РОССИЙСКАЯ ФЕДЕРАЦИЯ
ХАНТЫ-МАНСИЙСКИЙ АВТОНОМНЫЙ ОКРУГ-ЮГРА

ВТОНОМНАЯ НЕКОММЕРЧЕСКАЯ ПРОФЕССИОНАЛЬНАЯОБРАЗОВАТЕЛЬНАЯ ОРГАНИЗАЦИЯ
«СУРГУТСКИЙ ИНСТИТУТ ЭКОНОМИКИ, УПРАВЛЕНИЯ И ПРАВА»

ОТЧЕТ
по учебной практике

УП.09 ПРОЕКТИРОВАНИЕ, РАЗРАБОТКА И ОПТИМИЗАЦИЯ ВЕБ-ПРИЛОЖЕНИЙ

(наименование типа практики)

Специальность: 09.02.07 Информационные системы и программирование

обучающийся ИС 23/9-1 группы
Хазипов Максим Владиславович
(фамилия, имя, отчество (при наличии))
_____ «___» _____ 20___ г.
(личная подпись, дата)

Проверил руководитель практики
от образовательной организации:
Преподаватель профессионального модуля
(уч. степень, уч. звание, должность)
Мацюк Никита Александрович
(фамилия, имя, отчество (при наличии))
Оценка _____
_____ «___» _____ 20___ г.
(личная подпись, дата)

Сургут, 2025г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современный этап развития цифровых технологий характеризуется повсеместной цифровизацией и трансформацией рынка развлечений. В этом контексте показателен бум рынка настольных игр, который из нишевого увлечения превратился в технологичную и социально значимую индустрию с многомиллиардными оборотами. Данная учебная практика посвящена разработке инструмента для этого рынка — специализированного онлайн-магазина, и сосредоточена на одном из самых важных аспектов создания любого веб-приложения: проектировании и взаимодействии с базой данных.

Актуальность выбранной темы обусловлена ростом популярности настольных игр как формы досуга. «Цифровая усталость» стала катализаторами бума в этой области. Потребитель стал искушенным, он ищет решение — игру, которая подойдет под конкретный состав компании, уровень сложности, временной бюджет и тематические предпочтения. В этих условиях универсальные маркетплейсы зачастую проигрывают специализированным платформам, способным предложить глубокий каталог, описания, систему рекомендаций и фильтрации. Таким образом, создание онлайн-магазина, ориентированного на специфические потребности данной аудитории, является ответом на реальный и растущий рыночный запрос.

Основной целью данной учебной практики является освоение цикла backend-разработки веб-приложения на фреймворке Django с изучением логики работы с реляционными базами данных. Практика направлена на превращение теоретических знаний о моделях, запросах и архитектуре веб-приложений в практические навыки. Для достижения этой цели в работе решается ряд задач: анализ предметной области и проектирование логической схемы базы данных; реализация этой схемы с использованием Django ORM; разработка бизнес-логики для извлечения, фильтрации, сортировки и обновления данных; создание базового пользовательского интерфейса для

визуализации этих данных и взаимодействия с ними; интеграция всех компонентов в единое работоспособное приложение.

Объектом изучения и разработки выступает процесс создания динамического веб-сайта. Предметом является комплекс методов и инструментов для проектирования структуры данных, манипулирования ими на серверной стороне и их представления в браузере пользователя, реализуемый средствами стека технологий Python, Django и SQL.

Методологическую основу работы составили принципы объектно-ориентированного и структурного программирования, методология CRUD для управления данными, а также подход MVT. В качестве основного инструментария был выбран фреймворк Django благодаря его набору встроенных компонентов, строгой архитектуре и мощной ORM, которая позволяет абстрагироваться от написания SQL и работать с базой данных на языке Python. Для этапа разработки в качестве системы управления базами данных была выбрана SQLite.

Практическая значимость выполненной работы заключается в создании готового ядра полноценного интернет-магазина. Принципы проектирования моделей, построения сложных запросов и разделения логики между слоями приложения являются универсальными

1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Описание предметной области

Современный рынок настольных игр превращается в массовый вид досуга. По расчетам Market Research Future, в 2024 году объем мирового рынка настольных игр составлял $13,5 млрд. По прогнозам Statista, в 2025 году глобальный рынок принесет доход в размере $8,95 млрд (рост на 3,7% по сравнению с 2024 годом), включая как онлайн-, так и офлайн-каналы продаж. Российский рынок Hobby World оценивает в ₽32 млрд, GaGa Games — ₽25–40 млрд.

Этот рост обусловлен несколькими ключевыми факторами:

- После пандемии многие люди ищут альтернативу гаджетам, ценя «живое» общение.
- Настольные игры становятся популярным форматом для встреч с друзьями и семьей.
- Появление клубов, кафе и мероприятий, посвященных настольным играм, способствует популяризации такого хобби.
- Игры создаются для самых разных групп людей.

В условиях такой высокой конкуренции создание специального онлайн-магазина поможет в привлечении аудорим. Потребители ищут подробную информацию, сравнивают характеристики, читают отзывы и обзоры перед покупкой. Онлайн-магазин решает ряд задач, предоставляя клиенту детальную информацию о товаре, удобный поиск, охват большой аудитории и обеспечивая удобное управление ассортиментом. Таким образом, разработка онлайн-магазина настольных игр является ответом на вызовы современного рынка.

1.2 Логическая схема БД

Для хранения данных в веб-проекте используется Система Управления Базами Данных (СУБД). СУБД напрямую влияет на производительность, масштабируемость и надежность приложения. В рамках данной учебной практики для разработки была выбрана SQLite. Ключевыми причинами выбора именно SQLite были:

- Встроенна поддержка в Django. SQLite – СУБД по умолчанию для проектов Django. Фреймворк полностью настроен для работы с ней, это позволяет приступить к разработке логики приложения не тратя времени на сложную настройку и конфигурацию сервера БД.
- Идеальная среда для разработки и тестирования. Вся база данных хранится в одном файле на диске, что максимально упрощает начало работы. Файл базы данных легко создать, скопировать, удалить или перенести, что удобно при активном изменении структуры моделей.
- Соответсвие требованиям Учебной Практики. Основной задачей учебной практики было освоение логики работы с реляционной базой данных через Django ORM, а SQLite поддерживает стандартный SQL и основные возможности ORM Django.

Таким образом была спроектирована база данных, структура которой. будет приведена ниже.

Модель BoardGame – центральная сущность системы, представляющая собой настольную игру. Её атрибуты: name (название), price (цена), description (описание), players_number (количество игроков), game_time (время игры), min_age (минимальный возраст), game_image (изображение игры), add_date (дата добавления), popularity (популярность). Релизованна связь многие-ко-многим с моделью Genre, которое имеет только атрибут name с названием. Таким образом у одной игры может быть несколько жанров.

Profile хранит информацию о клиенте. Атрибуты: first_name (имя), last_name (фамилия), middle_name (отчество), email (адрес электронной

почты), phone_number (номер телефона), birthday (дата рождения). В модели релизованна связь один-к-одному с моделью BounsCards через атрибут bonus_card_id. Связи многие-ко-многим представляются атрибутами cart_items и order, которые проходят через промежуточные таблицы, Cart и Orders соответственно.

Модель BonusCards для управления бонусными картами. Атрибуты: balance (баланс), percentage_of_return (процент возврата), amount_of_purchases (общая сумма всех покупок).

Orders и Cart две очень похожие модели, но хранящие данные о разных дейсвтиях. Orders – это заказ, а Cart – это корзина. У этих моделей одинаковые атрибуты: profile_id (id профиля), board_game_id (id настольной игры) и count (количество игр).

Модель Purchased хранит в себе историю покупок. В ней хранятся данные о profile_id (id профиля), board_game_id (id игры), amount (количество игр), cost (стоимость игр), bonus_amount (количество бонусных баллов). Данная логическая схема приведена ниже не рисунке 1.

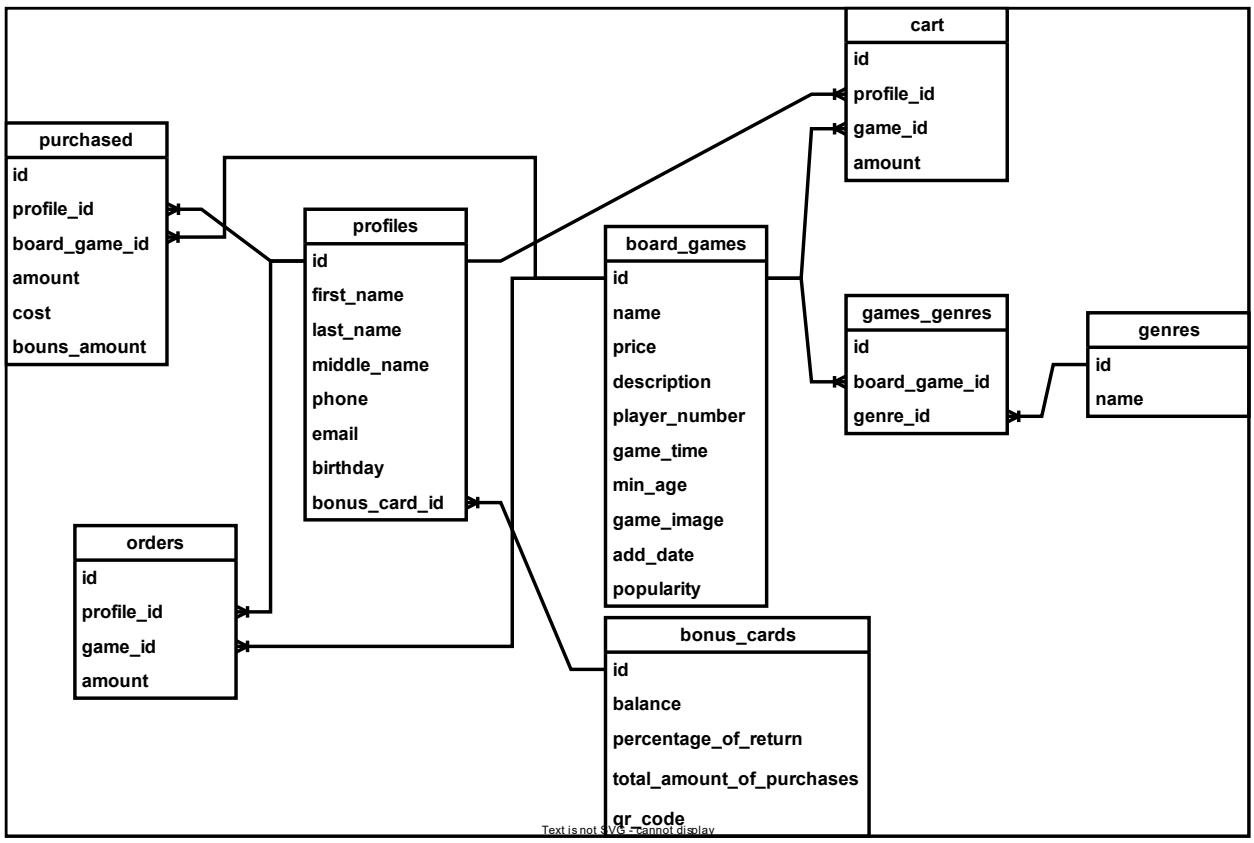


Рисунок 1 – Логическая схема базы данных магазина настольных игр

1.3 Список URL страниц сайта

В Django реализованна система маршрутизации. Маршрутизация (URL dispatching) – фундаментальный механизм любого веб-фреймворка, который отвечает за то какая часть кода должна выполниться при обращении к определенному URL-адресу. В текущей работе тоже был реализован такой механизм. Список URL страниц представлен на рисунке 2.

```
urlpatterns = [
    path('', views.root),
    path('game-pages/<int:item_id>', views.page),
]
```

Рисунок 2 – Список URL страниц

Из рисунка видно, что маршуртизация происходит с помощью функции path. Первым аргументом в функцию подается шаблон url, а вторым функция отображения из views. Первый шаблон обрабатывает запрос к главной странице сайта. Второй же запрос обеспечивает отображение страницы товара. Во втором случае `<int:item_id>` является частью url, который передается в функцию page в качестве аргумента.

Таким образом данные паттерны связывают URL с логикой веб-приложения. Паттерны позволяют: обрабатывать статические пути; извлекать данные из динамических путей; Передавать параметры в view-функции.

2 ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Django ORM. Модели

Django ORM (Object-Relational Mapping) — это компонент фреймворка, основная задача которого обеспечить взаимодействие с реляционной базой данных, используя синтаксис Python, без необходимости написания SQL-кода. Это повышает безопасность, ускоряет разработку и обеспечивает переносимость между различными СУБД. Ключевым принципом является то, что Каждый класс модели в Django отображается на таблицу в базе данных, а его атрибуты — на столбцы этой таблицы.

Трансформация класса Python в таблицу БД осуществляется самим фреймворком наследованием от базового класса `models.Model`. Это наделяет класс метаданными, которые Django использует для построения соответствующей структуры в базе данных. Атрибуты объявляются как экземляры классов полей. Каждый атрибут модели, которому присвоен экземпляр класса поля (`CharField`, `IntegerField` и др.), становится столбцом в таблице. Параметры поля определяют его свойства в БД: тип, ограничения, значение по умолчанию и т.д. При этом можно не указывать первичный ключ. Если явно не объявить ключ, Django автоматически добавляет в модель поле, это поле с автоинкрементом, которое гарантирует уникальную идентификацию каждой записи в таблице. Имя таблицы в базе данных генерируется автоматически по шаблону: <имя_приложения>_<имя_модели_в_нижнем_регистре>. Это поведение можно переопределить в классе Meta модели. Связи между моделями также определяются через поля: ForeignKey (внешний ключ), ManyToManyField (многие-ко-многим) и OneToOneField (один-к-одному). Для управления схемой БД Django использует систему миграций для приведения структуры базы данных в соответствие с изменяющимися моделями. Управление происходит с помощью следующих команд:

- makemigrations – анализирует текущие модели и создает файлы миграций – инструкции по изменению схемы БД.
- Migrate – применяет эти инструкции к реальной базе данных, выполняя необходимые SQL-запросы.

Ниже представлен разбор реализованных моделей. Во всех моделях Класс Meta с verbose_name и verbose_name_plural обеспечивает удобное отображение таблиц в интерфейсе администратора.

Модель BoardGames – центральная сущность системы, хранит информацию о настольных играх. Для названия игры выбран CharField с max_length=35 и unique=True, что обеспечивает краткое и уникальное наименование. Цена определена как DecimalField с параметрами max_digits=20 и decimal_places=2 для определения цена до двух знаков после запятой. Связь «многие-ко-многим» с жанрами через ManyToManyField позволяет одной игре принадлежать к нескольким категориям. Поле game_image использует ImageField с upload_to='media/' для хранения графических файлов. Параметры default для add_date и popularity задают начальные значения. На рисунке 3 представлена реализация данной модели. Все реализации будут представлены без класса Meta.

```
class BoardGames(models.Model):
    name = models.CharField('Название игры', max_length=35,
unique=True)
    price = models.DecimalField('Цена', max_digits=20,
decimal_places=2)
    description = models.TextField('Описание игры')
    players_number = models.CharField('Количество игроков',
max_length=6)
    game_time = models.CharField('Среднее время игры',
max_length=30)
    min_age = models.IntegerField('Минимальный возраст')
    games_genres = models.ManyToManyField('Genres')
    game_image = models.ImageField('Картинка игры',
upload_to='media/')
    add_date = models.DateField("Дата добавления", default='2025-01-
01')
    popularity = models.IntegerField('Популярность', default=0)
```

Рисунок 3 – Модель настольной игры

Модель Genres – это справочник категорий игр. Её единственное поле name (CharField с max_length=35 и unique=True) хранит название жанра, что гарантирует отсутствие дубликатов. На рисунке 4 представлена реализация данной модели.

```
class Genres(models.Model):
    name = models.CharField('Название жанра', max_length=35,
unique=True)
```

Рисунок 4 – Модель жанров

Модель Profiles хранит данные о пользователях. Для хранения личной информации (имя, фамилия, отчество) используются поля CharField с соответствующей максимальной длиной. Поле bonus_card_id реализовано как ForeignKey с параметром unique=True, что фактически устанавливает связь «один-к-одному» с моделью BonusCards, гарантируя, что у каждого профиля есть уникальная бонусная карта. Связи order и cart_items через ManyToManyField с аргументом through указывают на промежуточные модели Orders и Cart для истории заказов и корзины покупок соответственно. На рисунке 5 представлена реализация данной модели.

```
class Profiles(models.Model):
    first_name = models.CharField('Имя', max_length=40)
    surname = models.CharField('Фамилия', max_length=40)
    middle_name = models.CharField('Отчество', max_length=40,
null=True)
    email = models.EmailField('Электронная почта', max_length=50,
null=True)
    phone_number = models.CharField('Телефон', max_length=50,
null=True)
    birthday = models.DateField('Дата рождения')
    bonus_card_id = models.ForeignKey('BonusCards',
verbose_name='Бонусная карта', on_delete=models.CASCADE)
    order = models.ManyToManyField('BoardGames', through='Orders',
through_fields=('profile_id', 'board_game_id'))
    cart_items = models.ManyToManyField('BoardGames',
through='Cart', through_fields=('profile_id', 'board_game_id'),
related_name='in_cart')
```

Рисунок 5 – Модель профиля

Модель BonusCards управляет программой лояльности. Поле balance (IntegerField) хранит бонусные баллы, percentage_of_return (CharField) —

процент возврата, а amount_of_purchases (DecimalField) — накопленную сумму покупок. DecimalField используется для точности хранимых данных о финансах. На рисунке 6 представлена реализация данной модели.

```
class BoardGames(models.Model):
    name = models.CharField('Название игры', max_length=35,
unique=True)
    price = models.DecimalField('Цена', max_digits=20,
decimal_places=2)
    description = models.TextField('Описание игры')
    players_number = models.CharField('Количество игроков',
max_length=6)
    game_time = models.CharField('Среднее время игры',
max_length=30)
    min_age = models.IntegerField('Минимальный возраст')
    games_genres = models.ManyToManyField('Genres')
    game_image = models.ImageField('Картинка игры',
upload_to='media/')
    add_date = models.DateField("Дата добавления", default='2025-01-
01')
    popularity = models.IntegerField('Популярность', default=0)
```

Рисунок 6 – Модель настольной игры

Модель Orders – промежуточная таблиця для связи «многие-ко-многим» между профилями и играми, фиксируя факт заказа. Она содержит внешние ключи (ForeignKey) на Profiles и BoardGames, а также поле amount для указания количества товара. Такая структура позволяет одному профилю оформлять несколько заказов с разными играми. Модель Cart также служит промежуточной таблицей для связи профилей и игр с такой же структурой, но показывает текущее состояние корзины. На рисунке 7 представлена реализация данной модели.

```
class BoardGames(models.Model):
    name = models.CharField('Название игры', max_length=35,
unique=True)
    price = models.DecimalField('Цена', max_digits=20,
decimal_places=2)
    description = models.TextField('Описание игры')
    players_number = models.CharField('Количество игроков',
max_length=6)
    game_time = models.CharField('Среднее время игры',
max_length=30)
    min_age = models.IntegerField('Минимальный возраст')
    games_genres = models.ManyToManyField('Genres')
    game_image = models.ImageField('Картинка игры',
upload_to='media/')
    add_date = models.DateField("Дата добавления", default='2025-01-
01')
    popularity = models.IntegerField('Популярность', default=0)
```

Рисунок 7 – Модель настольной игры

Модель Purchased предназначена для хранения истории совершённых покупок. Имеет связи с профилем и игрой (ForeignKey), поле с количеством, ценой (DecimalField) и bonus_amount для списанных или начисленных бонусов. На рисунке 8 представлена реализация данной модели.

```
class BoardGames(models.Model):
    name = models.CharField('Название игры', max_length=35,
unique=True)
    price = models.DecimalField('Цена', max_digits=20,
decimal_places=2)
    description = models.TextField('Описание игры')
    players_number = models.CharField('Количество игроков',
max_length=6)
    game_time = models.CharField('Среднее время игры',
max_length=30)
    min_age = models.IntegerField('Минимальный возраст')
    games_genres = models.ManyToManyField('Genres')
    game_image = models.ImageField('Картинка игры',
upload_to='media/')
    add_date = models.DateField("Дата добавления", default='2025-01-
01')
    popularity = models.IntegerField('Популярность', default=0)
```

Рисунок 8 – Модель настольной игры

Спроектированные модели Django ORM отражают логическую схему данных, необходимую для работы интернет-магазина настольных игр. Каждая

сущность получила представление в виде отдельной модели с соответствующими атрибутами. Типы полей были выбраны корректно: DecimalField для финансовых данных, CharField с ограничениями длины обеспечивает хранение строковой информации, а ImageField организует работу с изображениями. Связи между моделями реализованы через ForeignKey и ManyToManyField. Наличие класса Meta с упрощает администрирование системы. Таким образом, созданная структура базы данных является фундаментом для реализации всего функционала, включая фильтрацию, поиск и сортировку.

2.2 Создание дизайна сайта

Базовый шаблон main.html является фундаментальной основой для всех страниц веб-сайта интернет-магазина настольных игр. Его главная роль заключается в обеспечении единообразия визуального представления и навигации на всем сайте. Шаблон реализует принцип “не повторяй себя” через систему наследования Django, определяя неизменяемые структурные элементы — шапку и подвал — и предоставляя дочерним шаблонам блоки ({% block style %} и {% block main %}) для вставки уникального контента и стилей. Шапка сайта представляет собой многоуровневую навигационную структуру. Визуально и структурно она разделена на три блока, каждый из которых решает определенный круг задач. Первый, верхний блок, выполненный в темно-коричневом цвете, содержит служебную информацию: указание геолокации с иконкой карты, ссылки на разделы офлайн-магазинов и условий доставки, а также контакты и форму обратной связи. Этот блок, стилизован шрифтом Lancelot. Второй, центральный блок шапки, является ядром навигации. Он объединяет логотип магазина, кнопку вызова каталога, оформленную шрифтом Leckerli One, полноразмерную поисковую строку и панель быстрого доступа пользователя. Поисковая строка реализована как элемент <input type="search"> со стилизованной кнопкой-лупой,

использующей background для вставки иконки. Правая часть этого блока содержит панель иконок для входа в аккаунт, просмотра заказов, избранного и корзины. Каждая иконка сопровождается текстовой подписью. Данный блок отделен использованием более светлого оттенка коричневого и обеспечивает мгновенный доступ ко всем ключевым действиям на сайте. Третий, нижний навигационный блок шапки представляет собой горизонтальное меню, которое организует доступ к разделам и активностям магазина. Элементы меню, такие как Promotion, Sales, Guide, What should I play?, Gift certificates и Play journal, группируют контент, помогая сориентироваться в дополнительных услугах. Акцент сделан на элементе Online, он стилизован под интерактивный ярлык с отдельным фоновым изображением. Все элементы выровнены с помощью Flexbox. Дизайн шапки будет представлен на рисунке 9. Основная часть страницы заключена в тег <main> и представляет собой изменяемую область. Ее содержимое определяется дочерними шаблонами, которые заполняют блок {% block main %}.

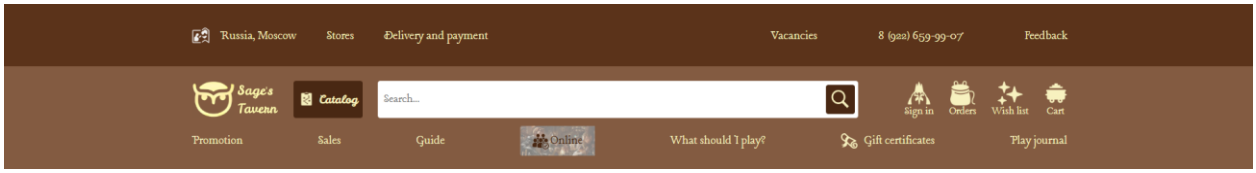


Рисунок 9 – Дизайн шапки

Подвал сайта выполняет информационную и справочную функцию, завершая композицию страницы. Он визуально выделен отличающимся цветом и разделен на три колонки с использованием Flexbox в контейнере footer-wrapper. Левая колонка содержит название магазина и иконоки-ссылки на основные социальные сети стилизованные в виде горизонтального списка. Центральная часть, разделенная на два подраздела, предоставляет расширенную текстовую навигацию по ключевым разделам сайта. В нижней части подвала, отделенной горизонтальной линией, расположена юридическая информация. Шрифт подвала - Fjalla One. Стилизация реализована в отдельных CSS-файлах. Сброс стилей браузера выполняется в файле null.css. Основное оформление шапки и подвала вынесено в файл header-footer.css.

Ключевую роль в дизайне играет цветовая палитра, построенная на теплых, «деревянных» оттенках коричневого, которые ассоциируются с фэнтезийной таверной. Акцентные цвета направляют внимание пользователя. Для построения сложных макетов используется Flexbox. Контейнер с фиксированной шириной центрирует контент и задает границы макета. Дизайн подвала будет приведен на рисунке 10.

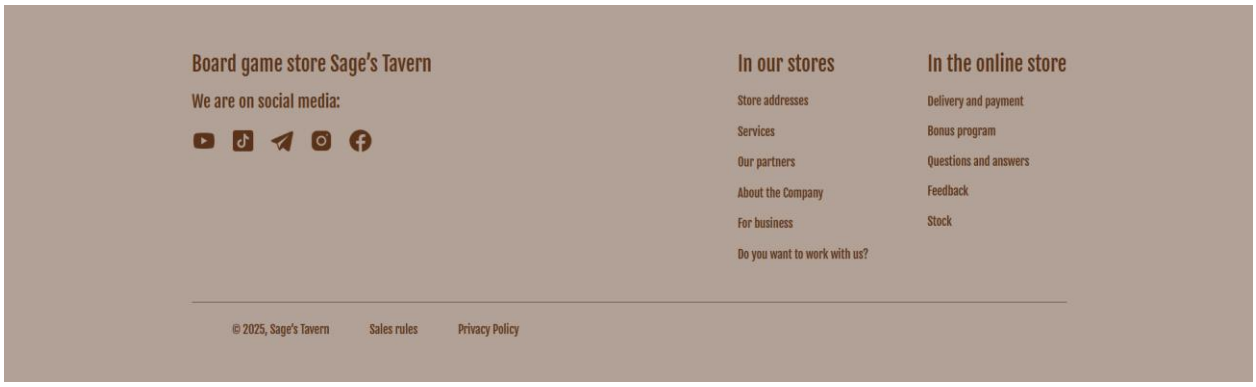


Рисунок 10 – Дизайн подвала сайта

Шаблон main-content.html, является наследником базового шаблона main.html, формирует ядро главной страницы. Его основная задача — предоставить пользователю инструмент для взаимодействия с каталогом товаров, элементы с системой фильтрации, сортировки и представления сетки товаров. Полный дизайн главной страницы представлен в приложении к курсовой работе. Этот шаблон демонстрирует интеграцию backend-логики с динамическим фронтендом. Открывает страницу верхний блок. Слева размещено крупное изображение. Справа расположены две вертикальные информационные карточки, стилизованные под элементы игры. Первая карточка, с иконкой пламени, анонсирует лучшие игры сезона. Вторая, с иконкой игральных костей, показывает раздел "Таверна" — сообщество для общения и обзоров. Карточки оформленны с использованием тонкой рамки и основных цветов сайта. Дизайн верхнего блока с изображением и карточками будет представлен на рисунке 11.

Рисунок 11 – Дизайн верхнего блока главной страницы

Центральная часть страницы представляет собой двухколоночный макет, где левая колонка отведена под систему фильтрации по жанрам. Этот блок, озаглавленный "Store", содержит формируемый список всех доступных жанров, полученных из контекста. Каждый жанр отображается в виде элемента списка с маркером в виде иконки. Ключевая особенность реализации - это выделение активного фильтра: при совпадении названия жанра с параметром GET-запроса маркер и цвет текста меняются. Клик по жанру обновляет URL, добавляя или изменяя параметр genres, при этом сохраняя текущие параметры поиска и сортировки. Релизация данного переключение представленна на рисунке 12 .

```
{% for genre in all_genres %}
<li class="tags_list-item" {%if request.GET.genres == genre.name%}
style="background-image:url(../static/media/-active.svg);"{%endif%}>
<a
href="?genres={{genre.name}}&search={{request.GET.search}}&sort={{re
quest.GET.sort}}"{%if request.GET.genres == genre.name%}
style="color: #CE895D;" {%endif%}>{{genre.name}}</a>
</li>
{% endfor %}
```

Рисунок 12 – Релизация переключения цветов при выборе категории

Правая, основная колонка, состоит из панели управления и сетки товаров. Панель управления включает в себя два инструмента. Первый — выпадающий список сортировки с восемью вариантами: по дате добавления, цене, алфавиту и популярности. Изменение выбора отправляет форму, перестраивая выдачу. В форме скрыто передаются текущие параметры фильтра и поиска, что позволяет комбинировать их с сортировкой. Второй

инструмент — форма текстового поиска с полем ввода и кнопкой в виде лупы. Логика шаблона анализирует наличие параметров и отображает соответствующие кнопки сброса — "Reset all", "Reset search" или "Reset genres", что дает пользователю контроль над запросом. Реализация логики появления кнопок будет представлена на рисунке 13 .

```
{% if genres_req and query %}
<a href="/?sort={{request.GET.sort}}" class="reset_button">Reset
all</a>
<a href="/?sort={{request.GET.sort}}&genres={{request.GET.genres}}"
class="reset_button">Reset search</a>
<a href="/?sort={{request.GET.sort}}&search={{request.GET.search}}"
class="reset_button">Reset genres</a>
{% elif genres_req%}
<a href="/?sort={{request.GET.sort}}" class="reset_button">Reset
genres</a>
{% elif query%}
<a href="/?sort={{request.GET.sort}}" class="reset_button">Reset
search</a>
{% endif %}
```

Рисунок 13 – Релизация логики появления кнопок сброса

Сетка товаров динамически рендерит карточки для каждого объекта board_game, переданного в контексте. Использование Grid Layout создает, адаптивную трехколоночную структуру. Каждая карточка товара содержит изображение игры. Ниже располагаются название игры и цена, набранные отличающимися шрифтами разного размера. Важным элементом карточки является кнопка "Buy" зеленого цвета, которая ведет на уникальную страницу товара (game-pages/{{ board_game.id }}). Дополнительная ссылка "more" предлагает альтернативный путь к деталям. Карточка имеет белый фон, тень и рамку, что визуально отделяет ее от фона и создает эффект глубины. Шаблон также обрабатывает сценарий поиска: если есть поисковый запрос (query), выводится соответствующий заголовок, а при отсутствии результатов — сообщение "Nothing...". Все элементы сделаны в шрифте "Fjalla One". Для построения сложных макетов используются Flexbox и CSS Grid (для сетки товаров). Отступы создают "воздух" и облегчают восприятие. Дизайн каталога будет приведен на рисунке 14.

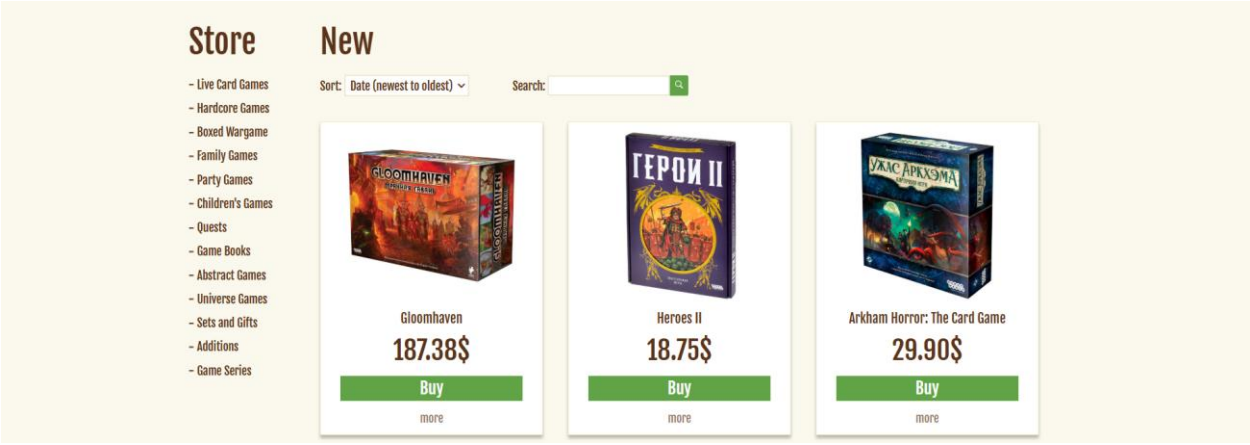


Рисунок 14 – Дизайн каталога

Шаблон game-page.html предназначен для отображения детальной информации о конкретной настольной игре. Его основная задача заключается в предоставлении пользователю полной информации о товаре. Заголовок представлен крупным элементом <h1 class="game-title">, который отображает название игры, переданное из представления. Использование большого размера шрифт и коричневого цвета акцентирует внимание на главном объекте страницы. Основной информационный блок разделен на две колонки с использованием Flexbox. Левая колонка занимает изображение игры. Правая колонка, занимающая 46.6% ширины блока, посвящена ключевым характеристикам. Верхнюю часть этой колонки занимает блок который объединяет отображение цены и инструмент выбора количества. Цена игры представлена в крупном формате. Рядом расположен счетчик количества, технически реализованный как поле ввода типа number, дополненное двумя кнопками. Кнопки "+" и "-" визуально отличаются по цвету зеленый для увеличения, красный для уменьшения. Для поля ввода убраны стандартные браузерные стрелки, заданы внутренние отступы и тень. Под блоком с ценой и количеством расположена крупная кнопка "Buy". Ее дизайн, использует зеленый цвет фона и белый цвет текста. Нижнюю часть правой колоннки занимает блок с техническими характеристиками игры. Он реализован как двухколоночный макет внутри Flex-контейнера, где левая колонка содержит названия параметров с соответствующими иконками, а правая — их значения, подставляемые из объекта game. Отображаются четыре ключевых параметра:

количество игроков, время игры, минимальный возраст и язык правил. После основного блока следует секция с полным описанием игры. Эта часть отделена заголовком "Description", который оформлен как вкладка с рамкой, которая не имеет нижней границы, а под ним проходит сплошная горизонтальная линия. Само описание, переданное в переменной {{ game.description }}, набирается крупным шрифтом 24px. Дизайн страницы товара будет приведен на рисунке 15.

Рисунок 15 – Дизайн каталога

2.3 Выборка и фильтрация данных из БД

Логика работы главного представления root. Главным элементом представления основной страницы является выборка всех товаров из базы данных, которая выполняется с помощью метода BoardGames.objects.all(). Данная операция формирует каталог. При выполнении этого кода Django ORM генерирует SQL-запрос, который равняется SELECT * FROM boardgames, и возвращает специальный объект — QuerySet, содержащий все экземпляры модели BoardGames. Важно отметить, что QuerySet является “ленивым”, обращение к базе данных и извлечение записей происходит не в момент

вызова .all(), а лишь при следующей итерации по объекту или явном преобразовании в список.

После получения выборки всех игр представление начинает динамическое формирование итогового QuerySet на основе параметров, переданных пользователем через строку запроса. Это реализует ключевые функции поиска и фильтрации. Извлечение параметров выполняется с помощью метода request.GET.get(), который извлекает значение по ключу, возвращая пустую строку по умолчанию, если параметр отсутствует. На рисунке 15 представлены запросы для извлечения параметров.

```
query = request.GET.get('search', '')
genres_req = request.GET.get('genres', '')
sort = request.GET.get('sort', '')
```

Рисунок 16 – Извлечение параметров

- search_query = request.GET.get('search', ''): получает строку для поиска.
- genre_name = request.GET.get('genres', ''): получает название жанра для фильтрации.
- sort_option = request.GET.get('sort', ''): получает выбранный вариант сортировки.

Затем происходит применение фильтров. Фильтрация по жанру, если передан параметр genres, к QuerySet применяется фильтр с использованием связи «многие-ко-многим». Это приводит к SQL-запросу с JOIN через промежуточную таблицу, выбирая только те игры, которые связаны с указанным жанром. На рисунке 16 представлена фильтрация по жанрам.

```
elif genres_req:
        board_games =
BoardGames.objects.filter(games_genres__name=genres_req)
```

Рисунок 17 – Фильтрация по жанрам

Текстовый поиск передан. Если передан параметр search, выполняется фильтрация по совпадению в названии игры. icontains обеспечивает

регистронезависимое сравнение. На рисунке 17 представлена реализация механики поиска.

```
elif query:
        board_games =
BoardGames.objects.filter(name__icontains=query)
```

Рисунок 18 – Поиск по названию

Если нужно выполнить одновременно поиск и фильтрацию. Если передан параметр search и параметр genres, тогда применяется фильтр по названию и игровым жанрам. На рисунке 18 представлена реализация механики поиска и фильтрации.

```
if query and genres_req :
        board_games =
BoardGames.objects.filter(name__icontains=query,
games_genres__name=genres_req)
```

Рисунок 19 – Поиск по названию и фильтрация по жанру

Для реализации настраиваемой системы сортировки в представлении главной страницы используется конструкция match/case. Этот механизм предоставляет альтернативу цепочкам if/elif/else при обработке множественных вариантов. Принцип работы заключается в сравнении значения параметра sort, полученного из GET-запроса, с заранее определенными шаблонами. В зависимости от совпадения к базовому QuerySet применяется соответствующий метод .order_by() с указанием поля и направления сортировки. Дефис (-) перед именем поля указывает на обратный порядок. Без дефиса — сортировка по возрастанию. На рисунке 19 представлена реализация сортировки.

```
match sort:
     case "date-reverse":
          board_games = board_games.order_by("add_date")
     case "price-asc":
          board_games = board_games.order_by("price")
     case "price-des":
          board_games = board_games.order_by("-price")
     case "alphabet":
          board_games = board_games.order_by("name")
     case "reverse-alphabet":
          board_games = board_games.order_by("-name")
     case "popular":
          board_games = board_games.order_by("-popularity")
     case "reverse-popular":
          board_games = board_games.order_by("popularity")
     case _:
          board_games = board_games.order_by("-add_date")
```

Рисунок 20 – Реализация сортировки

Логика работы представления page. Ключевым элементом представления страницы товара является безопасное извлечение конкретной игры из базы данных с использованием функции get_object_or_404(). Данная функция пытается получить объект модели BoardGames по заданному идентификатору, но если объект с таким id не существует, вместо возникновения исключения функция автоматически возвращает стандартную страницу ошибки 404 (Not Found). Техническая реализация приведена на рисунке 20.

```
game = BoardGames.objects.filter(id=item_id)
game = get_object_or_404(game)
```

Рисунок 21 – Реализация получения конкретной игры

После получения объекта, представление выполняет операцию по обновлению данных этой конкретной записи в базе данных. Механизм обновления реализуется в два этапа:

- К текущему значению числового поля popularity объекта game прибавляется единица: game.popularity += 1. Эта операция происходит в памяти Python, изменяя атрибут объекта, но не затрагивая базу данных.
- Для записи обновленного значения в базу данных вызывается метод game.save(). Этот метод формирует и выполняет SQL-запрос типа UPDATE, который применяет все изменения к соответствующей строке в таблице BoardGames.

Впоследствии это поле используется для сортировки товаров в каталоге по популярности (order_by('-popularity')), создавая автоматически обновляемый рейтинг игр на основе реального интереса пользователей.

2.4 Описание страниц сайта

Главная страница интернет-магазина выполняет функцию витрины и навигации. Ее основная задача — предоставить пользователю удобный инструмент для быстрого поиска, фильтрации и сортировки товаров, а также отображение каталога в виде сетки карточек. Представление, обрабатывающее этот маршрут, построено по принципу динамического формирования контента на основе параметров GET-запроса. При загрузке страницы без параметров инициируется начальная выборка всех товаров через BoardGames.objects.all(). Этот полный набор данных обрабатывается, к нему применяются фильтры по жанру при наличии соответствующего параметра, выполняется текстовый поиск по названию и описанию, а затем происходит сортировка. Для сортировки используется конструкция match/case, которая сопоставляет полученное значение параметра sort с одним из предопределенных вариантов — по возрастанию или убыванию цены, дате добавления, популярности или алфавитному порядку. По умолчанию товары упорядочиваются по новизне. Шаблон рендерит страницу, состоящую из ключевых элементов: поисковой строки, блока фильтров в виде списка жанров, выпадающего списка сортировки и адаптивной сетки товаров. Каждая карточка товара в этой сетке

содержит: изображение игры, название, цену, количество игроков и минимальный возраст. Клик по кнопке "Купить" ведет на уникальную страницу товара, что обеспечивает переход детальному изучению игры.

Страница товара представляет детальную карточку настольной игры. Ее назначение — предоставить полную информацию об игре. При обращении к URL вида /game-pages/15 активируется представление. В первую очередь, с помощью осуществляется получение объекта игры по переданному item_id. После успешного получения объекта происходит скрытое от пользователя обновление бизнес-логики, значение счетчика популярности увеличивается на единицу с последующим сохранением в базу данных. Эта операция реализует механизм аналитики, формируя рейтинг игр на основе интереса посетителей. Подготовленный объект передается в контекст шаблона, который рендерит страницу. Визуальный акцент сделан на крупном изображении игры. Рядом располагается блок с основными данными: название, цена и полное текстовое описание, раскрывающее механику, сюжет и особенности игры. Для быстрого ознакомления вынесены ключевые характеристики: количество игроков, среднее время игры и минимальный возраст. Кнопка "Добавить в корзину" обеспечивает переход к следующему шагу.

ЗАКЛЮЧЕНИЕ

В ходе выполнения учебной практики по разработке веб-приложения на основе фреймворка Django с взаимодействим с реляционной базой данных был реализован функционирующий прототип онлайн-магазина настольных игр. Выполненный проект демонстрирует полный цикл создания веб-приложения: от анализа предметной области и проектирования структуры данных до реализации бизнес-логики и интеграции с пользовательским интерфейсом

Исходным этапом работы стал анализ развивающегося рынка настольных игр. Изучение рыночных тенденций, целевой аудитории и ее специфических потребностей позволило сформулировать техническое задание. Это понимание стало фундаментом для проектирования функционала, ориентированного на решение конкретных задач пользователей. Таким образом, работа носила не только технический, но и аналитический характер.

Центральной и наиболее содержательной частью практики стало проектирование и реализация структуры данных. Процесс трансформации бизнес-сущностей в нормализованную логическую схему базы данных, а затем в конкретные модели Django ORM. Выбор SQLite на этапе разработки доказал свою эффективность, позволив сосредоточиться на освоении абстракций Django — таких как декларативное объявление полей, типов связей и системы миграций — без отвлечения на сложности администрирования серверной СУБД.

Особое значение имела реализация логики взаимодействия с данными. Разработка представлений для главной страницы и карточки товара потребовала погружения в работу QuerySet — основного инструмента Django для построения запросов. Были применены на практике такие операции, как фильтрация, связанная фильтрация по полям связанных моделей, регистронезависимый текстовый поиск, а также разнообразная сортировка результатов. Следует отметить реализацию механизма динамического

формирования запроса на основе пользовательского ввода. Отдельно был проработан сценарий работы с единичным объектом: его безопасное извлечение, изменение атрибутов и сохранение обратно в базу, что иллюстрирует цикл CRUD-операций.

Параллельно с backend-разработкой велась работа над фронтенд-составляющей проекта. Создание пользовательского интерфейса на основе системы шаблонов Django с наследованием позволило понять принципы поддержания целостности и единообразия дизайна на всех страницах сайта. Разработка дизайна, ориентированного на удобство восприятия и простоту навигации, потребовала учитывать эстетические аспекты. Взаимодействие между слоями приложения — когда представление готовит данные, а шаблон отвечает за их отображение — было успешно настроено, что завершило создание целостного, работоспособного веб-приложения.

Таким образом, в результате прохождения учебной практики были достигнуты все поставленные цели и решен комплекс взаимосвязанных задач. Были получены, и закреплены на практике навыки backend-разработки на Django: от описания моделей данных и построения миграций до реализации сложной бизнес-логики в представлениях. Созданный проект представляет собой полноценное веб-приложение, готовое к дальнейшему расширению.

Приобретенный опыт создает прочный фундамент для изучения более сложных аспектов разработки, таких как оптимизация запросов к базе данных, работа с асинхронными операциями, внедрение API. Данный проект может служить отправной точкой для реализации более масштабных идей, например, добавления системы онлайн-оплат, интеграции с сервисами доставки, разработки рекомендательной системы или мобильного клиента.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Карты в руки: как развивается рынок настольных игр в России и мире // РБК Тренды URL: https://trends.rbc.ru/trends/social/68527f0f9a794728f84d392d (дата обращения: 22.11.2025).
2. Система управления базами данных: что это такое и зачем она нужна // Skillbox URL: https://skillbox.ru/media/code/sistema-upravleniya-bazami-dannykh-chto-eto-takoe-i-zachem-ona-nuzhna/ (дата обращения: 24.11.2025).
3. SQLite — замечательная встраиваемая БД (часть 1) // Хабр URL: https://habr.com/ru/articles/149356/ (дата обращения: 25.11.2025).
4. Django documentation // Django URL: https://docs.djangoproject.com/en/6.0/ (дата обращения: 25.11.2025).
5. Django ORM для начинающих | Оптимизируем запросы // Хабр URL: https://habr.com/ru/articles/503526/ (дата обращения: 26.11.2025).
6. URL dispatcher // Django URL: https://docs.djangoproject.com/en/5.2/topics/http/urls/ (дата обращения: 28.11.2025).
7. Создание моделей // Metanit URL: https://metanit.com/python/django/5.1.php (дата обращения: 28.11.2025).
8. Маграции // devman URL: https://dvmn.org/encyclopedia/django-migrations/migrations/ (дата обращения: 29.11.2025).
9. Templates // Django URL: https://docs.djangoproject.com/en/5.2/topics/templates/ (дата обращения: 30.11.2025).
10. Большой гайд по миграциям в Django: полезные советы и обход типичных подводных камней // Хабр URL: https://habr.com/ru/companies/idaproject/articles/865036/ (дата обращения: 31.11.2025).

11. Как работают Django Class-based views // Хабр URL: https://habr.com/ru/articles/568198/ (дата обращения: 1.12.2025).

12. Представления (Views) // Хекслет URL: https://ru.hexlet.io/courses/python-django-basics/lessons/views/theory_unit (дата обращения: 1.12.2025).

13. Django settings // Django URL: https://docs.djangoproject.com/en/6.0/topics/settings/ (дата обращения: 1.12.2025).

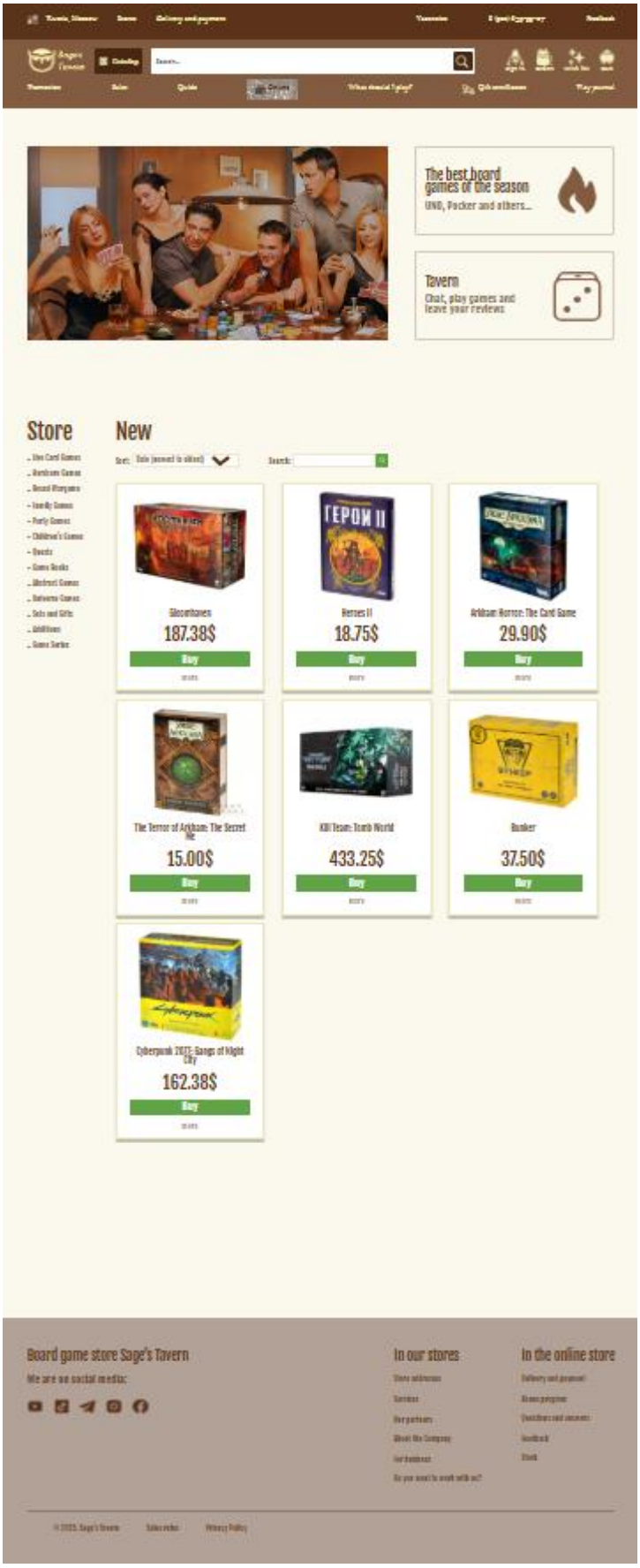
14. Шаблоны // Metanit URL: https://metanit.com/python/django/2.1.php (дата обращения: 2.12.2025).

15. Система шаблонов Django: как использовать для создания динамических сайтов // Skypro URL: https://sky.pro/wiki/python/rabota-s-shablonami-v-django/ (дата обращения: 2.12.2025).

16. HTML: HyperText Markup Language // MDN URL: https://developer.mozilla.org/en-US/docs/Web/HTML (дата обращения: 2.12.2025).

17. CSS: Cascading Style Sheets // MDN URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата обращения: 2.12.2025).

ПРИЛОЖЕНИЕ


The best board games of the season
UNO, Poker and others...
Tavern
Chat, play games and leave your reviews
Store
New
Герои II
187.38$
18.75$
29.90$
15.00$
433.25$
37.50$
162.38$
Buy
Board game store Sage's Tavern
We are on social media:
In our stores
In the online store