

Оглавление

TODO

Вступление

В современном мире, когда технологии развиваются с невероятной скоростью, многие сферы жизни человека претерпевают кардинальные изменения. Одной из таких сфер является автомобильный транспорт и обслуживание автомобилей. С ростом числа автомобилей на дорогах, увеличивается и потребность в качественных автосервисах, которые могут удовлетворить потребности автовладельцев в ремонте, обслуживании и диагностике.

Актуальность темы данной дипломной работы заключается в том, что создание удобного и эффективного веб-приложения для поиска автосервисов может значительно упростить жизнь автовладельцам, предоставив им доступ к актуальной информации о ближайших автосервисах, их услугах, ценах и отзывах других клиентов. Кроме того, такое приложение может помочь автосервисам увеличить свою клиентскую базу, улучшить качество обслуживания и оптимизировать рабочие процессы.

Целью данной дипломной работы является разработка веб-приложения, которое будет обеспечивать эффективный поиск автосервисов, а также предоставление пользователям подробной информации о каждом автосервисе. Для достижения данной цели необходимо решить следующие задачи:

1. Проанализировать существующие аналоги веб-приложений для поиска автосервисов и выявить их недостатки.
2. Разработать архитектуру веб-приложения с учетом современных требований и стандартов веб-разработки.
3. Реализовать функционал поиска и фильтрации автосервисов по различным параметрам, таким как местоположение, предоставляемые услуги, стоимость и отзывы клиентов.
4. Обеспечить интеграцию с внешними сервисами для получения актуальной информации об автосервисах и отзывов клиентов.
5. Разработать удобный пользовательский интерфейс, который позволит пользователям быстро и легко находить необходимую информацию.
6. Протестировать разработанное веб-приложение на предмет его эффективности, надежности и удобства использования.

В ходе работы над проектом будут использоваться современные технологии, такие как HTML, CSS, JavaScript, React.js, Node.js, MongoDB, Bootstrap, jQuery и другие. Также будет уделено внимание вопросам безопасности и масштабируемости веб-приложения.

Разработанное веб-приложение для поиска автосервисов будет доступно для использования широкой аудитории автовладельцев, а также сможет стать полезным инструментом для автосервисов в продвижении своих услуг и улучшении качества обслуживания клиентов. Результаты данной дипломной работы могут быть использованы как в коммерческих, так и в некоммерческих целях, способствуя развитию отрасли автосервиса и улучшению качества жизни автовладельцев.

Описание проекта

Цель проекта

Целью проекта является разработка веб приложения на основе фронтенд фреймворка REACT и бэкенд фреймворка Django. Приложение должно удовлетворять требованиям и иметь минимальный необходимый функционал. Этот функционал должен включать:

- Регистрация пользователя в приложении
- Регистрация владельца автосервиса в приложении
- Возможность изменения данных пользователя в личном кабинете
- Возможность изменения данных владельца автосервиса в личном кабинете
- Возможность просмотра ближайших автосервисов пользователем
- Возможность отправки заявки на запись на обслуживание в автосервис
- Возможность просмотра заявок на ремонт владельцем автосервиса
- Возможность вести переписку с клиентом по отправленной заявке непосредственно в приложении
- Возможность изменять статус заявки
- Возможность закрывать заявку с различными статусами

Структура проекта

Проект размещен в нескольких репозиториях

- [backend](#)
- [frontend](#)
- [gateway](#)
- [QA](#)
- [api-docs](#)
- [car-service](#)
- [tests](#)

Основными репозиториями являются [backend](#) и [frontend](#).

В репозитории [frontend](#) размещен код для фронтенд сайта. Разработкой фронтенд части приложения занимается команда фронтенд разработки.

Фронтенд приложения разработан с использованием технологий Type Script и REACT. Фронтенд часть развернута в Docker контейнере так как и Nginx сервер, который отвечает за передачу статического контента.

Как бэкенд разработчик, моя роль в проекте состоит в разработке API и бэкенд части приложения. Бэкенд часть приложения написана на языке программирования Python. Для создания приложения использован бэкенд фреймворк Django. Также в процессе работы на бэкенд частью использованы следующие технологии и инструменты:

Технологии и инструменты

- Linux - операционная система на которой работает приложение. Также я использую Linux как операционную систему на моей рабочей станции. Я использую [Fedora Workstation](#).
- Docker - для размещения готового приложения на сервере и для облегчения разработки путем создания локальных контейнеров для тестирования работы как бэкенд так и фронтенд частей

приложения.

- GitHub Actions - для автоматизации развертывания приложения на сервере и организации CI-CD pipeline.
- Django REST framework - для создания API приложения.
- DRF spectacular - для создания документации к API приложения.
- Poetry - для управления зависимостями и сборки проекта.
- flake8 - линтер для Python кода
- black - для форматирования Python кода
- Pyright - Языковой сервер для авто-дополнения в редакторе кода и статического анализа Python кода.
- Neovim - мой редактор кода
- Postman - для тестирования API приложения.

В этой пояснительной записке далее я более подробно разберу использование каждого из этих инструментов и технологий.

Подготовка окружения для разработки приложения

Выбор операционной системы

Linux выбирают в качестве операционной системы для серверов в Интернете по следующим причинам:

- Открытость: Linux является открытым исходным кодом, что позволяет пользователям и разработчикам свободно изучать, модифицировать и распространять систему. Это обеспечивает гибкость и возможность адаптировать систему под индивидуальные потребности.
- Бесплатное распространение: Linux предлагается бесплатно, что снижает затраты на внедрение и поддержку сервера.
- Поддержка сообщества: Linux имеет огромное и активное сообщество разработчиков, пользователей и поставщиков, что обеспечивает поддержку и развитие системы.
- Безопасность: В Linux используется модель безопасности на основе привилегий, которая предотвращает несанкционированный доступ к системе. Кроме того, ядро Linux разработано таким образом, чтобы минимизировать уязвимости и эксплойты.
- Надежность: Linux зарекомендовал себя как стабильная и надежная операционная система для серверов. Он обеспечивает непрерывную работу и устойчивость к сбоям, что особенно важно для онлайн-сервисов и веб-приложений.
- Масштабируемость: Linux легко масштабируется для работы на различных типах серверов, от небольших и недорогих до высокопроизводительных и мощных. Это позволяет использовать Linux для различных задач, включая веб-хостинг, облачные вычисления, базы данных и многое другое.
- Совместимость: Linux поддерживает множество аппаратных и программных компонентов, что делает его совместимым с широким спектром оборудования. Это упрощает процесс развертывания и поддержки сервера.
- Гибкость и настраиваемость: Linux предлагает широкий выбор дистрибутивов и пакетов, что позволяет быстро и легко адаптировать систему для решения конкретных задач.

Варианты установки Linux для разработки приложения

Варианты установки Linux при разработке на этой платформе могут быть разными, в зависимости от требований к оборудованию, удобству использования и доступности инструментов. Вот несколько вариантов:

- Использование виртуальной машины (VM): Этот вариант подходит для тех, кто работает на платформе Windows или macOS и хочет протестировать свое приложение на Linux без необходимости установки этой операционной системы на свой компьютер. Виртуальная машина позволяет создать изолированную среду Linux, которую можно запускать на компьютере без влияния на основную операционную систему. Преимущества: простота использования, возможность легкого переключения между разными версиями Linux, совместимость с различными платформами. Недостатки: производительность может быть ниже, чем при использовании реальной установки Linux, некоторые приложения могут не работать должным образом.
- Установка Linux на отдельный жесткий диск или раздел: Этот вариант идеально подходит для тех, кому нужно постоянное и стабильное окружение для разработки на Linux. Преимущества: стабильная среда, возможность использовать все доступные инструменты Linux, высокая производительность. Недостатки: требуется больше времени на первоначальную настройку, нужен отдельный компьютер или отдельный раздел на компьютере.
- Использование облачной платформы: Некоторые облачные платформы, такие как Amazon Web Services (AWS), Microsoft Azure и Google Cloud Platform, предлагают предустановленные образы Linux для разработки и тестирования. Преимущества: возможность быстро развернуть среду Linux в облаке, масштабируемость, доступность инструментов для разработки. Недостатки: возможно, потребуется оплатить подписку на облако, производительность может зависеть от качества соединения с интернетом.
- Использование контейнеров: Это относительно новый подход к развертыванию и управлению приложениями, который позволяет упаковывать приложение и все его зависимости в один легкий и переносимый контейнер. Преимущества: быстрое развертывание и масштабирование, легкое обновление и миграция приложений. Недостатки: некоторые инструменты могут быть сложнее в использовании, чем виртуальные машины.

Установка нужной версии python

Проект основан на версии python 3.9.18. На Fedora 39 установлена версия python 12. Для того чтобы запустить версию python 3.9.18 необходимо использовать утилиту `pyenv`.

`pyenv` позволяет легко переключаться между несколькими версиями Python. Он прост, ненавязчив и следует традициям UNIX, когда инструменты одного назначения хорошо справляются с одной задачей.

Установка Poetry

Poetry помогает объявлять, управлять и устанавливать зависимости для Python-проектов, гарантируя, что у вас везде будет правильный стек.

Poetry заменяет `setup.py`, `requirements.txt`, `setup.cfg`, `MANIFEST.in` и `Pipfile` на простой формат проекта `pyproject.toml`.

Установка: Для начала работы с Python Poetry, вам нужно установить его. Это можно сделать несколькими способами:

- Если виртуальное окружение проекта уже создано, то poetry можно установить в него с помощью команды `pip: pip install poetry`
- `pipx` используется для глобальной установки приложений Python CLI, при этом изолируя их в виртуальных средах. `pipx` будет управлять обновлениями и удалениями, когда используется для установки Poetry.
- Poetry предоставляет пользовательскую программу установки, которая установит Poetry в новую виртуальную среду и позволит Poetry управлять своим окружением.

Я воспользовался официальным установщиком poetry.

Для это я скачал и запустил скрипт установки с официального сайта:

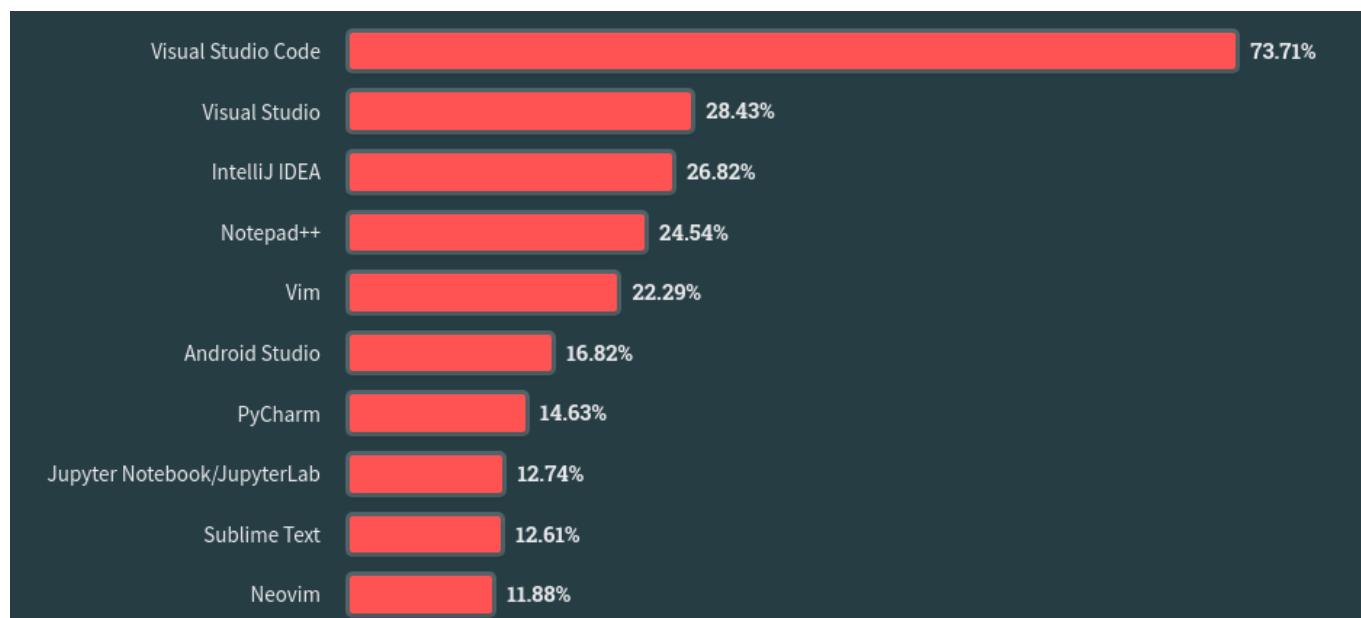
```
curl -sSL https://install.python-poetry.org | python3 -
```

Установка редактора

Каждый год на сайте StackOverflow проходит опрос пользователей сайта на темы связанные с разработкой программного обеспечения. В 2022 году в нем приняло участие более 74 тысячи человек. Stack Overflow survey

На вопрос какую среду разработки используют пользователи сайта были получены следующие результаты:

Visual Studio Code с большим отрывом опережает другие редакторы кода. Но интересно что редактор Vim занял пятое место и обошел другие более современные IDE.



Преимущества Vim:

Скорость редактирования кода

Визитной карточкой Vim является система горячих клавиш. В отличие от “обычных” редакторов текста, Vim использует систему режимов. В обычном режиме клавиши на клавиатуре используются для перемещения по тексту. Чтобы вводить текст нужно включить режим ввода текста.

Подобный подход позволяет отказаться от использования мыши для выделения и перемещения по тексту. Все действия выполняются с клавиатуры а руки не отрываются от среднего ряда клавиатуры.

В таком режиме выделение и редактирование текста происходит быстрее чем при использовании мыши. Например в случае если необходимо изменить текст внутри круглых скобок нужно набрать короткую фразу `ci()`. Текст внутри скобок будет удален и редактор перейдет в режим ввода текста.

Грамматика команд Vim может быть освоена в течении недели. После чего она поможет сэкономить гораздо большее времени затраченного на ее изучение. Использование горячих клавиш Vim в других программах

Клавиши Vim могут помочь не только при работе в самом редакторе Vim. Другие программы также имеют возможность использовать Vim режимы и клавиши. Zsh, Obsidian, Logseq, Emacs, Visual studio Code и продукты JetBrains - эти программы имеют возможность использовать горячие клавиши Vim в своей работе.

Если уметь работать в Vim даже при переходе на новую среду разработки можно будет продолжить использовать известные комбинации клавиш. Не придется тратить время чтобы запомнить либо переназначить клавиши в новой программе.

Близость к терминалу

Vim это консольное приложение. Это означает что оно запускается непосредственно в терминале. Что это значит для пользователя? Даже если ему придется работать с текстом на удаленном сервере, он сможет использовать Vim. Ему не придется устанавливать редактор, Vim или Vi предустановлены на большинстве Unix like систем.

Работа Vim в терминале означает что при работе с ним можно использовать любые команды из терминала.

- Можно вставить путь рабочей директории сразу в текст.
- Можно форматировать и фильтровать текст при помощи программы `awk`.
- Можно выполнять Git команды непосредственно в редакторе.

Vim позволяет по полной использовать `coreutils`.

Работа в связке с TMUX

В связке с мультиплексором терминалов Vim превращается в мощную среду разработки в которой можно работать над проектами, сохранять и восстанавливать состояние среды разработки между сессиями.

Тестирование и дебагинг кода может выполняться непосредственно в терминале. При этом переключение между редактором и процессом в консоли практически мгновенное.

Конфигурирование в текстовых файлах

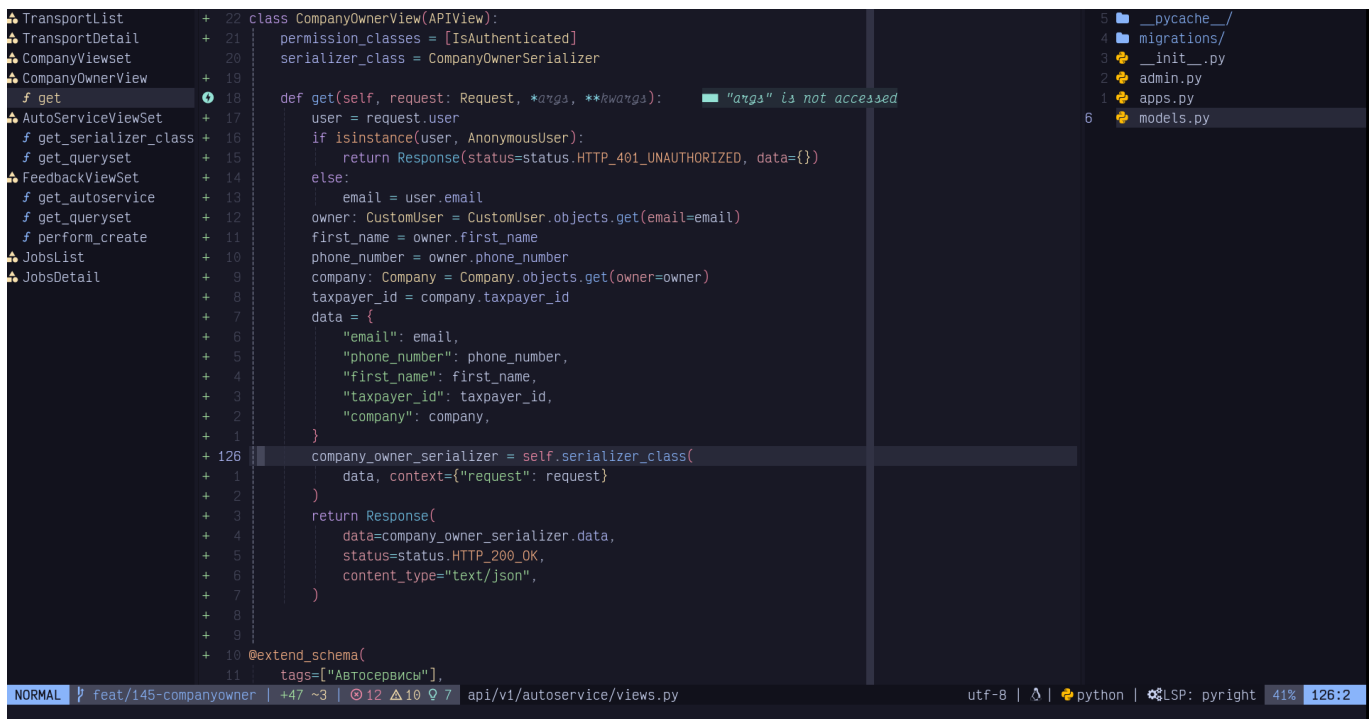
Вим конфигурируется в текстовых файлах при помощи Vim script или Lua. По началу это может показаться странным тем пользователям, кто привык к выбору настроек в графическом интерфейсе. Но такой способ конфигурации является более простым и понятным. На пользователя не вываливается разом огромный набор возможных настроек. Пользователь сам выбирает какие

настройки добавить в конфигурацию. Он может постепенно разобратся что дают ему выбранные им опции.

Изменения в текстовых файлах легко контролировать при помощи Git. В случае если изменение в настройках не устроило пользователя, оно может быть легко возвращено назад к предыдущему состоянию.

Большое число плагинов и расширений

В Vim есть большое число различных плагинов и расширений. Они расширяют функционал работы редактора и превращают его в полноценную среду разработки. LSP, completions, tree-sitter все это есть в Vim. В зависимости от потребностей пользователя Vim может быть как простым текстовым редактором так и не уступать в функциональности продуктам JetBrains.



```
TransportList + 22 class CompanyOwnerView(APIView):
TransportDetail + 21     permission_classes = [IsAuthenticated]
CompanyViewSet + 20     serializer_class = CompanyOwnerSerializer
CompanyOwnerView + 19
+ 18     def get(self, request: Request, *args, **kwargs):
+ 17         user = request.user
+ 16         if isinstance(user, AnonymousUser):
+ 15             return Response(status=status.HTTP_401_UNAUTHORIZED, data={})
+ 14         else:
+ 13             email = user.email
+ 12             owner: CustomUser = CustomUser.objects.get(email=email)
+ 11             first_name = owner.first_name
+ 10             phone_number = owner.phone_number
+ 9             company: Company = Company.objects.get(owner=owner)
+ 8             taxpayer_id = company.taxpayer_id
+ 7             data = {
+ 6                 "email": email,
+ 5                 "phone_number": phone_number,
+ 4                 "first_name": first_name,
+ 3                 "taxpayer_id": taxpayer_id,
+ 2                 "company": company,
+ 1             }
+ 126         company_owner_serializer = self.serializer_class(
+ 1             data, context={"request": request}
+ 2         )
+ 3         return Response(
+ 4             data=company_owner_serializer.data,
+ 5             status=status.HTTP_200_OK,
+ 6             content_type="text/json",
+ 7         )
+ 8
+ 9     @extend_schema(
+ 10         tags=["Автосервисы"],
+ 11         tags=["Автосервисы"],
```

Установка проекта

Создание проекта при помощи Poetry

Описание: Python Poetry - это утилита для управления зависимостями и автоматизации сборки проектов на языке программирования Python. Она позволяет легко устанавливать, обновлять и управлять зависимостями вашего проекта, а также автоматизировать процесс сборки и публикации вашего кода. Руководство: Создание нового проекта: После установки Python Poetry вы можете создать новый проект. Для этого используйте команду poetry: poetry new my_project Управление зависимостями: После создания проекта вы можете добавить зависимости с помощью файла pyproject.toml. Этот файл находится в корне вашего проекта и содержит информацию о зависимостях, а также другие настройки проекта. Добавить зависимость можно с помощью команды poetry add: poetry add some-dependency Обновление зависимостей: Чтобы обновить зависимости проекта, используйте команду poetry update: poetry update Публикация проекта: Вы можете опубликовать свой проект на PyPi (Python Package Index) или другом хранилище с помощью команды publish: poetry publish

Файл pyproject.toml

Сборка проекта: Python Poetry также может автоматизировать процесс сборки вашего проекта. Для этого нужно создать файл с расширением `.poetry.lock` в корне проекта. В этом файле будет храниться информация о зависимостях и их версиях. Затем вы можете настроить процесс сборки с помощью файла `build.rs`:

```
[build-system]
requires = ["poetry-core>=1.0.0"]
build-backend = "poetry.core.masonry.api"

[tool.poetry]
name = "my_project"
version = "0.1.0"
description = "My project description"
authors = ["Your Name <your@email.com>"]

[tool.build]
publish = true
```

Использование виртуальных окружений: Python Poetry поддерживает использование виртуальных окружений. Вы можете создать новое виртуальное окружение с помощью команды `poetry env use`:

```
poetry env use 3.9
```

Настройка: Python Poetry позволяет настроить множество параметров вашего проекта. Вы можете сделать это, отредактировав файл `pyproject.toml`: `[tool.poetry.source] remotes = ["https://example.com/simple/"]`

Определение зависимостей

При работе с `poetry` все зависимости явно указываются в файле `pyproject.toml`. При этом `poetry` позволяет разделять зависимости на группы. Например на основную группу, зависимости из которой будут использоваться как при разработке, тестировании проекта, так и при сборке продакшн версии приложения. А в отдельную группу `dev` можно выделить те зависимости, которые будут использоваться только в процессе разработки и тестирования проекта, но которые не нужны в продакшене. Эти зависимости могут быть: тестовые фреймворки, инструменты как например линтеры и языковые серверы, и тд.

Для этого конкретного проекта определены следующие зависимости в файле `pyproject.toml`:

```
[tool.poetry.dependencies]
python = "3.9.18"
djangoestframework = "3.14.0"
Django = "3.2.16"
djoser = "2.1.0"
```



```
drf-spectacular = "0.26.5"
drf-spectacular-sidecar = "*"
drf-yasg = "1.21.5"
django-filter = "22.1"
django-cors-headers = "3.14.0"
unicorn = "20.1.0"
python-dotenv = "0.21.0"
geoip2 = "2.9.0"
pytz = "2023.3.post1"
requests = "2.28.1"
Pillow = "9.3.0"

[tool.poetry.group.dev.dependencies]
black = "*"
Flake8-pyproject = "*"
django-extensions = "*"
Werkzeug = "*"
pyOpenSSL = "*"
django-stubs = "*"
djangoRESTframework-stubs = "*"
```

Как мы видим, зависимости разделены на две группы. В первой группе **main** находятся обязательные основные зависимости. Во второй группе **dev** находятся необязательные зависимости используемые только в ходе разработки проекта.

Установка дополнительных инструментов

Установка языкового сервера и статического анализатора кода

В современном мире написание кода не представляется без использования статических анализаторов кода и языковых серверов.

Статический анализатор кода это программа которая парсит кодовую базу проекта и определяет ошибки типов, синтаксиса, логики в коде. Она показывает пользователю отчет с диагностическими данными и указывает как эти несоответствия можно исправить.

Изначально статические анализаторы кода представляли собой cli инструмент который применяется из командной строки. Но с развитием инструментов, и со стремлением компании Microsoft продвигать свой редактор VS Code, этой компанией был разработан language server protocol. Этот протокол устанавливает стандартный способ взаимодействия между инструментами командной строки либо редакторами кода и языковым сервером.

Языковой сервер это программа которая обрабатывает код внутри файла или во всей кодовой базе и статически анализирует его. Клиентами же могут выступать редакторы кода инструменты командной строки и так далее.

Так language server protocol это opensource технология, большинство современных редакторов кода поддерживают интеграцию с языковыми серверами.

Для python существует несколько языковых серверов. Одним из наиболее популярных является языковой сервер pyright разработанный компанией Microsoft.

Pyright можно выполнить несколькими способами. Установить локально в проект, установить глобально на систему, установить внутри редактора кода, где он будет доступен при редактировании.

В этом проекте я добавил его как development зависимость в файл pyproject.toml. Это позволит использовать его при необходимости из командной строки. Также это послужит своего рода документацией для других разработчиков о том, какие инструменты используются в проекте.

Текстовый редактор Neovim поддерживает протокол языкового сервера и может использовать языковой сервер pyright при работе для предоставления автодополнения, подсвечивания ошибок в коде, статического анализа и его визуализации внутри редактора, перемещения внутри файла и кодовой базы.

Для настройки статического анализатора кода используется файл pyproject.toml. Для этого нужно перечислить желаемые настройки в разделе `[tool.pyright]`.

Код настройки pyright

```
[tool.pyright]
include = []
exclude = [".pytest_cache",
            "**/__pycache__",
]
pythonVersion = "3.9.18"
typeCheckingMode = "standard"
```

Эти настройки означают:

- `include` - файлы или директории которые мы хотим явно включить в путь в котором работает pyright.
- `exclude` - файлы или директории которые мы хотим исключить пути. Это делается для улучшения скорости работы инструмента.
- `pythonVersion` - версия питон. В зависимости от версии python pyright будет указывать на устаревшие либо пока еще не используемые в данной версии питон синтаксические конструкции.
- `typeCheckingMode` - строгость к которой pyright будет подходить к анализу кода.

Установка линтера

Линтер это программа которая проверяет соответствие написанного кода определенным правилам форматирования текста. Величина отступов, количество пустых строк между параграфами и так далее.

Как и в случае с языковыми серверами существует несколько популярных линтеров для python. Некоторые из них это:

- flake8

- pylint
- ruff

Для данного проекта выбран линтер flake8.

flake8 проверяет код на соответствие стандарту pep8.

flake8 можно установить несколькими способами. Установить локально в проект, установить глобально на систему, установить внутри редактора кода, где он будет доступен при редактировании.

В этом проекте я добавил его как development зависимость в файл pyproject.toml. Это позволит использовать его при необходимости из командной строки. Также это послужит своего рода документацией для других разработчиков о том, какие инструменты используются в проекте.

Для настройки линтера кода используется файл pyproject.toml. Для этого нужно перечислить желаемые настройки в разделе `[tool.flake8]`.

Код настройки flake8

```
[tool.flake8]
max-line-length = 88
extend-ignore = ["E203", "I001", "I005", "R504"]
exclude = [
    ".git",
    "__pycache__",
    "env",
    "migrations",
    "settings.py",
    "venv",
    "management"
]
max-complexity = 10
```

max-line-length - Устанавливает максимальную длину, которую может иметь любая строка (за некоторыми исключениями). extend-ignore - Указывает список кодов для добавления в список игнорируемых. exclude - Укажите список глобальных паттернов, разделенных запятыми, чтобы исключить их из проверок. max-complexity - Устанавливает максимально допустимое значение сложности McCabe для блока кода.

Установка форматера

Если линтер только проверяет код на соответствие стилю, то формater автоматически редактирует код так чтобы он соответствовал выбранным правилам. Наиболее популярными форматерами для python являются:

- Black
- Ruff

Для данного проекта был выбран форматтер black. Он позволяет форматировать код согласно установленным правилам, либо согласно правилам выбранным разработчиками форматтера как

значения по умолчанию.

Разработчики называют black - optionated. Это означает что ряд правил которые установлены для него как правил форматирования по умолчанию не соответствуют стандартам pep8. К примеру pep8 определяет максимальную ширину строки в 79 знаков. Когда как black по умолчанию устанавливает максимальную ширину строки в 88 знаков.

black можно установить несколькими способами. Установить локально в проект, установить глобально на систему, установить внутри редактора кода, где он будет доступен при редактировании.

В этом проекте я добавил его как development зависимость в файл pyproject.toml. Это позволит использовать его при необходимости из командной строки. Также это послужит своего рода документацией для других разработчиков о том, какие инструменты используются в проекте.

Для настройки форматера кода используется файл pyproject.toml. Для этого нужно перечислить желаемые настройки в разделе `[tool.black]`.

Код настройки black

```
[tool.black]
line-length = 88
```

Для black явно указываем длину строки. Остальные правила используем по умолчанию.

Работа над проектом

Структура проекта Django

Если вы впервые используете Django, вам придется позаботиться о начальной настройке. В частности, вам нужно будет автоматически сгенерировать код, который создаст проект Django - набор настроек для экземпляра Django, включая конфигурацию базы данных, специфические для Django опции и настройки для приложения.

В командной строке перейдите в каталог, где будет храниться ваш код, и выполните следующую команду:

```
django-admin startproject mysite
```

Это создаст каталог mysite в вашем текущем каталоге.

Давайте посмотрим, что создал startproject:

```
mysite/
  manage.py
  mysite/
    __init__.py
    settings.py
```

```
urls.py
asgi.py
wsgi.py
```

Эти файлы:

- Внешний корневой каталог `mysite/` - это контейнер для вашего проекта. Его имя не имеет значения для Django; вы можете переименовать его во что угодно.
- `manage.py`: Утилита командной строки, которая позволяет вам взаимодействовать с проектом Django различными способами. Все подробности о `manage.py` вы можете прочитать в разделах `django-admin` и `manage.py`.
- Внутренний каталог `mysite/` - это собственно Python-пакет для вашего проекта. Его имя - это имя пакета Python, которое вам нужно будет использовать для импорта всего, что находится внутри него (например, `mysite.urls`).
- `mysite/init.py`: Пустой файл, который сообщает Python, что эта директория должна считаться пакетом Python. Если вы новичок в Python, прочитайте больше о пакетах в официальной документации Python.
- `mysite/settings.py`: Настройки/конфигурация для этого проекта Django. Django settings расскажет вам все о том, как работают настройки.
- `mysite/urls.py`: Декларации URL для этого Django-проекта; "оглавление" вашего сайта на базе Django. Подробнее об URL-адресах вы можете прочитать в разделе "Диспетчер URL".
- `mysite/asgi.py`: Точка входа для ASGI-совместимых веб-серверов для обслуживания вашего проекта. Подробнее см. в разделе Как разворачивать с помощью ASGI.
- `mysite/wsgi.py`: Точка входа для WSGI-совместимых веб-серверов для обслуживания вашего проекта. Подробнее см. в разделе Как развернуть с помощью WSGI.

Теперь, когда ваше окружение - "проект" - создано, вы можете приступить к работе.

Каждое приложение, которое вы пишете в Django, состоит из пакета Python, который следует определенным соглашениям. Django поставляется с утилитой, которая автоматически генерирует базовую структуру каталогов приложения, так что вы можете сосредоточиться на написании кода, а не на создании каталогов.

В чем разница между проектом и приложением? Приложение - это веб-приложение, которое что-то делает - например, система блогов, база данных публичных записей или небольшое приложение для опросов. Проект - это набор конфигураций и приложений для определенного веб-сайта. Проект может содержать несколько приложений. Приложение может находиться в нескольких проектах.

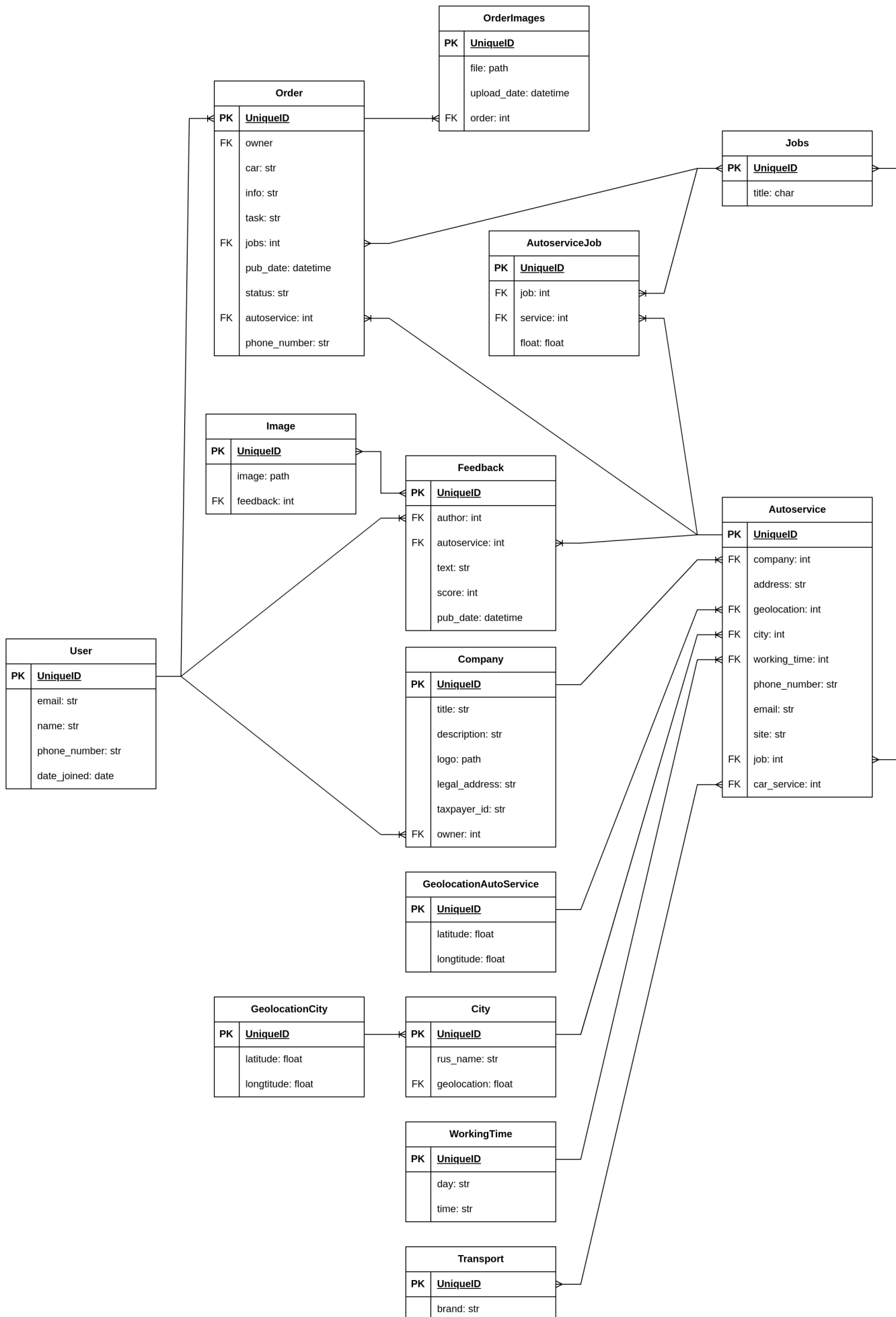
Ваши приложения могут находиться в любом месте вашего пути к Python. В этом уроке мы создадим наше приложение `poll` в том же каталоге, что и файл `manage.py`, чтобы его можно было импортировать как собственный модуль верхнего уровня, а не как подмодуль `mysite`.

Чтобы создать приложение, убедитесь, что вы находитесь в той же директории, что и файл `manage.py`, и введите эту команду:

```
python manage.py startapp polls
```

Структура данных проекта

Данные которыми будет оперировать приложение представляют собой следующую структуру:



Базовые модели проекта

Транспорт

Модель для выбора марки автомобиля при фильтрации поиска и указания конкретной специализации автосервиса.

Поля модели:

```
brand
```

Виды работ

Модель хранит список производимых автосервисами работ, с указанием стоимости определенного вида работы для конкретного автосервиса. Также позволяет производить фильтрацию по полю при выполнении поиска. Связана с моделью автосервиса через AutoserviceJob.

Поля модели:

```
title  
description
```

Компания

Хранит информацию о компании владеющей сетью автосервисов. Представляет собой не редактируемый личный кабинет компании. В дальнейшем представляет возможность сделать полноценный личный кабинет для владельца.

Поля модели:

```
title  
description  
logo  
legal_address
```

Автосервис

Хранит информацию об автосервисе.

Поля модели:

```
company  
address  
geolocation  
city  
working_time
```



```
phone_number
email
site
job
car_service
```

WorkingTime

Модель позволяет создать расписание работы автосервиса в разные дни недели Отзыв

Модель позволяет создать отзыв на автосервис. Находится в доработке. На текущий момент любой авторизованный пользователь может оставить отзыв на любой сервис. В будущем будет добавлен функционал позволяющий оставить отзыв только после оформления пользователем заявки в выбранный автосервис.

Поля модели:

```
author
autoservice
text
score
pub_date
```

Пользователь (нужна помощь с реализацией)

Модель кастомного пользователя. Для работы с пользователями используются возможности библиотеки Djoser. Для пользователя предусмотрена авторизация по электронной почте, номеру телефона или никнейму, для этого используется класс AuthBackend. В разработке находится процесс регистрации пользователя по телефону или электронной почте. На текущий момент возникают сложности с сохранением пароля. Предположительное решение проблемы - добавление кастомного сериализатора для регистрации.

Поля модели:

- email - Поле для хранения информации о электронной почте пользователя
- name - Поле для хранения информации о имени пользователя
- phone_number - Поле для хранения информации о телефоне пользователя
- date_joined - Поле для хранения информации о дате регистрации пользователя username email last_name first_name last_name phone_number date_joined image

Описание API проекта.

В данный момент структура Django-приложения состоит из приложений для соответствующих моделей и приложение api, в котором находятся все файлы urls.py, serializers.py, views.py. Данная структура позволяет быстро создать новую версию api, но нагромождает проект директориями. Нужна помощь в подходе к выбору структуры Django-приложения.

Приложение является API сервером. Все возможности предоставляются при помощи создания точек API к которым фронтенд приложение обращается с запросами.

Файлы `urls.py`, `serializers.py`, `views.py` расположены в директории `api/v1/`. Такое размещение файлов позволяет контролировать их изменения и упростит развитие API в дальнейшем. При появлении потребности серьезно изменить функционал API, возможно будет создать отдельную директорию `v2` и заботиться о новой версии. Приложение является API сервером. Все возможности предоставляются при помощи создания точек API к которым фронтенд приложение обращается с запросами.

Файлы `urls.py`, `serializers.py`, `views.py` расположены в директории `api/v1`. Такое размещение файлов позволяет контролировать их изменения и упростит развитие API в дальнейшем. При появлении потребности серьезно изменить функционал API возможно будет создать новую директорию `v2` и разрабатывать новую версию API не опасаясь за работоспособность версии API которая используется в продакшене.

Работоспособность API обеспечивают файлы различных типов:

- `models.py` - Здесь содержатся модели данных для связи с таблицами базы данных
- `views.py` - Здесь содержатся функции отвечающие за обработку, представление данных полученных из базы данных либо из запроса.
- `serializers.py` - Здесь содержатся сериализаторы данных, отвечающие за сериализацию, десериализацию и валидацию данных.
- `urls.py` - Здесь содержатся рутинги для связи view функций конкретными путями которым они соответствуют.

Фреймворк аутентификации и работа с пользователем

Django поставляется с системой аутентификации пользователей. Она работает с учетными записями пользователей, группами, разрешениями и пользовательскими сессиями на основе cookie. В этом разделе документации рассказывается о том, как работает стандартная реализация из коробки, а также о том, как расширить и настроить ее под нужды вашего проекта. Обзор

Система аутентификации Django работает как с аутентификацией, так и с авторизацией. Вкратце, аутентификация проверяет, является ли пользователь тем, за кого он себя выдает, а авторизация определяет, что разрешено делать аутентифицированному пользователю. Здесь термин аутентификация используется для обозначения обеих задач.

Система аутентификации состоит из:

- Пользователи .
- Разрешения: Двоичные (да/нет) флаги, обозначающие, может ли пользователь выполнять определенную задачу.
- Группы: Общий способ применения меток и разрешений к нескольким пользователям.
- Настраиваемая система хеширования паролей
- Формы и инструменты просмотра для регистрации пользователей или ограничения содержимого.
- Подключаемая бэкэнд-система

Система аутентификации в Django стремится быть очень общей и не предоставляет некоторых возможностей, обычно встречающихся в веб-системах аутентификации. Решения некоторых из этих общих проблем были реализованы в сторонних пакетах:

- Проверка надежности пароля
- Ограничение попыток входа в систему
- Аутентификация от третьих лиц (например, OAuth)
- Разрешения на уровне объектов

Для упрощения работы с представлениями рутингами и системой аутентификации пользователя в проекте используется библиотека Djoser.

REST-реализация системы аутентификации Django. Библиотека djoser предоставляет набор представлений Django Rest Framework для обработки основных действий, таких как регистрация, вход, выход, сброс пароля и активация аккаунта. Она работает с пользовательской моделью пользователя.

Вместо того чтобы повторно использовать код Django (например, PasswordResetForm), мы переделали несколько вещей, чтобы они лучше вписывались в архитектуру Single Page App.

Приложения проекта

Теперь, когда ваше окружение - "проект" - создано, вы можете приступить к работе.

Каждое приложение, которое вы пишете в Django, состоит из пакета Python, который следует определенному соглашению. Django поставляется с утилитой, которая автоматически генерирует базовую структуру каталогов приложения, так что вы можете сосредоточиться на написании кода, а не на создании каталогов.

Проекты против приложений

В чем разница между проектом и приложением? Приложение - это веб-приложение, которое что-то делает - например, система блогов, база данных публичных записей или небольшое приложение для опросов. Проект - это набор конфигураций и приложений для определенного веб-сайта. Проект может содержать несколько приложений. Приложение может находиться в нескольких проектах.

Ваши приложения могут находиться в любом месте на пути Python. В этом уроке мы создадим наше приложение poll в той же директории, что и файл manage.py, чтобы его можно было импортировать как собственный модуль верхнего уровня, а не как подмодульmysite.

Для проекта создано несколько приложений которые определяют его структуру:

- autoservice - приложение для работы с данными автосервиса
- users - приложение для работы с данными пользователей
- orders - приложение для работы с заявками пользователей автосервисам

Тестирование и отладка проекта

Тестовый фреймворк django

Автоматизированное тестирование - чрезвычайно полезный инструмент для устранения ошибок для современного веб-разработчика. Вы можете использовать набор тестов - набор тестов - для решения или предотвращения ряда проблем:

При написании нового кода вы можете использовать тесты для проверки того, что ваш код работает так, как ожидалось. При рефакторинге или модификации старого кода вы можете использовать тесты, чтобы убедиться, что внесенные изменения не повлияли на поведение приложения неожиданным образом.

Тестирование веб-приложения - сложная задача, поскольку веб-приложение состоит из нескольких слоев логики - от обработки запросов на уровне HTTP, проверки и обработки форм до рендеринга шаблонов. С помощью фреймворка Django для тестирования и различных утилит вы можете имитировать запросы, вставлять тестовые данные, проверять результаты работы приложения и вообще проверять, что ваш код делает то, что должен делать.

Предпочтительным способом написания тестов в Django является использование модуля unittest, встроенного в стандартную библиотеку Python.

Тестирование API

Docker контейнер для тестирования API

Для удобства тестирования и ускорения итерации в процессе разработки приложение запускается в Docker контейнере.

Использование Postman для тестирования API

Для тестирования API проекта я использовал приложение Postman.

Развертывание проекта на сервере

Docker контейнер для запуска приложения на сервере

Запуск приложения на сервере

Результаты работы над проектом