Здравствуйте, уважаемые члены жюри. Я – Вершинин Сергей и Мой коллега – Воробьев Сергей, учащиеся 11 лицейского класса ГБОУ школы №924 хотим представить Вам наш проект SOLVE-GIA. В данный момент он находится в процессе разработки.

В наши дни сайты для подготовки к экзаменам становятся все более популярными. Они предлагают удобный формат подготовки в любом месте и в любое время. Однако, задания в варианте не имеют единой медианы сложности, что создает проблемы для учителей и их учеников. Мы опросили учителей нашей школы и выяснили, что проблема с вариантами разной сложности и заданиями в варианте с разной сложностью, действительно существует. Для решения данной проблемы мы разработаем собственное ядро веб-приложения, а также Telegram бота с расширенным функционалом. Для этого нам нужно изучить большой объем информации, разработать формулу и логику расчета сложности каждого задания в отдельности, а также продумать каркас и дополнительные возможности веб-приложения.

Проблемы, которые удалось выделить:

1 Неравномерность сложности заданий на сайтах для подготовки к экзаменам.

2 ограниченный функционал Telegram бота или его отсутствие.

3 Плохая отказоустойчивость из-за отсутствия брокера сообщений.

Цель нашего проекта – создание собственного ядра веб-приложения и телеграмм бота с расширенным функционалом.

Задачи проекта:

1 Поиск и анализ новой информации по теме.

2 Разработка формулы сложности 3 Создание сайта и телеграмм бота 4 тестирование и завершение проекта

Разработку нашего проекта мы решили начать с поиска информации и анализа аналогов, после которого мы определили задачи, проблемы а также начали работу над backend’om веб-приложением. Основным языком программирования мы выбрали Python, потому что это не только один из самых популярных языков программирования, но и имеет большое количество фреймворком и отличное сообщество. В качестве фреймворка для создания backend’a сайта для начала выбрали FLASK, но после некоторого времени разработки мы перешли на Django, потому что он имеет больший функционал, а также встроенные функции, которые на FLASK пришлось бы реализовывать самим. Для разработки многофункционального Бота в Telegram мы решили использовать Aiogram, тк он имеет большое количество встроенный функций, таких FINITIAL STATE MACHINE в отличие от некоторых аналогов, а также более удобная структура кода.

А теперь перейдем непосредственно к архитектуре проекта.

Для начала я бы хотел ввести некоторые понятия нашей терминологии.

* Брокер сообщений — это программа для связи между приложениями, системами и службами, помогающее им обмениваться информацией друг с другом.
* Машина – арендуемый сервер для разворачивания сервиса.
* Монолит – некоторое количество сервисов, связанных в единое взаимозависящее целое.
* Горизонтальное масштабирование - разрешение проблемы высоконагруженности компонента, посредством размещения копий данного компонента на дополнительных машинах.
* Компонент-посредник – программа, обеспечивающая общение между Kafka и сервисом.
* Сообщение – информация в формате json.
* Фатальная зависимость – зависимость одного компонента от другого, при которой, ошибка, возникшая в первом сервисе, вызовет ошибку и во втором сервисе.

В нашем варианте архитектуры ни один сервис не должен общаться напрямую с другим сервисом, во избежание построения монолита, где каждый компонент имеет непосредственную зависимость от остальных компонентов. Каждый сервис будет обращаться к компоненту-посреднику (АКА локальному брокеру) при помощи http-запросов на локальном порте. Компонент-посредник будет обрабатывать запрос, отправлять его брокеру сообщений. Брокер сообщений, в свою очередь, будет хранить и отдавать сообщения по запросам остальных сервисов. Таким образом, Kafka будет обеспечивать защиту от взаимозависимостей между сервисами, равномерное распределение нагрузки, а также, хранение сообщений, на случай непредвиденного падения одного из сервисов, с целью их дальнейшей обработки после восстановления работоспособности. Суммируя все вышесказанное, брокер сообщений даёт: отказоустойчивость, отсутствие фатальных зависимостей между сервисами, возможность горизонтального масштабирования. На этом часть с архитектурой закончена, перейдем непосредственно к ядру нашего приложения.

Модели – бизнес-сущности, от которых зависит поведение, смысл и работа бизнес-логики.

\*Показать связи моделей\*