Министерство образования Республики Беларусь

Учреждение образования

«БелорусскиЙ государственный университет

информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

Дисциплина «Объектно-ориентированное программирование»

|  |
| --- |
| *К защите допустить*: |
| Руководитель курсовой работы  ассистент |
| \_\_\_\_\_\_\_\_\_\_\_\_А.В.Михалькевич  \_\_\_.\_\_\_. 20\_\_\_ |

**Пояснительная записка**

к курсовой работе

на тему

**Разработка онлайн-тренажера по белорусскому языку** БГУИР КР 1-40 05 01-10 014 ПЗ

|  |  |  |
| --- | --- | --- |
| Студент | (подпись студента) | А.Г. Якшук |
|  |  | Курсовая работа представлена на проверку \_\_\_.\_\_\_. 20\_\_\_ |
|  |  | (подпись студента) |

Минск 2020

СОДЕРЖАНИЕ

[Введение 4](#_Toc6294113)

[1 Описание проекта 5](#_Toc6294114)

[1.1](#_Toc6294117) Серверная часть  [5](#_Toc6294117)

[1.2](#_Toc6294118) Клиентская часть  [8](#_Toc6294118)

[2 Обоснование выбора технологий 10](#_Toc6294115)

[2.1](#_Toc6294117) Технологии программирования  [10](#_Toc6294117)

[2.2](#_Toc6294118) Клиентская часть  [13](#_Toc6294118)

[3 Инструментарий 16](#_Toc6294116)

[3.1](#_Toc6294117) [Обоснование используемых инструментов 16](#_Toc6294117)

[3.2](#_Toc6294118) [Использование системы контроля версий GIT 18](#_Toc6294118)

[4 Архитектурный шаблон проектирования MVC 19](#_Toc6294119)

[5](#_Toc6294120) Шаблон проектирования практических задач [23](#_Toc6294120)

[Заключение 25](#_Toc6294122)

[Список использованных источников 26](#_Toc6294123)

[Приложение А (обязательное) Листинг кода](#_Toc6294124) программы  [27](#_Toc6294124)

[Приложение Б (обязательное) Графический материал 43](#_Toc6294124)

**ВВЕДЕНИЕ**

Современные веб-сайты – это настоящий кладезь информации. В открытом доступе находится масса полезного и важного контента различных форматов, что позволяет пользователям постоянно расширять свой кругозор, а также, что немаловажно, взаимодействовать с различными коммерческими объектами. Любой интернет-ресурс ориентирован на какую-то целевую группу пользователей. Всех их объединяют общие интересы к той или иной информации. Для них ключевая цель использования сети – нахождение максимально точного ответа на поставленный вопрос.

Все большее количество разработчиков старается создать удобные и легкие в использовании программы, сайты, порталы для людей, которые позволяют им не только проводить свое время для развлечений, но и для получения дополнительных знаний. Проанализировав рынок онлайн-образования, я заметил, что какой-то конкретной среды для изучения белорусского языка на данный момент нет. Исходя из этого и следуя мировым трендам, нужно реализовать портал, который дает возможность быстро найти нужную обучающую и свежую информацию об белорусском языке, а также возможность протестировать свои знания.

Цель данной курсовой работы — это разработать образовательную платформу по изучению белорусского языка на фреймворке Django. Основными задачами являются создание интуитивно понятного пользовательского интерфейса, реализация модуля для изучения материала, проверка знаний и получения отчета по проверке знаний.

Для достижения поставленной цели необходимо решить следующие задачи:

* Изучение современных инструментов разработки сайтов;
* Разработка макета платформы;
* Создание баз данных для хранения информации;
* Создание сервиса для работы с тестами;

**1** **ОПИСАНИЕ ПРОЕКТА**

**1.1 Серверная часть**

Сервер написан на Python-фреймворке Django, который широко применяется для веб приложения. Благодаря большому количеству модулей в нем, его можно использовать для проектов любой сложности и задачи.

На серверной части можно выделить три главных компонента, которые соответствуют трем компонентам шаблона проектирования MVC (шаблон полностью описан в следующих разделах пояснительной записки): модель, представление, контроллер. В модели сосредоточена основанная бизнес-логика приложения, контроллер – связующее звено между моделью и представлением, который также управляет потоком данных, а представление – итоговая html-страница, которую видит пользователь.

Контроллер — это код, запускаемый в ответ на поступление клиентского запроса, который содержит интернет-адрес определенного формата. Именно в контроллерах выполняются все действия по подготовке данных для вывода, равно как и обработка данных, поступивших от посетителя.

Контроллер Django может представлять собой как функцию (контроллер-функция), как и класс (контроллер-класс), которые представлены на рисунке 1.1. Первые более универсальны, но зачастую трудоемки в программировании, вторые позволяют выполнить типовые задачи, наподобие вывода списка каких-либо позиций, минимумом кода.

Основная часть функциональности различных контроллеров-классов реализована в примесях (классах, предназначенных лишь для расширения функциональности других классов), от которых наследуют контроллеры-классы.

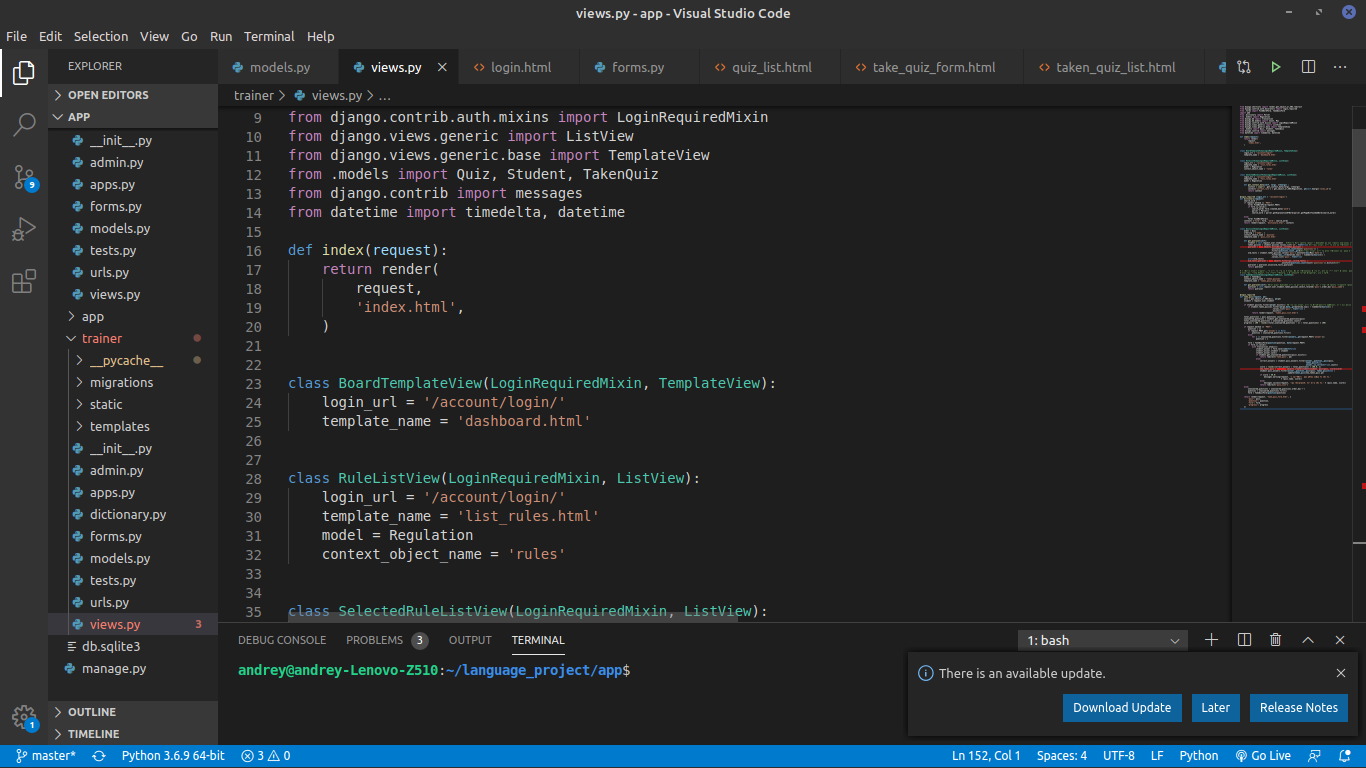


Рисунок 1.1— Контроллер-функция и контроллер-класс

ListView самостоятельно извлекает из модели набор записей, записывает его в атрибут object\_list (чтобы успешно работали наследуемые им примеси) и выводит на экран страницу со списком записей. Контроллер RuleListView унаследован от LoginRequiredMixin и ListView. Первый контроллер-класс не позволяет неавторизированными пользователям посетить данную страницу, перенаправляя их на форму авторизации. Плюс ко всему сказанному, данный класс контролирует обратное перенаправление на ту странуцу, куда пользователь хотел попасть изначально. Второй класс позволяет выводить шаблон со всеми записями модели Regulation, сохраненных в контексте шаблона.

Пользователи могут быть неавторизированными и авторизированными. Неавторизированными будут новые пользователи или те, кто ранее вышли из системы, а авторизированными являются пользователи, вошедшие в приложение раннее. Даже при перезапуске сервера, пользователи, которые были в системе, в ней и останутся, так как вся информация о сессиях сохраняется не в приложения, а в отдельной таблице БД.

Авторизация пользователя проходи в контроллере-функции user\_login (рисунок 1.2). Прежде всего проверяется, какой запрос был отправлен. В случаи POST запроса, проверяется правильность заполнения формы, после чего проверяется, есть ли данный пользователь и его пароль. При успешной авторизации пользователь перенаправляется на главную страницу.

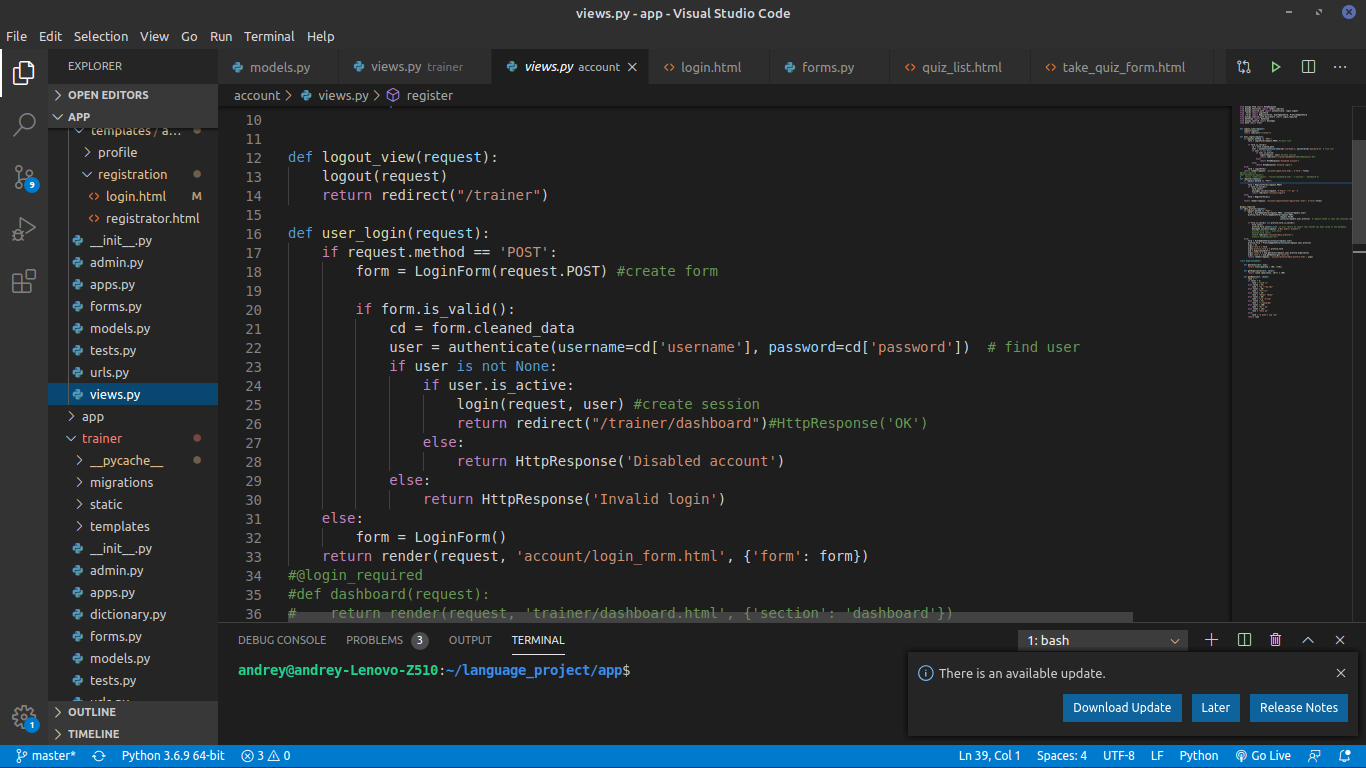


Рисунок 1.2— Авторизация пользователя

Для формирования списка доступных пользователю используется контроллер QuizListView (рисунок 1.3). Здесь переопределен метод get\_queryset, который отвечает за формирование контекста для шаблона. Данные естественно берутся из базы данных. Сначала мы получаем список тестов, которые пользователь еще ни разу не проходил. Далее находим список тестов, которые пользователь отвечал, но может повторить. То есть при прохождении теста, он будет доступен для прохождения только через 2 часа. Все результаты тестов сохраняются в модель TakenQuiz, которой соответствует таблица в базе данных.

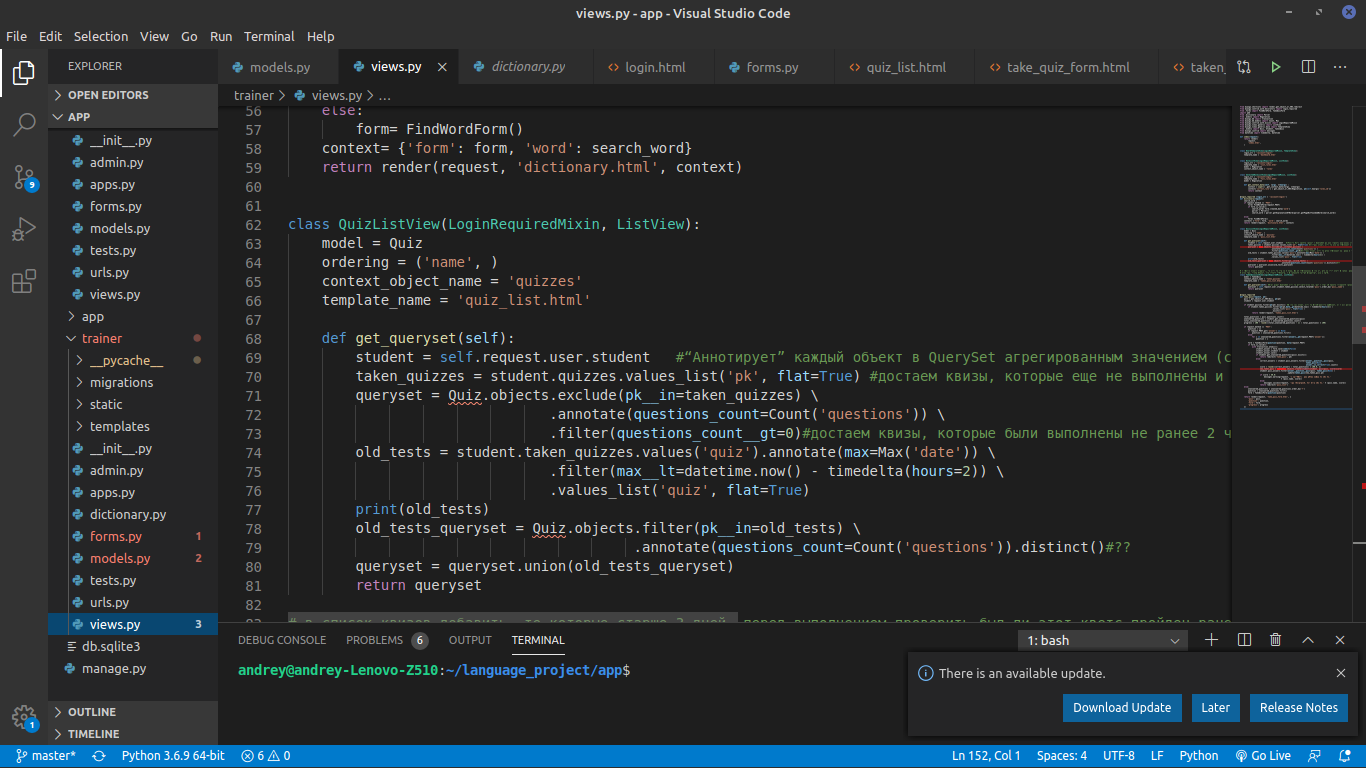


Рисунок 1.3— Контроллер QuizListView

Для пользователя также реализована страница для редактирования его профиля. За формирование и изменение профиля отвечает контроллер edit\_profile. Он представлен на рисунке 1.4. Сначала формируется форма, в которую заносится уже известная информация об пользователе: его аватар(каждому пользователю существует картинка по умолчанию), дата рождения, если заполнена, его опыт, который получает он за прохождение тестов. Далее в этой форме, при желании, вносится изменения, после чего форма проходит валидацию. При успешном изменении профиля система выводит сообщение об этом. Данные профиля сохраняются в соответствующей модели Profile. Данная модель связана с моделью пользователя, которая предоставлена Django по умолчанию, один к одному.

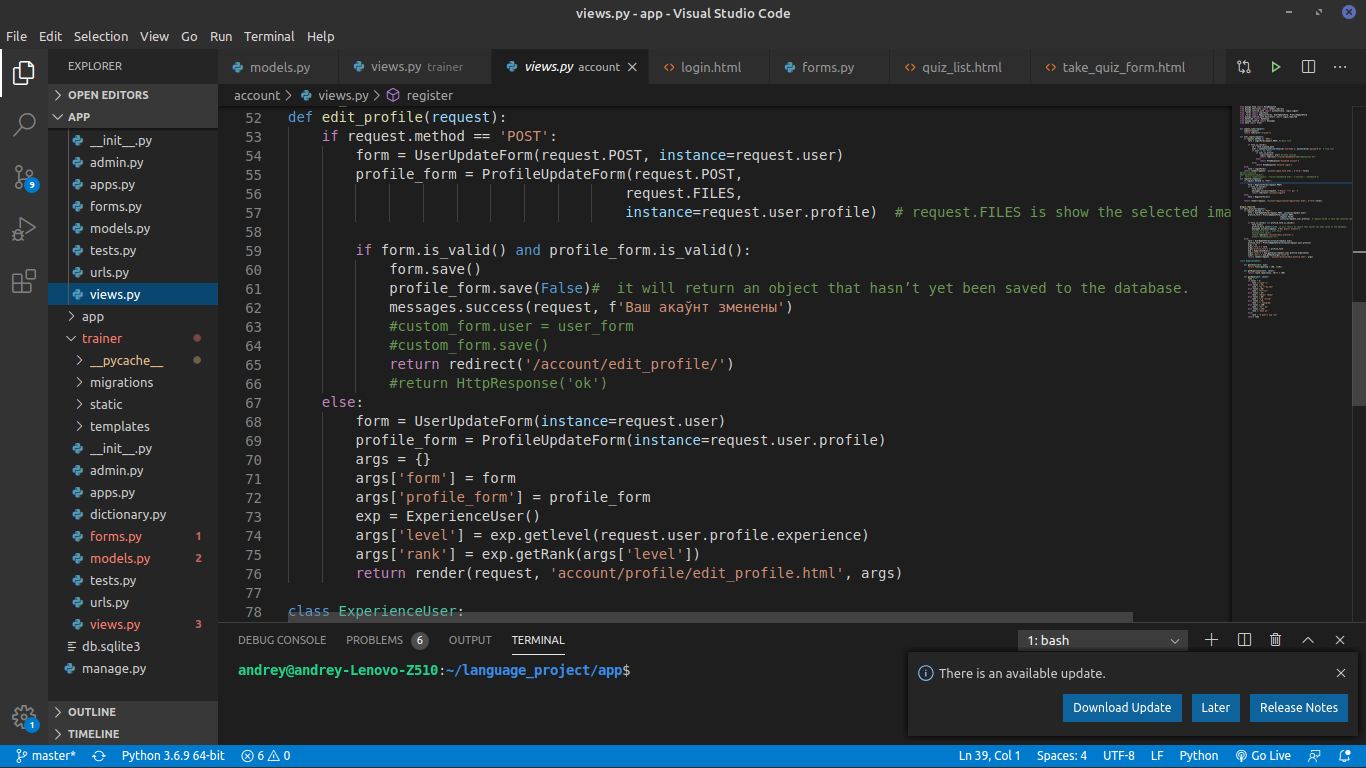


Рисунок 1.4— Контроллер edit\_profile

**1.2 Клиентская часть**

Будучи веб фреймверком, Django позволяет динамически генерировать HTML. Самый распространенный подход - использование шаблонов. Шаблоны содержат статический HTML и динамические данные, рендеринг которых описан специальным синтаксисом.

Проект Django может использовать один или несколько механизмов создания шаблонов (или ни одного, если вы не используете шаблоны). Django предоставляет бэкенд для собственной системы шаблонов, которая называется - язык шаблонов Django (Django template language, DTL), и популярного альтернативного шаблонизатора Jinja2. Сторонние приложения могут предоставлять бэкенд и для других систем шаблонов.

Шаблон — это образец для формирования веб-страницы, которая будет отправлена посетителю в ответ на его запрос. Также шаблон может применяться для формирования текстовых документов — например сообщений электронной почты.

Шаблонизатор - подсистема фреймворка, непосредственно формирующая окончательные веб-страницы (и текстовые документы), объединяя шаблоны и предоставленные контроллерами контексты шаблонов, содержащие все необходимые данные.

Преимуществом является то, что есть возможность создать базовый шаблон, а после чего его наследовать в другие, тем самым сокращая объем кода. Все шаблоны хранятся в папке templetes приложения.

Теги шаблонизатора управляют генерированием содержимого результирующего документа. Они записываются в формате:{ %< собственно запись тега со всеми необходимыми параметрами> % }

Как и НТМL-теги, теги шаблонизатора бывают одинарными и парными. Одинарный тег просто помещается в нужное место кода шаблона. Как правило, он вставляет в это место какое-либо значение, вычисляемое самим фреймворком. Пример одинарного тега, вставляющего в код шаблона электронный жетон, который используется подсистемой безопасности фреймворка:{% csrf token %}

Парный тег «охватывает» целый фрагмент кода шаблона. Он применяется для выполнения над этим фрагментом каких-либо действий. Например, парный тег for .. in повторяет содержащийся в нем фрагмент столько раз, сколько элементов находится в указанной в этом теге последовательности. Пример реализации на рисунке 1.5.

Как видим, парный тег фактически состоит из двух тегов: открывающего и закрывающего (в приведенном только что примере это теги for .. in и endfor). Закрывающий тег имеет то же имя, что и открывающий, но с добавленным впереди префиксом end. Фрагмент кода, находящийся внуrри парного тега, носит название содержимого.

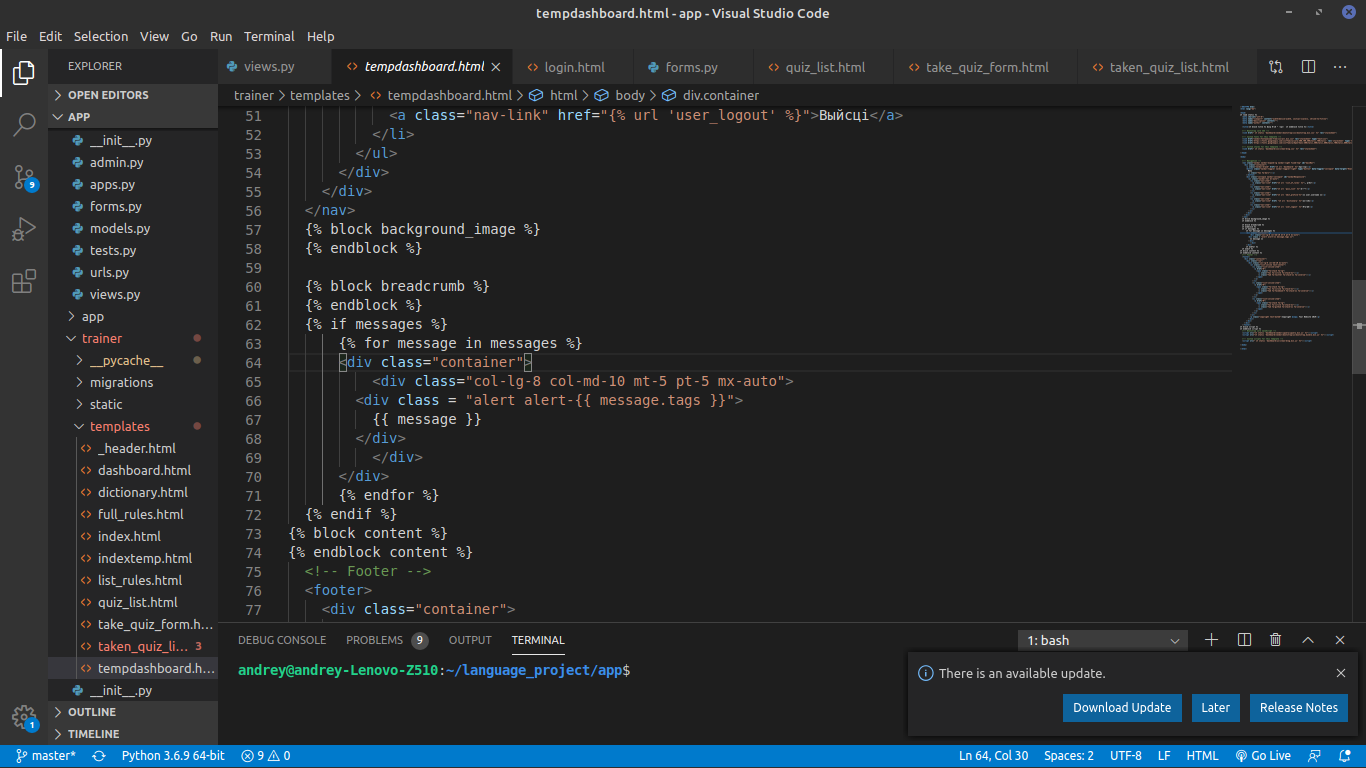


Рисунок 1.5— Пример парного тега for

Особенностью является также возможность использовать фильтры шаблонизатора. Они выполняют определенные преобразования значения перед его выводом. Фильтр записывается непосредственно в директиве, после имени переменной, из которой извлекается текущее значение для обработки, и отделяется от нее симво­лом вертикальной черты (рисунок 1.6).

В данном примере используется фильтр first, который дает возможность получить первый элемент из переменной word. Также используется функция length, которая возвращает значение длины списка.

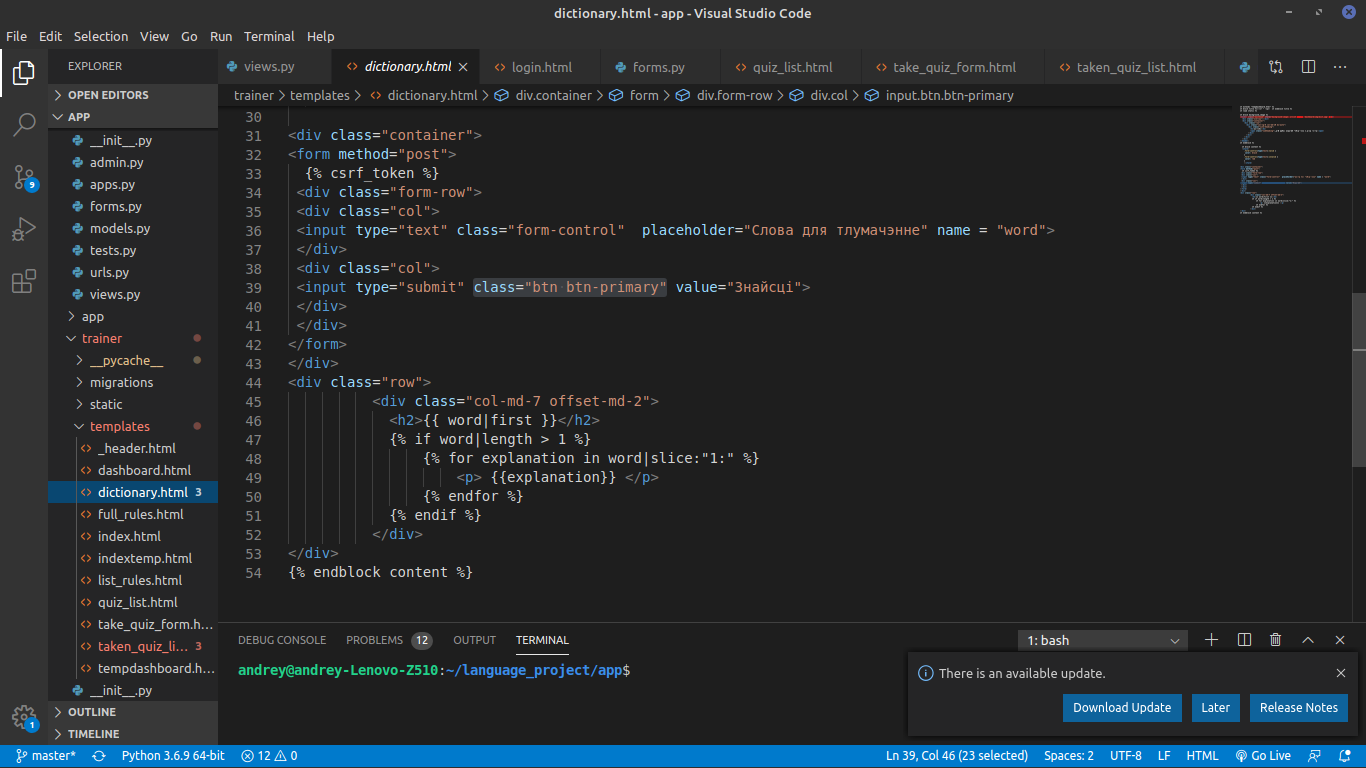


Рисунок 1.6— Пример фильтра шаблонизатора

Аналогично наследованию классов, практикуемому в Python, Django предлагает нам механизм наследования шаблонов. В Python базовый класс определяет функциональность, которая должна быть общей для всех наследующих его производных классов. Точно так же и вDjango: базовый шаблон определяет все элементы, которые должнь1 быть общими для всех наследующих его производных шаблонов. Помимо этого, базовый шаблон определяет набор блоков. Блок представляет собой место в коде базового шаблона, куда производный шаблон поместит свое уникаль­ное содержимое. Таких блоков может быть сколько угодно.

1. **ОБОСНОВАНИЕ ВЫБОРА ТЕХНОЛОГИЙ**

**2.1 Технологии программирования**

Исходя из обычного значения слова «технология» под термином «технологией программирования» понимают совокупность производственных процессов, которая обеспечивает создание требуемого программного средства, а также документирование этой последовательности процессов. Таким образом, под технологией программирования будем понимать процесс разработки программных средств, включая все стадии, начиная с момента появления идеи этого средства; причем каждый из этапов создания программного продукта опирается на использовании определенных методов.

В историческом аспекте в развитии технологии программирования можно выделить несколько этапов, таких как: стихийное программирование, структурный подход к программированию, объектный подход к программированию, компонентный подход и CASE-технологии. В рамках данной курсовой будет рассматриваться в основном объектный подход.

Объектно-ориентированная разработка программного обеспечения существует с начала 1960-х годов. Только во второй половине 1990-х годов объектно-ориентированная парадигма начала набирать обороты, несмотря на тот факт, что популярные объектно-ориентированные языки программирования вроде Smalltalk и C++ уже широко использовались. Расцвет объектно-ориентированных технологий совпал с началом применения Интернета в качестве платформы для бизнеса и развлечений. А после того как стало очевидным, что Глобальная сеть активно проникает в жизнь людей, объектно-ориентированные технологии уже заняли удобную позицию для того, чтобы помочь в разработке новых веб-технологий.

Вы сталкиваетесь с объектами в своей повседневной жизни, вероятно, даже не осознавая этого. Вы можете столкнуться с ними, когда едете автомобиле, разговариваете по телефону, играете в компьютерные игры, а также во многих других ситуациях. Электронные соединения по сути превратились в соединения, основанные на объектах. Тяготея к мобильным веб-приложениям, бизнес тяготеет к объектам, поскольку технологии, используемые для электронной торговли, по своей природе в основном являются объектно-ориентированными.

В объектно-ориентированной разработке во главу угла ставится объект, как основополагающая сущность данной технологии. В таком случае нужно задать вопрос, а что же такое объект? Этот термин интуитивно понятен, потому что люди уже мыслят объектно. Например, когда вы смотрите на какого-то человека, вы видите его как объект. При этом объект определяется двумя компонентами: атрибутами и поведением. У человека имеются такие атрибуты, как цвет глаз, возраст, вес и т.д. Человек также обладает поведением, то есть он ходит, говорит, дышит и т.д. В соответствии со своим базовым определением объект — это сущность, одновременно содержащая данные и поведения. Объекты — это строительные блоки объектно-ориентированных программ. Та или иная программа, которая задействует объектно-ориентированную технологию, по сути является набором объектов. В качестве наглядного примера рассмотрим корпоративную систему, содержащую объекты, которые представляют собой работников соответствующей компании. Каждый из этих объектов состоит из данных и поведений, описанных в последующих разделах.

При создании экземпляра объекта используется класс как основа для того, как этот объект будет создаваться. Каждый объект содержит собственные атрибуты (данные) и поведения (функции или программы). Класс определяет атрибуты и поведения, которые будут принадлежать всем объектам, созданным с использованием этого класса. Классы — это фрагменты кода. Объекты, экземпляры которых созданы на основе классов, можно распространять по отдельности либо как часть библиотеки. Объекты создаются на основе классов, поэтому классы должны определять базовые строительные блоки объектов (атрибуты, поведения и сообщения).

Язык Python – типичный представитель ООП-семейства, обладающий элегантной и мощной объектной моделью. В этом языке от объектов никуда не спрятаться (ведь даже числа являются ими).

Выбор использования данного языка основан на том, что скорость выполнения программ, написанных на Python высока. Это связанно с тем, что основные библиотеки Python написаны на C++ и выполнение задач занимает меньше времени, чем на других языках высокого уровня. Поэтому есть возможность писать свои собственные модули для Python на C или C++. Следующим преимуществом языка является его кроссплатформенность, т.е. скрипты, написанные при помощи Python выполняются на большинстве современных ОС. Такая переносимость обеспечивает Python применение в самых различных областях. Стоит также отметить широкое распространение пакетов (библиотек), используемых в научных вычислениях. Они способны вычислять классические численные алгоритмы решения уравнений, задач линейной алгебры, вычисления определенных интегралов, аппроксимации, решения дифференциальных уравнений. Python подходит для любых решений в области программирования, будь то офисные программы, веб-приложения, GUI-приложения и т.д.

В рамках объектно-ориентированных технологий для создания веб-сервиса для обучения белорусскому языку был выбрана технология, реализованная на данном языке – веб-фрейворк Django. Этот инструмент удобно использовать для разработки сайтов, работающих с базами данных.

Фреймворк предлагает большую ценность для любого проекта веб-разработки. Данная технология поставляется с набором готовых компонентов, которые помогают ускорить и упростить процесс создания веб-приложений. Фреймворк содержит удобные компоненты, реализующие наиболее распространенные функции, которые мы находим в веб-приложениях. Например, аутентификацию пользователей, панель управления, загрузку файлов и многое другое. Тем не менее, используя эту технологию, веб-разработчикам не нужно каждый раз заново изобретать колесо. Не нужно создавать эти компоненты с нуля. Фреймворк помогает им сэкономить время, ускорить процесс разработки программного обеспечения и сосредоточиться на аспектах. Все это делает веб-приложение уникальным на рынке.

Значительную роль в успехе Django сыграли несколько известных сайтов, которые использовали этот фреймворк. В их число входят Pinterest, Dropbox, Spotify, сайт The Washington Post. В настоящее время сообщество Django включает более 11 тыс. разработчиков из 166 стран мира. В фрейворке реализован принцип DRY (don’t repeat yourself). Благодаря этому сокращается время создания сайтов. То есть при использовании Django вам не нужно несколько раз переписывать один и тот же код. Фреймворк позволяет создавать сайт из компонентов. Это можно сравнить со строительством крепости с помощью Lego.

Создатели реализовали в Django паттерн MVC, и он применяется в текущей версии фреймворка. Архитектура MVC позволяет разработчику работать с визуальным представлением и бизнес-логикой приложения отдельно. Кстати, при работе с Django специалисты чаще используют термин MVT — Model-View-Template или модель-представление-шаблон. Компоненты MVT можно использовать независимо друг от друга.

Документация Django определяет модель (model) как «источник информации о данных, в которых содержатся ключевые поля и поведение данных». Обычно одна модель указывает на одну таблицу в базе данных. Django поддерживает базы данных PostgreSQL, MySQL, SQLite и Oracle. Модели содержат информацию о данных. Эти данные представлены атрибутами или полями. Поскольку модель представляет собой простой класс, она ничего не знает о других уровнях Django. Взаимодействие между уровнями происходит через API. Модель отвечает за бизнес-логику, методы, свойства и другие элементы, связанные с манипуляцией данными. Также модели позволяют разработчикам создавать, читать, обновлять и удалять объекты в базе данных. Представление (view) решает три задачи: принимает HTTP-запросы, реализует бизнес-логику, определённую методами и свойствами, отправляет HTTP-ответ в ответ на запросы. То есть представление получает данные от модели и предоставляет шаблонам (templates) доступ к этим данным или предварительно обрабатывает данные и затем предоставляет к ним доступ шаблонам. В Django реализован мощный движок шаблонов и собственный язык разметки. Шаблоны представляют собой файлы с HTML-кодом, с помощью которого отображаются данные. Содержимое файлов может быть статическим или динамическим. Шаблоны не содержат бизнес-логики. Поэтому они только отображают данные.

Первоначально разработка Django велась для обеспечения более удобной работы с новостными ресурсами, что достаточно сильно отразилось на архитектуре: фреймворк предоставляет ряд средств, которые помогают в быстрой разработке веб-сайтов информационного характера. Например, разработчику не требуется создавать контроллеры и страницы для административной части сайта, что является, по моему мнению, килер-фичей. В Django есть встроенное приложение для управления содержимым, которое можно включить в любой сайт, сделанный на Django, и которое может управлять сразу несколькими сайтами на одном сервере. Административное приложение позволяет создавать, изменять и удалять любые объекты наполнения сайта, протоколируя все совершённые действия, и предоставляет интерфейс для управления пользователями и группами (с пообъектным назначением прав).

**Некоторые возможности Django:**

* ORM, API доступа к БД с поддержкой транзакций;
* встроенный интерфейс администратора, с уже имеющимися переводами на многие языки;
* диспетчер URL на основе регулярных выражений;
* расширяемая система шаблонов с тегами и наследованием;
* система кеширования;
* интернационализация;
* подключаемая архитектура приложений, которые можно устанавливать на любые Django-сайты;
* «generic views» — шаблоны функций контроллеров;
* авторизация и аутентификация, подключение внешних модулей аутентификации: LDAP, OpenID и прочие;
* система фильтров («middleware») для построения дополнительных обработчиков запросов, как например включённые в дистрибутив фильтры для кеширования, сжатия, нормализации URL и поддержки анонимных сессий;
* библиотека для работы с формами (наследование, построение форм по существующей модели БД);
* встроенная автоматическая документация по тегам шаблонов и моделям данных, доступная через административное приложение;

**2.2 Реализация ООП технологий**

Для реализация объектно-ориентированных технологий программирования, должны выполнятся 4 основных принципа: абстракция данных, инкапсуляция, полиморфизм, наследование.

Наследование — механизм, который позволяет описать новый класс на основе существующего (родительского). При этом свойства и функциональность родительского класса заимствуются новым классом. Реализация в проекте на рисунке 2.1.

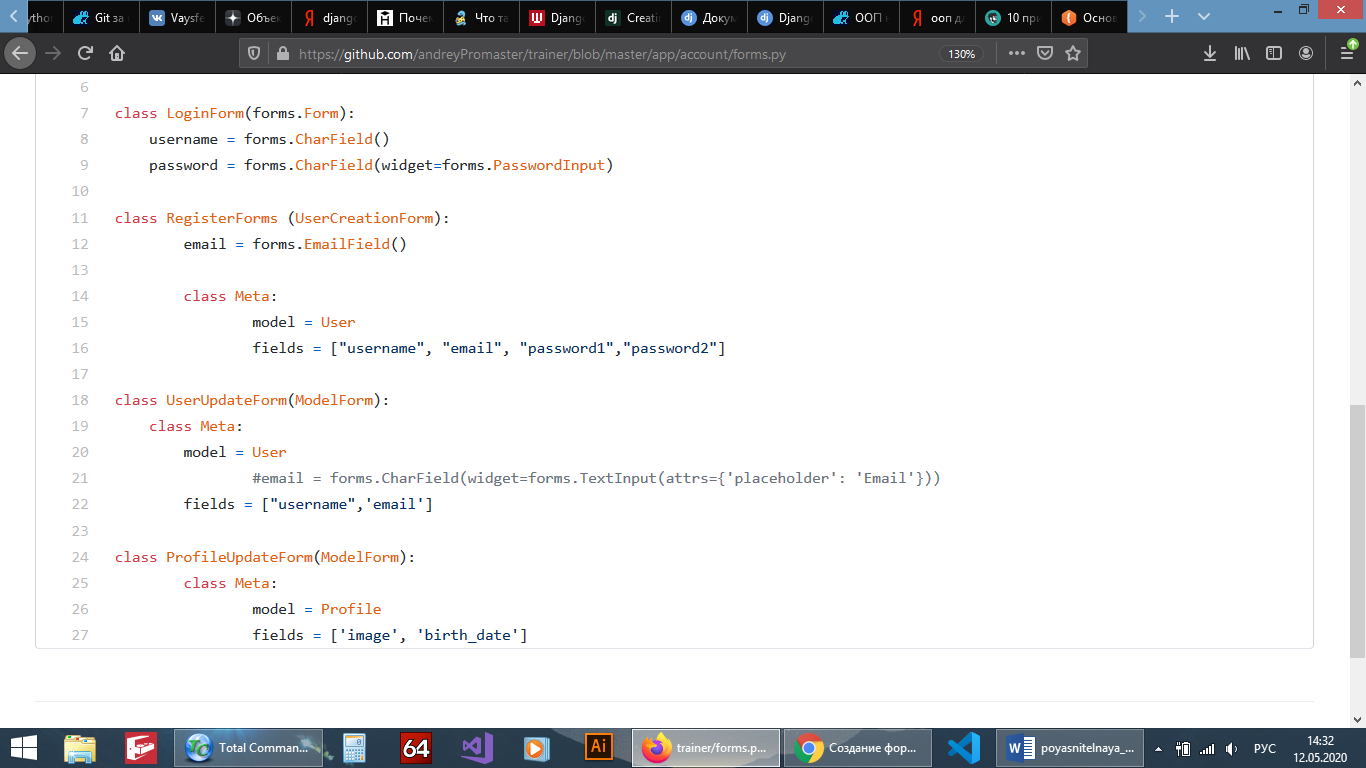


Рисунок 2.1— Пример наследования в курсовой работе

При создании приложения на основе базы данных, будут формы, которые тесно связаны с моделями Django. Например, у нас User модель, и мы хотим создать форму, которая позволяет изменять профиль пользователя. В этом случае было бы излишним определять типы полей в вашей форме, потому что мы уже определили поля в своей модели. По этой причине Django предоставляет вспомогательный класс, который позволяет создавать Form класс из модели Django.

Абстракция очень простой принцип. Благодаря абстракции данных, объекты представляют собою упрощенное, идеализированное описание реальных сущностей предметной области. Если соответствующие модели адекватны решаемой задаче, то работать с ними оказывается намного удобнее, чем с низкоуровневым описанием всех возможных свойств и реакций объекта.

Одно из основных преимуществ использования объектов заключается в том, что объекту не нужно показывать все свои атрибуты и поведения. При хорошем объектно-ориентированном проектировании (по крайней мере, при таком, которое повсеместно считается хорошим) объект должен показывать только интерфейсы, необходимые другим объектам для взаимодействия с ним. Детали, не относящиеся к использованию объекта, должны быть скрыты от всех других объектов. Инкапсуляция определяется тем, что объекты содержат как атрибуты, так и поведения. Скрытие данных является основной частью инкапсуляции. Пример можно увидеть на рисунке 2.2.

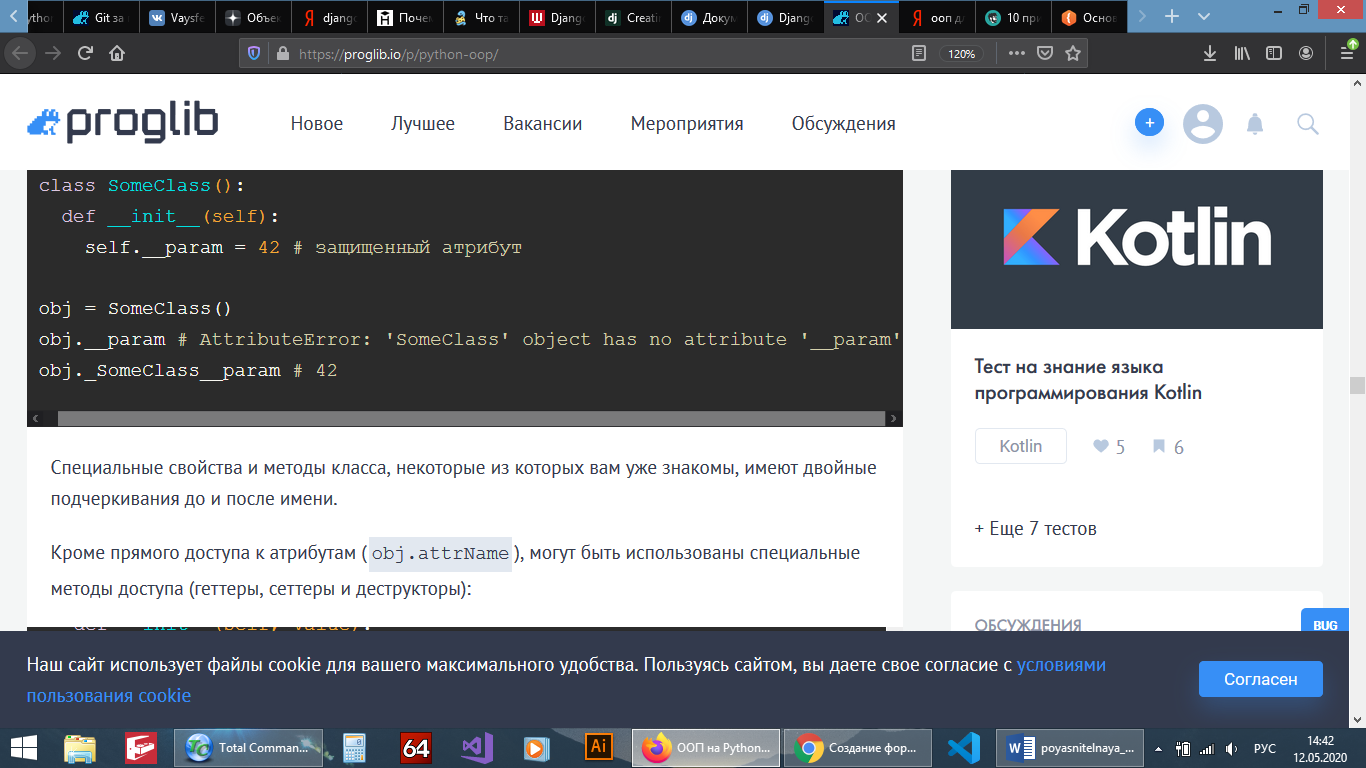


Рисунок 2.2 — Пример инкапсуляции

Полиморфизмом называют явление, при котором методу с одним и тем же именем соответствует разный программный код в зависимости от того, объект какого класса используется при вызове данного метода. Полиморфизм обеспечивается тем, что в классе-потомке изменяют реализацию метода класса-предка с обязательным сохранением сигнатуры метода. Это обеспечивает сохранение неизменным интерфейса класса-предка и позволяет осуществить связывание имени метода в коде с разными классами. Реализация на рисунке 2.3. Такой механизм называется динамическим (или поздним) связыванием — в отличие от статического (раннего) связывания, осуществляемого на этапе компиляции.

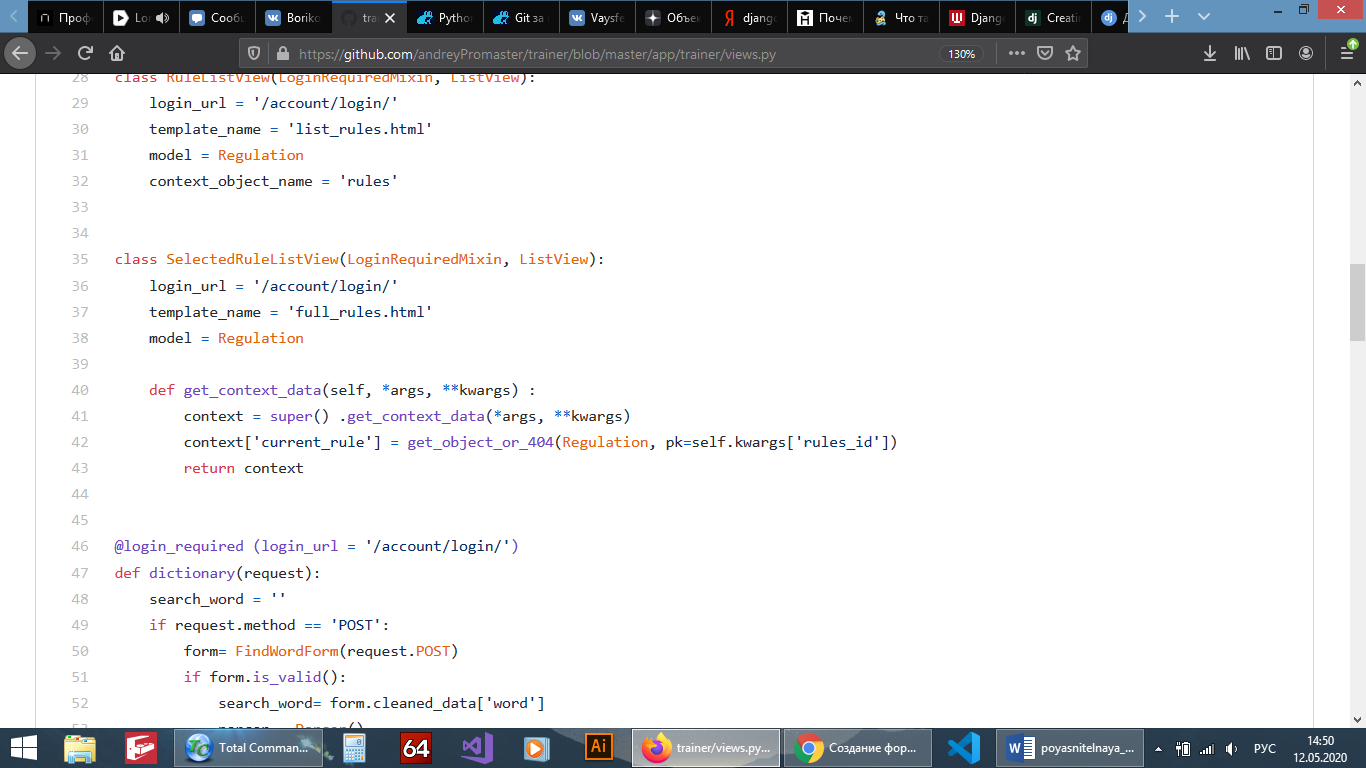


Рисунок 2.3 — Пример полиморфизма в курсовой работе

Также стоит отметить характерную черту для данного фрейворка – модульность. Модульность – это такая организация объектов, когда они заключают в себе полное определение их характеристик, никакие определения методов и свойств не должны располагаться вне его, это делает возможным свободное копирование и внедрение одного объекта в другие (рисунок 2.4). Весь проект может быть разбит на несколько приложений, которые в свою очередь, можно без труда переносить в другие проекты.

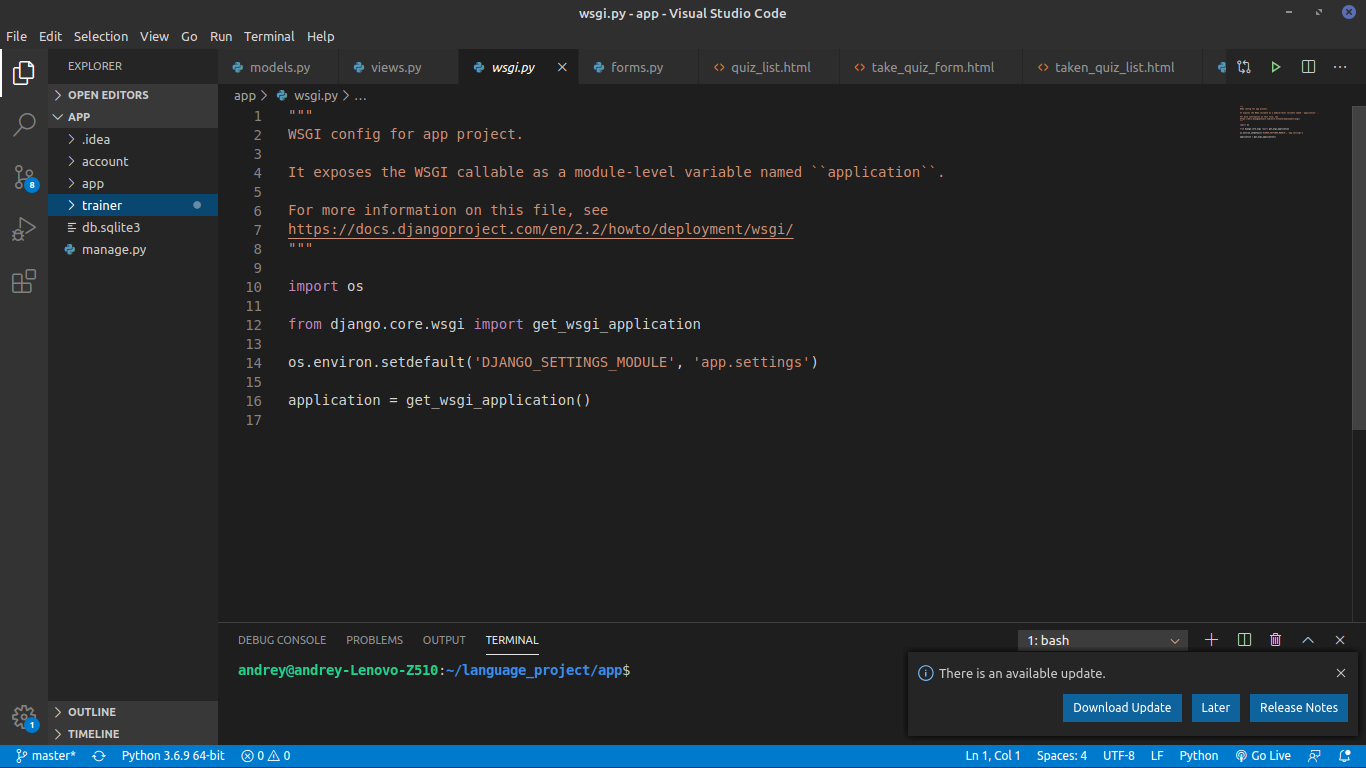
****

Рисунок 2.4 — Пример разбития на модули

# 3 ИНСТРУМЕНТАРИЙ

* 1. **Обоснование используемых инструментов**

В работе любого современного разработчика существуют инструменты, ставшие практически обязательными для каждодневного применения. Такими инструментами, в первую очередь, будут, конечно, интегрированная среда разработки (IDE) и система управления базами данных (СУБД).

Для создания этого проекта были использованы редактор кода VS Code от Microsoft и СУБД MySQL.

Visual Studio Code — редактор исходного кода, разработанный Microsoft для Windows, Linux и macOS. Позиционируется как «лёгкий» редактор кода для кроссплатформенной разработки веб- и облачных приложений. Включает в себя отладчик, инструменты для работы с Git, подсветку синтаксиса, IntelliSense и средства для рефакторинга. Имеет широкие возможности для настройки: пользовательские темы, сочетания клавиш и файлы конфигурации. Распространяется бесплатно, разрабатывается как программное обеспечение с открытым исходным кодом.

Плюсом ко всему будет являтся удобная система добавления расширений. Достаточно перейти во вкладку с расширениями и написать в поиске необходимое дополнение.

Основные возможности редактора для работы с окружением Python:

* Поддержка Python 3.4 и выше, а также Python 2.7;
* Автоматическое дополнение кода с помощью IntelliSense;
* Линтинг;
* Отладка;
* Сниппеты;
* Модульное тестирование;
* Автоматическое использование conda и виртуальных сред;
* Редактирование кода в средах Jupyter и Jupyter Notebooks;

База данных — это специально созданное хранилище важной информации, неотъемлемым атрибутом которого является удобный доступ ко всем хранящимся данным. Для обеспечения этого доступа используются так называемые системы управления базами данных или просто СУБД, представляющие собой особые приложения на базе определенной программной архитектуры.

Для реализации данного проекта, как ранее говорилось, была выбрана MySQL. Система управления базами данных (СУБД) MySQL - разработка шведской компании MySQL AB. СУБД MySQL является программным обеспечением с открытым исходным кодом, распространяемым по лицензии GNU (GPL) и коммерческой лицензии для ситуаций, не подпадающих под действие лицензии GPL. MySQL поддерживает реляционную модель данных, т. е. представляет собой реляционную СУБД.

Итак, я выбрал данную СУБД, потому что MySQL обладает целым комплексом важных преимуществ перед другими системами. В частности, следует отметить такие качества как:

1. Простота в использовании. MySQL достаточно легко инсталлируется, а наличие множества плагинов и вспомогательных приложений упрощает работу с базами данных.
2. Обширный функционал. Система MySQL обладает практически всем необходимым инструментарием, который может понадобиться в реализации практически любого проекта.
3. Безопасность. Система изначально создана таким образом, что множество встроенных функций безопасности в ней работают по умолчанию.
4. Масштабируемость. Являясь весьма универсальной СУБД, MySQL в равной степени легко может быть использована для работы и с малыми, и с большими объемами данных.
5. Скорость. Высокая производительность системы обеспечивается за счет упрощения некоторых используемых в ней стандартов.

Сам факт того, что MySQL является самой популярной СУБД на рынке, говорит об ее универсальности и предпочтительности в большинстве ситуаций. Чаще всего ее использую для работы с интернет-страницами и веб-приложениями, поскольку MySQL является наиболее удобной СУБД для этой сферы применения [3].

Вся разработка велась под операционной системой Linux Mint. Выбор данной ОС был обусловлен несколькими факторами. Для каждого пользователя важным фактором при выборе операционной системы является стабильность ее работы. Установив актуальную дистрибутив Linux, вы сразу «из коробки» получаете стабильную платформу. Все выпускаемые версии достаточно долго тестируются не только создателями, но и сообществом. Найденные ошибки и сбои исправляются практически сразу, а обновления становятся доступными для обычных юзеров только тогда, когда они удовлетворяют всем параметрам стабильности. Часто патчи и нововведения инсталлируются автоматически при активном подключении к интернету, вы можете даже не узнать о том, что кем-то найденные проблемы были оперативно исправлены. В этом заключается политика разработчиков практически всех актуальных открытых сборок, поэтому рассматриваемая ОС — одна из самых стабильных [4].

Кроме того, Linux – бесплатная операционная система, которая обладает минимальными требованиями. Все необходимые обновления не потребуют много времени, а программ под нее существует очень много, и их с легкостью может найти даже обычный пользователь. Также система Linux лучше других подходит под сетевые версии установки, поскольку в ней легче произвести настройку серверов электронной почты и WEB серверов. В линукс не существует системного реестра, который как губка накапливает в себе мусор и со временем компьютер начинает тормозить. В линукс даже через годы пользования этого не произойдет, он так и будет продолжать радовать пользователя стабильной скоростью работы.

* 1. **Использование системы контроля версий GIT**

Система управления версиями (Version Control System) — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов.

Git — распределённая система управления версиями. Система спроектирована как набор программ, специально разработанных с учётом их использования в сценариях. Это позволяет удобно создавать специализированные системы контроля версий на базе Git или пользовательские интерфейсы. Удалённый доступ к репозиториям Git обеспечивается git-демоном, SSH- или HTTP-сервером.

GitHub — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки. Веб-сервис основан на системе контроля версий Git. Сервис бесплатен для проектов с открытым исходным кодом и небольших частных проектов, предоставляя им все возможности (включая SSL), а для крупных корпоративных проектов предлагаются различные платные тарифные планы.

Создатели сайта называют GitHub «социальной сетью для разработчиков». Кроме размещения кода, участники могут общаться, комментировать правки друг друга, а также следить за новостями знакомых.

С помощью широких возможностей Git программисты могут объединять свои репозитории — GitHub предлагает удобный интерфейс для этого и может отображать вклад каждого участника в виде дерева. Для проектов есть личные страницы, небольшие Вики и система отслеживания ошибок. Прямо на сайте можно просмотреть файлы проектов с подсветкой синтаксиса для большинства языков программирования. Можно создавать приватные репозитории, которые будут видны только вам и выбранным вами людям.

Ссылка на репозиторий с проектом: https://github.com/andreyPromaster/trainer.git

**4 АРХИТЕКТУРНЫЙ ШАБЛОН ПРОЕКТИРОВАНИЯ MVC**

Шаблон проектирования Модель – Представление – Контроллер  
(MVC) – это шаблон программной архитектуры, построенный на основе сохранения представления данных отдельно от методов, которые взаимодействуют с данными.

Несмотря на то, что схема MVC была первоначально разработана для персональных компьютеров, она была адаптирована и широко используется веб-разработчиками из-за точного разграничения задач и возможности повторного использования кода. Схема стимулирует развитие модульных систем, что позволяет разработчикам быстро обновлять, добавлять или удалять функционал.

Название шаблона проектирования определяется тремя его основными составляющими частями: Модель, Представление и Контроллер. Визуальное представление шаблона MVCвыглядит, как показано на [приведенной ниже диаграмме](http://en.wikipedia.org/wiki/File:MVC-Process.png) (рис.4):

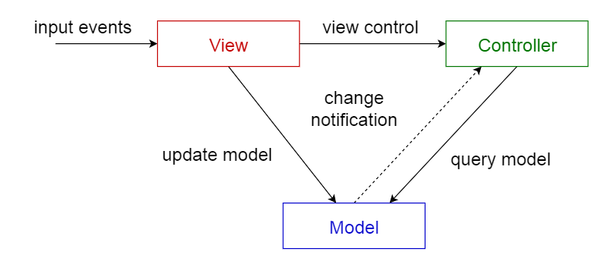


Рисунок 4.1 - Схема шаблона MVC

На рисунке показана структура одностороннего потока данных и пути его следования между различными компонентами, а также их взаимодействие.

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 задачи и выбирает подходящий вид.

Модель - это данные и правила, которые используются для работы с данными, которые представляют концепцию управления приложением. В любом приложении вся структура моделируется как данные, которые обрабатываются определённым образом. Что такое пользователь для приложения — сообщение или книга? Только данные, которые должны быть обработаны в соответствии с правилами (дата не может указывать в будущее, e-mail должен быть в определённом формате, имя не может быть длиннее Х символов, и так далее).

Модель даёт контроллеру представление данных, которые запросил пользователь (сообщение, страницу книги, фотоальбом, и тому подобное). Модель данных будет одинаковой, вне зависимости от того, как мы хотим представлять их пользователю. Поэтому мы выбираем любой доступный вид для отображения данных.

Модель содержит наиболее важную часть логики нашего приложения, логики, которая решает задачу, с которой мы имеем дело (форум, магазин, банк, и тому подобное). Контроллер содержит в основном организационную логику для самого приложения (очень похоже на ведение домашнего хозяйства). [5]

Представление — это наглядное отображение содержащихся в модели данных. Подмножество модели содержится в отдельном представлении, таким образом, представление действует в качества фильтра для данных модели. Пользователь взаимодействует с данными модели с помощью предлагаемого представлением наглядного отображения.

Представление также перехватывает действие пользователя, которое затем передается Контроллеру. Характерным примером этого является кнопка, генерируемая Представлением. Когда пользователь нажимает ее, запускается действие в Контроллере.

Но современные MVC-ориентированные фреймворки до такой степени впитали этот подход, что никто уже не заботится о том, поддерживается ли верная структура MVC или нет.

Фреймворк Django реализует архитектурный паттерн Model-View-Template или сокращенно MVT, который по факту является модификацией распростаненного в веб-программировании паттерна MVC (Model-View-Controller). Схематично мы можем представить архитектуру MVT в Django следующим образом на рисунке 4.2:

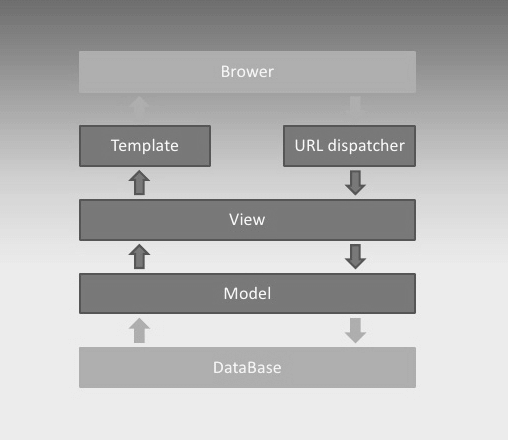


Рисунок 4.2 - Схема шаблона MVT

Основные элементы паттерна:

1. URL dispatcher: при получении запроса на основании запрошенного адреса URL определяет, какой ресурс должен обрабатывать данный запрос.
2. View: получает запрос, обрабатывает его и отправляет в ответ пользователю некоторый ответ. Если для обработки запроса необходимо обращение к модели и базе данных, то View взаимодействует с ними. Для создания ответа может применять Template или шаблоны. В архитектуре MVC этому компоненту соответствуют контроллеры (но не представления).
3. Model: описывает данные, используемые в приложении. Отдельные классы, как правило, соответствуют таблицам в базе данных.
4. Template: представляет логику представления в виде сгенерированной разметки html. В MVC этому компоненту соответствует View, то есть представления.

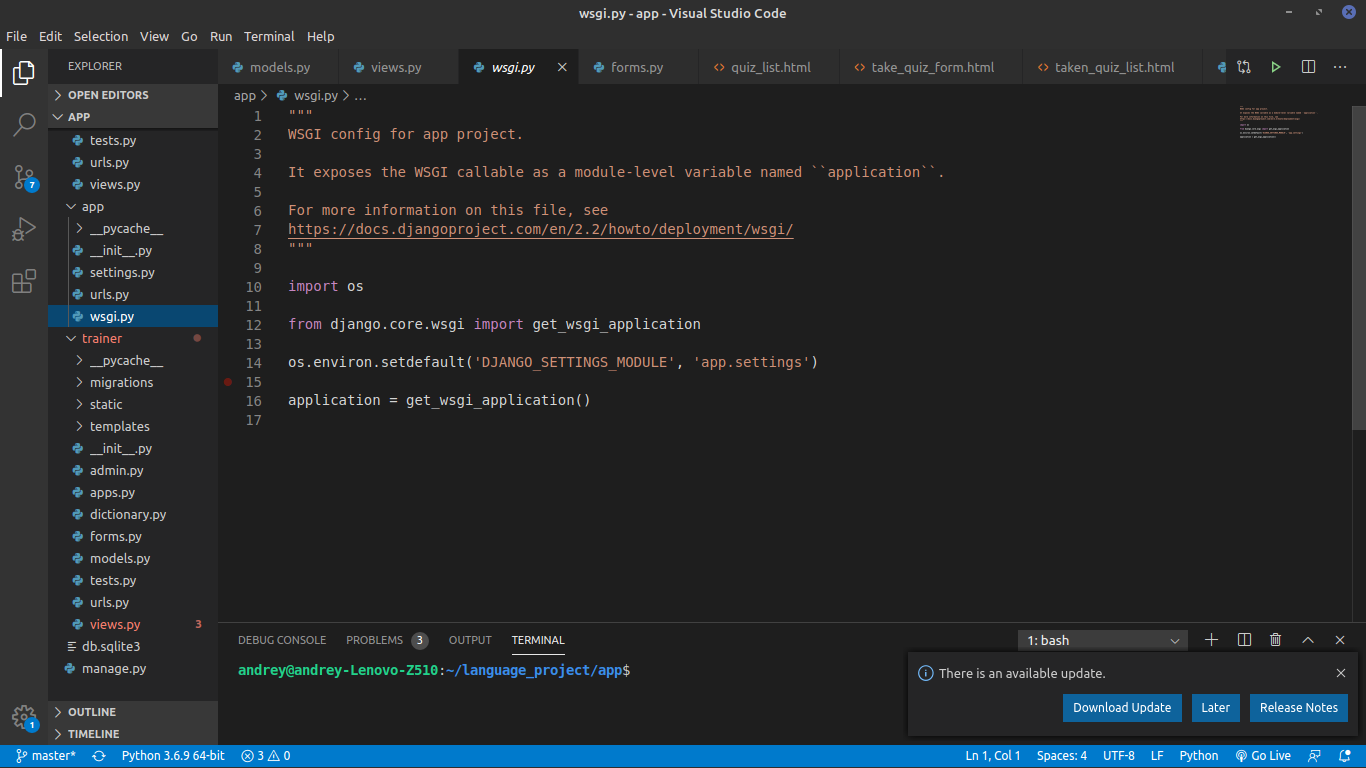
Когда к приложению приходит запрос, то URL dispatcher определяет, с каким ресурсом сопоставляется данный запрос и передает этот запрос выбранному ресурсу. Ресурс фактически представляет функцию или View, который получает запрос и определенным образом обрабатывает его. В процессе обработки View может обращаться к моделям и базе данных, получать из нее данные, или, наоборот, сохранять в нее данные. Результат обработки запроса отправляется обратно, и этот результат пользователь видит в своем браузере. Как правило, результат обработки запроса представляет сгенерированный html-код, для генерации которого применяются шаблоны (Template). Схема типичного приложения в Django представлен на рисунке 4.3.

Рисунок 4.3- MVT в курсовой работе

1. **ШАБЛОН ПРОЕКТИРОВАНИЯ ПРАКТИЧЕСКИХ фывв ЗАДАЧ**

При создании программных систем перед разработчиками часто встает проблема выбора тех или иных проектных решений. В этих случаях на помощь приходят паттерны.  Дело в том, что почти наверняка подобные задачи уже решались ранее и уже существуют хорошо продуманные элегантные решения, составленные экспертами. Если эти решения описать и систематизировать в каталоги, то они станут доступными менее опытным разработчикам, которые после изучения смогут использовать их как шаблоны или образцы для решения задач подобного класса. Паттерны как раз описывают решения таких повторяющихся задач [6].

Сообразное использование паттернов проектирования дает разработчику ряд неоспоримых преимуществ. Приведем некоторые из них. Модель системы, построенная в терминах паттернов проектирования, фактически является структурированным выделением тех элементов и связей, которые значимы при решении поставленной задачи. Помимо этого, модель, построенная с использованием паттернов проектирования, более проста и наглядна в изучении, чем стандартная модель. Тем не менее, несмотря на простоту и наглядность, она позволяет глубоко и всесторонне проработать архитектуру разрабатываемой системы с использованием специального языка. Применение паттернов проектирования повышает устойчивость системы к изменению требований и упрощает неизбежную последующую доработку системы.  
 Существуют несколько типов паттернов проектирования, каждый из которых предназначен для решения своего круга задач:

* Порождающие паттерны, предназначенные для создания новых объектов в системе;
* Структурные паттерны, решающие задачи компоновки системы на основе классов и объектов;
* Паттерны поведения, предназначенные для распределения обязанностей между объектами в системе;

Порождающие шаблоны — шаблоны проектирования, которые абстрагируют процесс инстанцирования. Они позволяют сделать систему независимой от способа создания, композиции и представления объектов. Шаблон, порождающий классы, использует наследование, чтобы изменять наследуемый класс, а шаблон, порождающий объекты, делегирует инстанцирование другому объекту. Существуют следующие порождающие шаблоны такие как: простая фабрика (Simple Factory), фабричный метод (Factory Method), абстрактная фабрика (Abstract Factory) и д.р.

Структурные шаблоны в основном связаны с композицией объектов, другими словами, с тем, как сущности могут использовать друг друга. Ещё одним объяснением было бы то, что они помогают ответить на вопрос «Как создать программный компонент?». Итак, структурные шаблоны — шаблоны проектирования, в которых рассматривается вопрос о том, как из классов и объектов образуются более крупные структуры. Примерами данного вида шаблонов являются адаптер (Adapter), мост (Bridge), компоновщик (Composite), декоратор (Decorator) и д.р.

Поведенческие шаблоны связаны с распределением обязанностей между объектами. Их отличие от структурных шаблонов заключается в том, что они не просто описывают структуру, но также описывают шаблоны для передачи сообщений / связи между ними. Или, другими словами, они помогают ответить на вопрос «Как запустить поведение в программном компоненте?». Поведенческие шаблоны — шаблоны проектирования, определяющие алгоритмы и способы реализации взаимодействия различных объектов и классов. К этому виду шаблонов относят цепочка обязанностей (Chain of Responsibility), команда (Command), итератор (Iterator), посредник (Mediator) и другие.

В Django реализован поведенческие паттерн Наблюдатель для организации сигналов, которые возникают, например, при добавлении записи в базу данных. Он создаёт механизм подписки, позволяющий одним объектам следить и реагировать на события, происходящие в других объектах.

Сигналы Django - это стратегия, позволяющая отделенным приложениям получать уведомления при наступлении определенных событий. Допустим, вы хотите аннулировать кэшированную страницу каждый раз, когда обновляется данный экземпляр модели, но в вашей базе кода есть несколько мест, где эту модель можно обновить. Вы можете сделать это, используя сигналы, перехватывая некоторые фрагменты кода, которые будут выполняться каждый раз, когда срабатывает метод сохранения этой конкретной модели. Другой распространенный случай - расширение пользовательского Django-пользователя с помощью стратегии профиля через отношение один-к-одному. Обычно мы используем «диспетчер сигналов» для прослушивания события post\_save пользователя, чтобы также обновить экземпляр профиля.

По сути система сигналов разделяется на два компонента: sender — компонент посылающий сигнал; receiver — компонент отвечающий за обработку сигнала. Сам сигнал «испускается» в ORM Django (точнее за это отвечает Signal Dispatcher), наша задача только реализовать приемник (ни или обработчик сигнала) данного типа. Для этого в Django есть декоратор reciever(signal, \*\*kwargs).

**ЗАКЛЮЧЕНИЕ**

Связи с реалиями сегодняшнего дня актуальность дистанционного обучения заключается в том, что результаты общественного прогресса, сегодня концентрируются в информационной сфере. Этап её развития в данный момент можно характеризовать как телекоммуникационный. Эта область общения, информации и знаний. Исходя из того, что профессиональные знания стареют очень быстро, необходимо их непрерывное совершенствование. Дистанционная форма обучения дает сегодня возможность создания систем массового непрерывного самообучения, всеобщего обмена информацией, независимо от наличия временных и пространственных поясов.

Потребителями образовательных услуг являются не только школьники и студенты, но и значительная часть взрослого населения страны. В результате в современном мире отмечается резкое увеличение спроса на образовательные услуги. В связи со всем вышесказанным появилась потребность в дистанционном виде обучения, который характеризуется большой гибкостью. Данные проект в достаточной мере способен предоставлять средства для удовлетворения этой потребности, а в частности изучения белорусского языка.  
 При написании курсового проекта были использованы ресурсы для изучения концепций и технологий объектно-ориентированного программирования и фреймворкa Django. Работа с ними происходила в среде разработки VS Code. Задача была реализована при использовании шаблона MVC, СУБД MySQL, языка разметки html, языка описания внешнего вида CSS и языка сценариев JavaScript. Также проект был занесён на онлайн-репозиторий GitHub.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Вайсфельд, М. Объектно-ориентированное мышление: учеб. пособие / Вайсфельд. М. — СПб.: Питер, 2014. — 304 с

[2] Django 2.1. Практика создания веб-сайтов на Python / В. Дронов // БХВ-Петербург – 2019 [Электронный ресурс]. – Режим доступа: https://codernet.ru/books/python/django\_2\_1\_praktika\_sozdaniya\_veb-sajtov\_na\_python/ – Дата доступа: 10.05.2020.

[3] Система управления базами данных MySQL [Электронный ресурс]. – Режим доступа: https://depix.ru/articles/sistema\_upravleniya\_bazami\_dannyh\_mysql

[4] Достоинства и недостатки дистрибутивов на ядре Linux [Электронный ресурс]. – Режим доступа: https://lumpics.ru/pros-and-cons-of-linux/

[5] Объектно-ориентированное программирование [Электронный ресурс]. – Режим доступа: <http://erud.by/object_orient_program/>

[6] Блог программиста, паттерны проектирования [Электронный ресурс]. – Режим доступа: <http://blog-programmista.ru/post/12-patterny-proektirovania.html>

[7] Django documentation [Электронный ресурс]. – Режим доступа: <https://docs.djangoproject.com/en/3.0/>

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Листинг кода программы**

trainer/view.py

|  |
| --- |
| from django.shortcuts import render,get\_object\_or\_404,redirect |
|  |

|  |
| --- |
| from django.contrib.auth.decorators import login\_required |
|  |

|  |
| --- |
| from .forms import FindWordForm, TakeQuizForm |
|  |

|  |
| --- |
| import pdb |
|  |

|  |
| --- |
| from .dictionary import Parcer |
|  |

|  |
| --- |
| from .models import Regulation |
|  |

|  |
| --- |
| from django.db import transaction |
|  |

|  |
| --- |
| from django.db.models import Count, Max |
|  |

|  |
| --- |
| from django.contrib.auth.mixins import LoginRequiredMixin |
|  |

|  |
| --- |
| from django.views.generic import ListView |
|  |

|  |
| --- |
| from django.views.generic.base import TemplateView |
|  |

|  |
| --- |
| from .models import Quiz, Student, TakenQuiz |
|  |

|  |
| --- |
| from django.contrib import messages |
|  |

|  |
| --- |
| from datetime import timedelta, datetime |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def index(request): |
|  |

|  |
| --- |
| return render( |
|  |

|  |
| --- |
| request, |
|  |

|  |
| --- |
| 'index.html', |
|  |

|  |
| --- |
| ) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class BoardTemplateView(LoginRequiredMixin, TemplateView): |
|  |

|  |
| --- |
| login\_url = '/account/login/' |
|  |

|  |
| --- |
| template\_name = 'dashboard.html' |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class RuleListView(LoginRequiredMixin, ListView): |
|  |

|  |
| --- |
| login\_url = '/account/login/' |
|  |

|  |
| --- |
| template\_name = 'list\_rules.html' |
|  |

|  |
| --- |
| model = Regulation |
|  |

|  |
| --- |
| context\_object\_name = 'rules' |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class SelectedRuleListView(LoginRequiredMixin, ListView): |
|  |

|  |
| --- |
| login\_url = '/account/login/' |
|  |

|  |
| --- |
| template\_name = 'full\_rules.html' |
|  |

|  |
| --- |
| model = Regulation |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def get\_context\_data(self, \*args, \*\*kwargs) : |
|  |

|  |
| --- |
| context = super() .get\_context\_data(\*args, \*\*kwargs) |
|  |

|  |
| --- |
| context['current\_rule'] = get\_object\_or\_404(Regulation, pk=self.kwargs['rules\_id']) |
|  |

|  |
| --- |
| return context |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| @login\_required (login\_url = '/account/login/') |
|  |

|  |
| --- |
| def dictionary(request): |
|  |

|  |
| --- |
| search\_word = '' |
|  |

|  |
| --- |
| if request.method == 'POST': |
|  |

|  |
| --- |
| form= FindWordForm(request.POST) |
|  |

|  |
| --- |
| if form.is\_valid(): |
|  |

|  |
| --- |
| search\_word= form.cleaned\_data['word'] |
|  |

|  |
| --- |
| parcer = Parcer() |
|  |

|  |
| --- |
| search\_word = parcer.getExplanationOfWord(parcer.getPageWitfoundedWord(search\_word)) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| else: |
|  |

|  |
| --- |
| form= FindWordForm() |
|  |

|  |
| --- |
| context= {'form': form, 'word': search\_word} |
|  |

|  |
| --- |
| return render(request, 'dictionary.html', context) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class QuizListView(LoginRequiredMixin, ListView): |
|  |

|  |
| --- |
| model = Quiz |
|  |

|  |
| --- |
| ordering = ('name', ) |
|  |

|  |
| --- |
| context\_object\_name = 'quizzes' |
|  |

|  |
| --- |
| template\_name = 'quiz\_list.html' |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def get\_queryset(self): |
|  |

|  |
| --- |
| student = self.request.user.student #“Аннотирует” каждый объект в QuerySet агрегированным значением (среднее, суииа и др.), которое будет вычислено из данных связанных объектов, которые связанны с объектами из``QuerySet``. |
|  |

|  |
| --- |
| taken\_quizzes = student.quizzes.values\_list('pk', flat=True) #достаем квизы, которые еще не выполнены и кол-во вопросов больше 0 |
|  |

|  |
| --- |
| queryset = Quiz.objects.exclude(pk\_\_in=taken\_quizzes) \ |
|  |

|  |
| --- |
| .annotate(questions\_count=Count('questions')) \ |
|  |

|  |
| --- |
| .filter(questions\_count\_\_gt=0)#достаем квизы, которые были выполнены не ранее 2 часов |
|  |

|  |
| --- |
| old\_tests = student.taken\_quizzes.values('quiz').annotate(max=Max('date')) \ |
|  |

|  |
| --- |
| .filter(max\_\_lt=datetime.now() - timedelta(hours=2)) \ |
|  |

|  |
| --- |
| .values\_list('quiz', flat=True) |
|  |

|  |
| --- |
| print(old\_tests) |
|  |

|  |
| --- |
| old\_tests\_queryset = Quiz.objects.filter(pk\_\_in=old\_tests) \ |
|  |

|  |
| --- |
| .annotate(questions\_count=Count('questions')).distinct()#?? |
|  |

|  |
| --- |
| queryset = queryset.union(old\_tests\_queryset) |
|  |

|  |
| --- |
| return queryset |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # в список квизов добавить, те которые старше 3 дней, перед выполнением проверить был ли этот кветс пройден ранее, |
|  |

|  |
| --- |
| # если так, тогда удалить все ответы студента, в пройденных тестах показывать еще и дату. |
|  |

|  |
| --- |
| class TakenQuizListView(LoginRequiredMixin, ListView): |
|  |

|  |
| --- |
| model = TakenQuiz |
|  |

|  |
| --- |
| context\_object\_name = 'taken\_quizzes' |
|  |

|  |
| --- |
| template\_name = 'taken\_quiz\_list.html' |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def get\_queryset(self): #Возвращает QuerySet который автоматически включает в выборку данные связанных объектов при выполнении запроса. |
|  |

|  |
| --- |
| queryset = self.request.user.student.taken\_quizzes.select\_related('quiz').order\_by('quiz\_\_name') |
|  |

|  |
| --- |
| return queryset |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| @login\_required |
|  |

|  |
| --- |
| def take\_quiz(request, pk): |
|  |

|  |
| --- |
| quiz = get\_object\_or\_404(Quiz, pk=pk) |
|  |

|  |
| --- |
| student = request.user.student |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if student.quizzes.filter(pk=pk).exists(): #проверяем квизы, которые послучайности запушены, но время блокировки еще не вышло |
|  |

|  |
| --- |
| if student.taken\_quizzes.filter(pk=pk,date\_\_gt=datetime.now() - timedelta(hours=2)) \ |
|  |

|  |
| --- |
| .values\_list('quiz', flat=True) \ |
|  |

|  |
| --- |
| .exists(): |
|  |

|  |
| --- |
| return render(request, 'taken\_quiz\_list.html') |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| total\_questions = quiz.questions.count() |
|  |

|  |
| --- |
| unanswered\_questions = student.get\_unanswered\_questions(quiz) |
|  |

|  |
| --- |
| total\_unanswered\_questions = unanswered\_questions.count() |
|  |

|  |
| --- |
| progress = 100 - round(((total\_unanswered\_questions - 1) / total\_questions) \* 100) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if request.method == 'POST': |
|  |

|  |
| --- |
| question = [] |
|  |

|  |
| --- |
| if request.POST.get('answer') is None: |
|  |

|  |
| --- |
| question = unanswered\_questions.first() |
|  |

|  |
| --- |
| else: |
|  |

|  |
| --- |
| for i in unanswered\_questions.filter(answers\_\_pk=request.POST['answer']): |
|  |

|  |
| --- |
| question = i |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| form = TakeQuizForm(question=question, data=request.POST) |
|  |

|  |
| --- |
| if form.is\_valid(): |
|  |

|  |
| --- |
| with transaction.atomic(): |
|  |

|  |
| --- |
| student\_answer = form.save(commit=False) |
|  |

|  |
| --- |
| student\_answer.student = student |
|  |

|  |
| --- |
| student\_answer.save() |
|  |

|  |
| --- |
| if student.get\_unanswered\_questions(quiz).exists(): |
|  |

|  |
| --- |
| return redirect('take\_quiz', pk) |
|  |

|  |
| --- |
| else: |
|  |

|  |
| --- |
| correct\_answers = student.quiz\_answers.filter(answer\_\_question\_\_quiz=quiz, |
|  |

|  |
| --- |
| taken\_quiz=None, |
|  |

|  |
| --- |
| answer\_\_is\_correct=True).count() |
|  |

|  |
| --- |
| score = round((correct\_answers / total\_questions) \* 100.0, 2) |
|  |

|  |
| --- |
| new\_taken\_quiz = TakenQuiz.objects.create(student=student, quiz=quiz, score=score) |
|  |

|  |
| --- |
| student.quiz\_answers.filter(answer\_\_question\_\_quiz=quiz, taken\_quiz=None) \ |
|  |

|  |
| --- |
| .update(taken\_quiz=new\_taken\_quiz.pk) |
|  |

|  |
| --- |
| if score < 50.0: |
|  |

|  |
| --- |
| messages.warning(request, 'У наступным разе будзе лепей %s лік %s.' |
|  |

|  |
| --- |
| % (quiz.name, score)) |
|  |

|  |
| --- |
| else: |
|  |

|  |
| --- |
| messages.success(request, 'Мае віншаванні, %s! Твой лік %s.' % (quiz.name, score)) |
|  |

|  |
| --- |
| return redirect('quiz\_list') |
|  |

|  |
| --- |
| else: |
|  |

|  |
| --- |
| unanswered\_questions = unanswered\_questions.order\_by('?') |
|  |

|  |
| --- |
| question = unanswered\_questions.first() |
|  |

|  |
| --- |
| form = TakeQuizForm(question=question) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return render(request, 'take\_quiz\_form.html', { |
|  |

|  |
| --- |
| 'quiz': quiz, |
|  |

|  |
| --- |
| 'question': question, |
|  |

|  |
| --- |
| 'form': form, |
|  |

|  |
| --- |
| 'progress': progress |
|  |

})

trainer/model.py

|  |
| --- |
| from django.db import models |
|  |

|  |
| --- |
| from django.contrib.auth.models import User |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Regulation(models.Model): |
|  |

|  |
| --- |
| title = models.CharField(max\_length=50,verbose\_name='Назва правіла') |
|  |

|  |
| --- |
| content = models.TextField(null=True, blank=True,verbose\_name='Тэкст') |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def \_\_str\_\_(self): |
|  |

|  |
| --- |
| return self.title |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Meta: |
|  |

|  |
| --- |
| verbose\_name\_plural='Правілы' |
|  |

|  |
| --- |
| verbose\_name='Правіла' |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Quiz(models.Model): |
|  |

|  |
| --- |
| rule = models.ForeignKey(Regulation, on\_delete=models.PROTECT,verbose\_name='Правіла') |
|  |

|  |
| --- |
| explanation = models.TextField(blank=True, verbose\_name='Агульнае заданне') |
|  |

|  |
| --- |
| name = models.CharField(max\_length=255) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def \_\_str\_\_(self): |
|  |

|  |
| --- |
| return self.explanation |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Meta: |
|  |

|  |
| --- |
| verbose\_name='Заданне для практыкавання' |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Task(models.Model): |
|  |

|  |
| --- |
| quiz = models.ForeignKey(Quiz, on\_delete=models.PROTECT, verbose\_name='Пытанне да задання', |
|  |

|  |
| --- |
| related\_name='questions') |
|  |

|  |
| --- |
| question = models.TextField(verbose\_name='Тэкст заданія', blank=True) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def \_\_str\_\_(self): |
|  |

|  |
| --- |
| return self.question |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Meta: |
|  |

|  |
| --- |
| verbose\_name\_plural='Заданні' |
|  |

|  |
| --- |
| verbose\_name='Заданне' |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Answer(models.Model): |
|  |

|  |
| --- |
| question = models.ForeignKey(Task, on\_delete=models.CASCADE, related\_name='answers') |
|  |

|  |
| --- |
| text = models.CharField('Адказ', max\_length=255) |
|  |

|  |
| --- |
| is\_correct = models.BooleanField('Правільны адказ', default=False) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def \_\_str\_\_(self): |
|  |

|  |
| --- |
| return self.text |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Student(models.Model): |
|  |

|  |
| --- |
| user = models.OneToOneField(User, on\_delete=models.CASCADE, primary\_key=True) |
|  |

|  |
| --- |
| quizzes = models.ManyToManyField(Quiz, through='TakenQuiz') |
|  |

|  |
| --- |
| # through Kласс модели, которая представляет связующую таблицу (связующая модель) |
|  |

|  |
| --- |
| # либо в виде ссьmки на него, либо в виде имени, представленном стро­кой. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def get\_unanswered\_questions(self, quiz): |
|  |

|  |
| --- |
| answered\_questions = self.quiz\_answers \ |
|  |

|  |
| --- |
| .filter(answer\_\_question\_\_quiz=quiz, taken\_quiz=None) \ |
|  |

|  |
| --- |
| .values\_list('answer\_\_question\_\_pk', flat=True) # получаем объекты через таблицу studentAnswer по ключу answer, |
|  |

|  |
| --- |
| # далее из таббл Answer по ключу question получаем сам запрос |
|  |

|  |
| --- |
| # массив из ключей отвеченных вопросов |
|  |

|  |
| --- |
| questions = quiz.questions.exclude(pk\_\_in=answered\_questions).order\_by('question') |
|  |

|  |
| --- |
| return questions |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def \_\_str\_\_(self): |
|  |

|  |
| --- |
| return self.user.username |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class TakenQuiz(models.Model): |
|  |

|  |
| --- |
| student = models.ForeignKey(Student, on\_delete=models.CASCADE, related\_name='taken\_quizzes') |
|  |

|  |
| --- |
| quiz = models.ForeignKey(Quiz, on\_delete=models.CASCADE, related\_name='taken\_quizzes') |
|  |

|  |
| --- |
| score = models.FloatField() |
|  |

|  |
| --- |
| date = models.DateTimeField(auto\_now\_add=True) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class StudentAnswer(models.Model): |
|  |

|  |
| --- |
| student = models.ForeignKey(Student, on\_delete=models.CASCADE, related\_name='quiz\_answers') |
|  |

|  |
| --- |
| answer = models.ForeignKey(Answer, on\_delete=models.CASCADE, related\_name='+') |
|  |

|  |
| --- |
| taken\_quiz = models.ForeignKey(TakenQuiz, on\_delete=models.CASCADE, |
|  |

|  |
| --- |
| related\_name='taken\_quiz\_answers', |
|  |

null=True)

trainer/urls.pu

|  |
| --- |
| from django.urls import path |
|  |

|  |
| --- |
| from . import views |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| urlpatterns = [ |
|  |

|  |
| --- |
| path('', views.index, name='index'), |
|  |

|  |
| --- |
| path('dashboard/', views.BoardTemplateView.as\_view(), name='dashboard'), |
|  |

|  |
| --- |
| path('dashboard/dictionary/', views.dictionary, name='dictionary'), |
|  |

|  |
| --- |
| path('dashboard/rules/', views.RuleListView.as\_view(), name='list\_of\_rules'), |
|  |

|  |
| --- |
| path('dashboard/rules/<int:rules\_id>', views.SelectedRuleListView.as\_view(), name='full\_rules'), |
|  |

|  |
| --- |
| path('dashboard/quiz/', views.QuizListView.as\_view(), name='quiz\_list'), |
|  |

|  |
| --- |
| path('dashboard/taken/', views.TakenQuizListView.as\_view(), name='taken\_quiz\_list'), |
|  |

path('dashboard/quiz/<int:pk>/', views.take\_quiz, name='take\_quiz'),

trainer/forms.py

|  |
| --- |
| from django import forms |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| from .models import (Answer, StudentAnswer) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class FindWordForm(forms.Form): |
|  |

|  |
| --- |
| word = forms.CharField(max\_length=100) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class TakeQuizForm(forms.ModelForm): |
|  |

|  |
| --- |
| answer = forms.ModelChoiceField( |
|  |

|  |
| --- |
| queryset=Answer.objects.none(), |
|  |

|  |
| --- |
| widget=forms.RadioSelect(), |
|  |

|  |
| --- |
| required=True, |
|  |

|  |
| --- |
| empty\_label=None, |
|  |

|  |
| --- |
| label="Тваі адказы") |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Meta: |
|  |

|  |
| --- |
| model = StudentAnswer |
|  |

|  |
| --- |
| fields = ('answer', ) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def \_\_init\_\_(self, \*args, \*\*kwargs): |
|  |

|  |
| --- |
| question = kwargs.pop('question') |
|  |

|  |
| --- |
| super().\_\_init\_\_(\*args, \*\*kwargs) |
|  |

self.fields['answer'].queryset = question.answers.order\_by('?')

trainer/dictionary.py

|  |
| --- |
| import requests |
|  |

|  |
| --- |
| from bs4 import BeautifulSoup |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Parcer(): |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def getPageWitfoundedWord(self, word): |
|  |

|  |
| --- |
| payload = {'term': word, 'lang': 'beld'} |
|  |

|  |
| --- |
| r = requests.get('https://www.skarnik.by/search', params = payload) |
|  |

|  |
| --- |
| soup = BeautifulSoup(r.text, "lxml") |
|  |

|  |
| --- |
| return soup |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def getExplanationOfWord(self, soup): |
|  |

|  |
| --- |
| data=[] |
|  |

|  |
| --- |
| try: |
|  |

|  |
| --- |
| data = list(filter(None, soup.find(id="trn").text.split('\xa0'))) |
|  |

|  |
| --- |
| data[0] = soup.title.text |
|  |

|  |
| --- |
| except: |
|  |

|  |
| --- |
| tags = soup.div.find\_all(class\_="span10") |
|  |

|  |
| --- |
| for tag in tags: |
|  |

|  |
| --- |
| try: |
|  |

|  |
| --- |
| data.append(tag.find('p').text) |
|  |

|  |
| --- |
| break |
|  |

|  |
| --- |
| except: |
|  |

|  |
| --- |
| continue |
|  |

return data

account/view.py

|  |
| --- |
| from django.http import HttpResponse |
|  |

|  |
| --- |
| from django.shortcuts import render,redirect |
|  |

|  |
| --- |
| from django.contrib.auth import authenticate, login,logout |
|  |

|  |
| --- |
| from .forms import LoginForm |
|  |

|  |
| --- |
| from .forms import RegisterForms, UserUpdateForm, ProfileUpdateForm |
|  |

|  |
| --- |
| from django.contrib.auth.decorators import login\_required |
|  |

|  |
| --- |
| from datetime import datetime |
|  |

|  |
| --- |
| from django.contrib import messages |
|  |

|  |
| --- |
| from math import floor |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def logout\_view(request): |
|  |

|  |
| --- |
| logout(request) |
|  |

|  |
| --- |
| return redirect("/trainer") |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def user\_login(request): |
|  |

|  |
| --- |
| if request.method == 'POST': |
|  |

|  |
| --- |
| form = LoginForm(request.POST) #create form |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if form.is\_valid(): |
|  |

|  |
| --- |
| cd = form.cleaned\_data |
|  |

|  |
| --- |
| user = authenticate(username=cd['username'], password=cd['password']) # find user |
|  |

|  |
| --- |
| if user is not None: |
|  |

|  |
| --- |
| if user.is\_active: |
|  |

|  |
| --- |
| login(request, user) #create session |
|  |

|  |
| --- |
| return redirect("/trainer/dashboard")#HttpResponse('OK') |
|  |

|  |
| --- |
| else: |
|  |

|  |
| --- |
| return HttpResponse('Disabled account') |
|  |

|  |
| --- |
| else: |
|  |

|  |
| --- |
| return HttpResponse('Invalid login') |
|  |

|  |
| --- |
| else: |
|  |

|  |
| --- |
| form = LoginForm() |
|  |

|  |
| --- |
| return render(request, 'account/login\_form.html', {'form': form}) |
|  |

|  |
| --- |
| #@login\_required |
|  |

|  |
| --- |
| #def dashboard(request): |
|  |

|  |
| --- |
| # return render(request, 'trainer/dashboard.html', {'section': 'dashboard'}) |
|  |

|  |
| --- |
| def register(request): |
|  |

|  |
| --- |
| if request.method == "POST": |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| form = RegisterForms(request.POST) |
|  |

|  |
| --- |
| if form.is\_valid(): |
|  |

|  |
| --- |
| form.save() |
|  |

|  |
| --- |
| messages.success(request, f'Акаўнт створаны ') |
|  |

|  |
| --- |
| return redirect("/account/login") |
|  |

|  |
| --- |
| else: |
|  |

|  |
| --- |
| form = RegisterForms() |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return render(request, "account/registration/registrator.html", {'form':form}) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| @login\_required |
|  |

|  |
| --- |
| def edit\_profile(request): |
|  |

|  |
| --- |
| if request.method == 'POST': |
|  |

|  |
| --- |
| form = UserUpdateForm(request.POST, instance=request.user) |
|  |

|  |
| --- |
| profile\_form = ProfileUpdateForm(request.POST, |
|  |

|  |
| --- |
| request.FILES, |
|  |

|  |
| --- |
| instance=request.user.profile) # request.FILES is show the selected image or file |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if form.is\_valid() and profile\_form.is\_valid(): |
|  |

|  |
| --- |
| form.save() |
|  |

|  |
| --- |
| profile\_form.save(False)# it will return an object that hasn’t yet been saved to the database. |
|  |

|  |
| --- |
| messages.success(request, f'Ваш акаўнт зменены') |
|  |

|  |
| --- |
| #custom\_form.user = user\_form |
|  |

|  |
| --- |
| #custom\_form.save() |
|  |

|  |
| --- |
| return redirect('/account/edit\_profile/') |
|  |

|  |
| --- |
| #return HttpResponse('ok') |
|  |

|  |
| --- |
| else: |
|  |

|  |
| --- |
| form = UserUpdateForm(instance=request.user) |
|  |

|  |
| --- |
| profile\_form = ProfileUpdateForm(instance=request.user.profile) |
|  |

|  |
| --- |
| args = {} |
|  |

|  |
| --- |
| args['form'] = form |
|  |

|  |
| --- |
| args['profile\_form'] = profile\_form |
|  |

|  |
| --- |
| exp = ExperienceUser() |
|  |

|  |
| --- |
| args['level'] = exp.getlevel(request.user.profile.experience) |
|  |

|  |
| --- |
| args['rank'] = exp.getRank(args['level']) |
|  |

|  |
| --- |
| return render(request, 'account/profile/edit\_profile.html', args) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class ExperienceUser: |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def getlevel(self, exp): |
|  |

|  |
| --- |
| return floor(pow(exp / 100, 7/10)) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def getExperience(self, level): |
|  |

|  |
| --- |
| return round (pow(level, 10/7) \* 100) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def getRank(self, level): |
|  |

|  |
| --- |
| rank = "" |
|  |

|  |
| --- |
| if level < 5: |
|  |

|  |
| --- |
| rank = "Навічок" |
|  |

|  |
| --- |
| elif level < 15: |
|  |

|  |
| --- |
| rank = "Карыстальнік" |
|  |

|  |
| --- |
| elif level < 30: |
|  |

|  |
| --- |
| rank = "Актыўны" |
|  |

|  |
| --- |
| elif level < 45: |
|  |

|  |
| --- |
| rank = "Гігант думак" |
|  |

|  |
| --- |
| elif level < 60: |
|  |

|  |
| --- |
| rank = "Старэйшына" |
|  |

|  |
| --- |
| elif level < 75: |
|  |

|  |
| --- |
| rank = "Мовазнаўца" |
|  |

|  |
| --- |
| elif level < 100: |
|  |

|  |
| --- |
| rank = "Ветэран" |
|  |

|  |
| --- |
| elif level < 125: |
|  |

|  |
| --- |
| rank = "Генерал" |
|  |

|  |
| --- |
| else: |
|  |

|  |
| --- |
| rank = "Францыск Скарына" |
|  |

return rank

account/models.py

|  |
| --- |
| from django.db import models |
|  |

|  |
| --- |
| from django.contrib.auth.models import User |
|  |

|  |
| --- |
| from django.db.models.signals import post\_save |
|  |

|  |
| --- |
| from django.dispatch import receiver |
|  |

|  |
| --- |
| from PIL import Image |
|  |

|  |
| --- |
| from trainer.models import Student, TakenQuiz |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Profile(models.Model): |
|  |

|  |
| --- |
| user = models.OneToOneField(User, on\_delete=models.CASCADE) |
|  |

|  |
| --- |
| experience = models.PositiveIntegerField(blank=True, default=0, verbose\_name= 'Вопыт карыстальніка') |
|  |

|  |
| --- |
| image = models.ImageField(blank=True, |
|  |

|  |
| --- |
| null = True, |
|  |

|  |
| --- |
| width\_field = "width\_field", |
|  |

|  |
| --- |
| height\_field = "height\_field", |
|  |

|  |
| --- |
| default = "default.png", |
|  |

|  |
| --- |
| upload\_to = "profile\_pics") |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| height\_field = models.IntegerField(default=0) |
|  |

|  |
| --- |
| width\_field = models.IntegerField(default=0) |
|  |

|  |
| --- |
| birth\_date = models.DateField(null=True, blank=True) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # def save(self, \*args, \*\*kwargs): |
|  |

|  |
| --- |
| # super(Profile, self).save(\*args, \*\*kwargs) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # img = Image.open(self.image.path) |
|  |

|  |
| --- |
| # if img.height > 400 or img.width > 400: |
|  |

|  |
| --- |
| # output\_size = (400, 400) |
|  |

|  |
| --- |
| # img.thumbnail(output\_size) |
|  |

|  |
| --- |
| # img.save(self.image.path) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| @receiver(post\_save, sender=User) |
|  |

|  |
| --- |
| def create\_user\_profile(sender, instance, created, \*\*kwargs): |
|  |

|  |
| --- |
| if created: |
|  |

|  |
| --- |
| Profile.objects.create(user=instance) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| @receiver(post\_save, sender=User) |
|  |

|  |
| --- |
| def create\_user\_student(sender, instance, created, \*\*kwargs): |
|  |

|  |
| --- |
| if created: |
|  |

|  |
| --- |
| Student.objects.create(user=instance) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| @receiver(post\_save, sender=User) |
|  |

|  |
| --- |
| def save\_user\_profile(sender, instance, \*\*kwargs): |
|  |

|  |
| --- |
| instance.profile.save() |
|  |

|  |
| --- |
| #Мы «зацепили» create\_user\_profile() и save\_user\_profile() к событию сохранения модели User. |
|  |

|  |
| --- |
| #сигнала post\_save. |
|  |

|  |
| --- |
| @receiver(post\_save, sender = TakenQuiz) |
|  |

|  |
| --- |
| def add\_score(instance, \*\*kwargs): |
|  |

|  |
| --- |
| profile = instance.student.user.profile |
|  |

|  |
| --- |
| profile.experience += 10 |
|  |

profile.save()

account/forms.py

|  |
| --- |
| from django import forms |
|  |

|  |
| --- |
| from django.contrib.auth.forms import UserCreationForm |
|  |

|  |
| --- |
| from django.contrib.auth.models import User |
|  |

|  |
| --- |
| from .models import Profile |
|  |

|  |
| --- |
| from django.forms import ModelForm |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class LoginForm(forms.Form): |
|  |

|  |
| --- |
| username = forms.CharField() |
|  |

|  |
| --- |
| password = forms.CharField(widget=forms.PasswordInput) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class RegisterForms (UserCreationForm): |
|  |

|  |
| --- |
| email = forms.EmailField() |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class Meta: |
|  |

|  |
| --- |
| model = User |
|  |

|  |
| --- |
| fields = ["username", "email", "password1","password2"] |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class UserUpdateForm(ModelForm): |
|  |

|  |
| --- |
| class Meta: |
|  |

|  |
| --- |
| model = User |
|  |

|  |
| --- |
| #email = forms.CharField(widget=forms.TextInput(attrs={'placeholder': 'Email'})) |
|  |

|  |
| --- |
| fields = ["username",'email'] |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class ProfileUpdateForm(ModelForm): |
|  |

|  |
| --- |
| class Meta: |
|  |

|  |
| --- |
| model = Profile |
|  |

fields = ['image', 'birth\_date']

account/urls.py

|  |
| --- |
| from django.urls import path |
|  |

|  |
| --- |
| from . import views |
|  |

|  |
| --- |
| from django.contrib.auth import views as auth\_views |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| urlpatterns = [ |
|  |

|  |
| --- |
| path ('login/', auth\_views.LoginView.as\_view(template\_name = "account/registration/login.html"), name = 'user login'), |
|  |

|  |
| --- |
| path ('register/',views.register, name = 'registrate'), |
|  |

|  |
| --- |
| path ('logout/', views.logout\_view, name = 'user\_logout'), |
|  |

|  |
| --- |
| path ('edit\_profile/', views.edit\_profile, name = 'edit\_profile'), |
|  |

]

**ПРИЛОЖЕНИЕ Б**

**(обязательное)**

**Графический материал**