *icon* – путь к иконке кнопки: string;
3. *link* – необязательный параметр, в случае присутствия означает, что кнопка является ссылкой (используется для кнопок для магазинов): string | undefined.

Кнопки отображаются по горизонтали, для этого *menu-items* задан как flexbox.

Информация о платформах, соответствующих им названий магазинов, а также ссылки на соответствующие иконки кнопок находится в файле *PlatformsConfig.json*, имеющего следующую структуру:

*{*

*[*

*<платформа>: {*

*"icon": <ссылка на иконку платформы>,*

*"stores": {*

*[*

*<магазин, соответствующий платформе>: {*

*"icon": <ссылка на иконку магазина>*

*}*

*]...*

*}*

*}*

*]...*

*}.*

Базовый элемент *MenuItem.svelte* класса *menu-item* имеет на фоне иконку кнопки. Также элементу задана небольшая прозрачность для создания эффекта засветления при наведении курсора. Внутри этого элемента находится элемент класса *item-info* – поля, высвечивающегося при наведении курсора на иконку (поэтому изначально скрытого). Элементы *menu-item* имеют идентификаторы, совпадающие с названием кнопки, это нужно для определения, что нужно сделать при нажатии.

Для событий наведения и уведения курсора и нажатия для кнопок заданы соответствующие обработчики. Для предотвращения перехода на официальный сайт игры при нажатии на данные кнопки (вследствие нажатия на карту игры) в соответствующем обработчике вызываются методы для события *preventDefault* и *stopPropagation*.

1. Backend-часть

Как известно, разработка backend-части веб-приложения велась в среде разработки Drogon C++. Для этого при помощи утилиты drogon\_ctl был создан проект под названием *backend*.

Для связи backend и frontend проект на Svelte был собран при помощи плагина adapter-static и файлы, возникшие после этого в директории *build/*, кроме *index.html*, были скопированы в папку с исполняемым файлом серверного приложения на Drogon.

**База данных и мультимедийные файлы.** Информация об играх хранится в базе данных *CardsView* на PostgreSQL. Эта база данных содержит единственную таблицу *cards\_view*, содержащую следующие столбцы:

1. *id*: integer, not null, auto increment;
2. *link* – ссылка на официальный сайт игры: text, not null;
3. *name* – название игры: text, not null;
4. *image* – ссылка на изображение игры: text, not null;
5. *video* – ссылка на видео игры: text;
6. *logo* – ссылка на логотип игры: text;
7. *description* – описание игры (в проекте не используется): text, not null;
8. *additional\_mark* – дополнительная информация об игре («NEW», «SOON» и т.п.): text;
9. *price* – цена игры: real;
10. *discount* – скидка на игру: real;
11. *links\_to\_description\_stores* – массив ссылок на магазины с игрой типа *«[<магазин>, <ссылка>]…»*: text[], not null;
12. *platforms* – массив доступных платформ для игры: text[], not null;
13. *genres* – массив жанров игры: text[], not null.

Соответствующие мультимедийные файлы для игр хранятся на сервере по ссылке */static/images/*. Для этого они были помещены в соответствующую директорию относительно исполняемого файла серверного приложения.

**Сервер для получения данных из базы данных.** В предыдущей главе упоминалось, что для получения данных об играх из базы данных выполняется POST-запрос по URL */getdata* с параметрами *«fromId»* (начальный идентификатор игры, начиная с которой необходимо вернуть ответ) и *«count»* (число игр, которое необходимо вернуть).

Для этого при помощи drogon\_ctl создан контроллер *GetData*. В этом контроллере реализовано, чтобы при обращении к */getdata* возвращался HTTP-ответ, содержащий JSON-данные с параметрами *finalId* (идентификатор последней игры, который в дальнейшем используется в frontend) и *data* (массив информации об играх в виде объектов с полями, описанными в предыдущей главе).

Для этого при помощи ORM для PostgreSQL происходит подключение к соответствующей базе данных с необходимыми параметрами, а затем формирование на основе параметров POST-запроса соответствующего SQL-запроса. Из полученного ответа при помощи библиотеки *Json* и метода *HttpResponse::newHttpJsonResponse* формируется HTTP-ответ с необходимым содержимым.

**Отправка веб-страницы пользователю при обращении и формирование статической страницы.** Для отправки страницы пользователю при обращении при помощи drogon\_ctlбыл создан контроллер *CardsView*, срабатывающий при отправке GET-запроса по URL */*. Пользователю возвращается статическая веб-страница, содержащая в себе уже предзагруженные данные об играх, которая формируется только при первом обращении, а затем сохраняется для повторных отправок.

Для формирования статической страницы при помощи drogon\_ctlбыло создано отображение *CardsView.csp*, а также модули для него *StaticHtml.csp* и *StaticJs.csp*. *CardsView.csp* использует в своей основе файл *index.html* из frontend, в который в начало *body*подключен модуль *StaticHtml.csp*, а в начало *script* – *StaticJs.csp*. Путем подстановки данных из базы данных в *body* возникает первоначальное содержимое страницы, которое будет доступно поисковым сервисам, а путем подстановки данных в *script* в frontend в дальнейшем формируется внешний вид страницы для пользователя.

ЗАКЛЮЧЕНИЕ

В процессе прохождения практики были получены навыки в разработке веб-приложений.

Были изучены как базовые языки разработки frontend: HTML, CSS и JavaScript, так и среды разработки Svelte и Tailwind CSS, а также расширение для JavaScript – TypeScript.

Используя Svelte вместе с Tailwind CSS и TypeScript, была реализована frontend-составляющая проекта «CardsView» (рисунки 2, 3) – игрового веб-магазина, включая визуальную, интерактивную составляющие и адаптивность под различные размеры экрана.

|  |  |  |
| --- | --- | --- |
| а. | б. | в. |

Рисунок 2. Вид веб-страницы для различных размеров экрана

|  |  |
| --- | --- |
| а. | б. |

Рисунок 3. Демонстрация интерактивных элементов карт

Для загрузки данных об играх в backend разработано серверное приложение, используя среду разработки Drogon C++, через которое происходит загрузка игр из базы данных на PostgreSQL. Таким образом получены навыки работы с HTTP-запросами и ответами, а также с реляционными СУБД.

Полученные знания будут полезны в разработке дальнейших проектов.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Svelte.dev: официальный сайт. – URL: <https://svelte.dev/> (дата обращения: 11.07.2023).
2. Tailwindcss: официальный сайт. – URL: <https://tailwindcss.com/> (дата обращения: 06.07.2023).
3. Html academy: сайт. – URL: <https://htmlacademy.ru/> (дата обращения: 03.07.2023).
4. TypeScript: официальный сайт. – URL: <https://www.typescriptlang.org/> (дата обращения: 14.07.2023).
5. Drogon wiki: официальная документация. – URL: <https://github.com/drogonframework/drogon/wiki> (дата обращения: 28.07.2023).