Программное обеспечение «Moscow digital school — платформа развития компетенций для цифровой экономики» (ПО)

Общее описание функциональных характеристик программного обеспечения и информации, необходимой для установки и эксплуатации программного обеспечения

1. Концепция

ПО представляет собой цифровой B2C-сервис, позволяющий проходить обучение онлайн. Представляет из себя систему из личных кабинетов для педагога, преподавателя, куратора и администратора. Сервис пользуется спросом у юристов из разнообразных сфер: от банковского сектора до игровой индустрии. Система решает комплекс задач по автоматизации процесса обучения:

- организация образовательного процесса
- онлайн-оплата доступа к курсу
- автоматический расчет прогресса и проверка знаний

Результатом обучения является сертификат, который автоматически отправляется на почту пользователя, согласно бизнес-процессу.

Роли пользователей

Таблица 3. Роли пользователей

Роль	**Описание доступных действий и полномочий**
Администратор	Управляет пользователями, системными настройками, имеет полный доступ к веб-интерфейсу системы
Куратор	Управляет пользователями в рамках назначенных ему учебных групп, оказывает поддержку слушателям, настраивает доступ к учебным материалам
Педагог	Организует обучение, проверку заданий, проводит онлайн-уроки

2. Решения по архитектуре

Архитектура системы монолитная. Система строится на основе программного обеспечения для автоматизации развертывания и управления в средах с поддержкой контейнеризации Docker, с оркестровкой и управлением, разворачиваемая на базе OS Linux

2.1.1. Описание компонентов и нейронных сетей

N	**сервис продукта**	**Описание**
1	API & business-logic	Единое API, которое организует взаимодействие сервисов внутри системы
2	PostgreSQL	База данных хранит, например, пользователей, временно - результаты распознавания, промежуточный / финальный статус результатов распознавания, настройки для пресетов, метрики.
3	Redis	Хранит кэш заданий - при повторной отправке файла на распознавание ответ возвращается из кэша, чтобы не создавать задание на распознавание.

2.1.2. Описание взаимодействия

Взаимодействие внутренних систем осуществляется через API. Протокол взаимодействия с внешними системами-источниками – REST.

Схема процесса распознавания:

- Логин и пароль от базы данных хранится в secrets и передаются в сервис через переменные окружения. При подключении к базе данных, сервис, передает логин и пароль в базу данных.
- Логин и пароль, аналогичным образом размещаются в secrets и передаются в сервис через переменные окружения.

2.1.2.1. Описание Endpoints системы

Система для взаимодействия с внешними системами-источниками использует API — представляет из себя сервис, который предоставляет для взаимодействия набор интерфейсов RESTful API.

#	Сообщение	Тип запроса/ответа	Формат запроса/ответа	endpoint
1	Запрос на получение авторизационного токена	Синхронный	json	/access-tokens
1.1	Ответ с токеном	Синхронный	json	
1.2	Ответ с ошибкой авторизации	Синхронный	json	
2	Запрос на создание задания обработки данных	Синхронный	json	/tasks
2.1	Ответ с id созданного задания	Синхронный	json	
2.2	Ответ с ошибкой “Файл слишком большой”	Синхронный	html	
2.3	Ответ с ошибкой валидности токена	Синхронный	json	
2.4	Ответ с ошибкой ID задачи	Синхронный	json	
2.5	Ответ с ошибкой валидации файла	Синхронный	json	
3	Ответ с результатом обработки файла	Асинхронный / webhook	json	
4	Запрос на получение результата обработки	Асинхронный	json	/tasks/{{taskId}}

1. Запрос на получение авторотационного токена
POST {url}/access-tokens

Параметр	Кол-во	Тип данных	Размещение	Описание
${url}	1	url	request line	URL стенда, на который осуществляется запрос
Content-Type	1	string	Header	application/json
email	1	string	body/raw	Адрес электронной почты, предоставленный для авторизации
password	1	string	body/raw	Пароль, предоставленный для авторизации

Ответ с токеном
HTTP-code: 200

Параметр	Кол-во	Тип данных	Размещение	Описание
token	1	string	body	Токен авторизации, с которым необходимо выполнять все следующие запросы Срок жизни токена 24 часа с момента получения

Пример ответа:

{ "token":
"eyJhbGciOiJIUzUxMiJ9.*.s4fom5lFUg7KNbz7TcytQIrVF1Olm9
zncCeXDXZUrtQU-RxMSA" }

Ответ с ошибкой авторизации
HTTP-code: 401 / 403

Параметр	Кол-во	Тип данных	Размещение	Описание
timestamp	1	string	body	время события на сервере
status	1	int	body	код ошибки
error	1	string	body	причина ошибки
message	1	string	body	текст сообщения об ошибке
path	1	string	body	путь запроса

Пример ответа:

```
{
   "timestamp": "2020-10-23T10:18:13.071+0000",
   "status": 401,
   "error": "Unauthorized",
   "message": "Unauthorized",
   "path": "/access-tokens"
}
```

2. Запрос на создание задания обработки данных

Параметр	**Кол-во**	**Тип данных**	**Размещение**	**Описание**
${url}	1	string	request line	URL стенда, на который осуществляется запрос
authorization	1	string	header	содержит авторизационный токен
files	1..40	object	body/form-data	объекты с файлами для распознавания
preset	1	string	body/form-data	выбранный пресета, для распознавания переданных файлов
webhookUrl	0..1	string	body/form-data	адрес вебхука для ответа
confidence				
priority				

Ответ с id созданного задания
HTTP-code: 201

Параметр	Кол-во	Тип данных	Размещение	Описание
id	1	string	body	Внешний ID задания
creatorId	1	string	body	ID автора задания
status	да	string	body	Completed - распознавание завершено Pending - распознавание в процессе Rejected - ошибка распознавания
solution	0..1	object	body	Не используется
filesUploaded	1	array	body	
preset	1	string	body	Шаблон работы с файлом
webhookUrl	0..1	string	body	Адрес для вебхука, на который будет возвращен ответ после распознавания (если не используется поллинг результата распознавания, см.п. 1.3.)
sessionId	0..1	string	body	Не используется. Допустимое значение: null
filesDetailed	0..1	array	body	Не используется. Допустимое значение: null или []
files	0..1	array	body	Не используется. Допустимое значение: null или []

Пример ответа


```
{
   "id": "cac0e745-153d-11eb-9dda-3a0d5824f1c0",
   "creatorId": "20c58eea-4475-4b9f-b847-315dfdb99048",
   "documentType": "passport",
   "status": "Pending",
   "solution": {},
   "filesUploaded": [
      {
         "path": "/tasks/cac0e745-153d-11eb-9dda-3a0d5824f1c0/ddf53d4c-476f-4ca9-b25c-563ccc3695b4.jpg",
         "downloadUrl": "https://357715.selcdn.ru/basis-doc-dev/tasks/cac0e745-153d-11eb-9dda-
3a0d5824f1c0/ddf53d4c-476f-4ca9-b25c-563ccc3695b4.jpg",
         "size": 173089,
         "originalFileName": "DL_SNILS.jpg",
         "checksum": "2502e82e7ec99d9894516aaee55127fb02e045805fa77ed7fab30a2c8a3f5038",
         "id": null
      }
   ],
   "preset": "passport.recognize-45",
   "webhookUrl": "https://webhook.site/8cac5464-373d-44cf-ac3b-79a5f7ecb5aa",
   "sessionId": null,
   "filesDetailed": [],
   "files": []
}
```


Ответ с ошибкой “Файл слишком большой”

HTTP-code: 413. Возвращается html-объект

Пример ответа

```
<html>
<head><title>413 Request Entity Too Large</title></head>
<body>
<center><h1>413 Request Entity Too Large</h1></center>
<hr><center>nginx/1.17.10</center>
</body>
</html>
```

Ответ с ошибкой валидности токена

HTTP-code: 401 / 403

Параметр	Кол-во	Тип данных	Размещение	Описание
error	1	object	body/form-data/	
code	1	int	body/form-data/error	код ошибки
message	1	string	body/form-data/error	сообщение об ошибке
details	1	object	body/form-data/error	детали ошибки
token	1	string	body/form-data/error/details	токен сессии

Пример ответа


```
{
"error": {
  "code": 401,
  "message": "Invalid token",
  "details": {
"token": "eyJhbGciOiJIUzUxMiJ9.*.s4fom5lFUg7KNbz7TcytQlrVF1OIm9zncCeXDXZUrtQU-RxMSA"
}
}
}
```


2.1.4. Технологический стек

JavaScript

Мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

Node.js

Программная платформа, основанная на движке V8 (компилирующем JavaScript в машинный код), превращающая JavaScript из узкоспециализированного языка в язык общего назначения. Node.js

добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API, написанный на C++, подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода.

PostgreSQL

Это популярная свободная объектно-реляционная система управления базами данных. PostgreSQL базируется на языке SQL и поддерживает многочисленные возможности.

Используется для хранения данных.

NestJS

Фреймворк для построения эффективных, масштабируемых Node.js. (opens new window) приложений на сервере. Он использует современный JavaScript, построен на основе и полностью поддерживает TypeScript.

VueJS

Прогрессивный фреймворк для создания пользовательских интерфейсов. В отличие от фреймворков-монолитов, Vue создан пригодным для постепенного внедрения. Его ядро в первую очередь решает задачи уровня представления (view), что упрощает интеграцию с другими библиотеками и существующими проектами.

Redis (стандартная версия, не Enterprise)

Хранит кэш заданий. При повторной отправке файла на распознавание ответ возвращается из кэша, чтобы не создавать задание на распознавание.

3. Реализация эксплуатационных требований

3.1. Спецификация решения по реализации отказоустойчивости

Отказоустойчивость сервиса обеспечивается разворачиванием нескольких экземпляров продукта на виртуальных машинах контура. Управление разворачиванием и поддержкой экземпляров в рабочем состоянии осуществляется стандартными средствами кластера.

Отказоустойчивость базы данных обеспечивается кластерным решением с классическим методом перехода на резерв в ручном режиме.

3.2. Реализация требований к доступности

Требования к доступности реализуются с помощью следующих методов:

Проверка работоспособности балансируемых серверов балансировщиком нагрузки. Балансировщик осуществляет проверку работоспособности (мониторинг) серверов кластера

Проверка работоспособности сервисов. Проверка готовности проверяет, готов ли контейнер обрабатывать запросы. Ошибка проверки готовности означает, что контейнер не должен получать трафик от прокси, даже если он работает. Проверка живучести проверяет, работает ли контейнер. Балансировщик осуществляет проверку работоспособности (мониторинг) контейнера. При проверке на живучесть и проверку на работоспособность сервиса, в случае, когда http-ответ возвращается не с кодом 200, сервер будет непрерывно стараться перезапустить экземпляр сервиса.

3.3. Реализация требований по резервному копированию, архивации и восстановлению

Резервное копирование на контуре разработки и контурах функционального тестирования не предполагается.

Резервное копирование данных для БД промышленного контура периодически по расписанию в автоматическом режиме. Выполнять резервное копирование необходимо ежедневно в 00:00 по московскому времени.

Раздел для бэкапа размещается на виртуальной машине backup, выделенная исключительно для бэкапа, и затем из него копироваться в СРК.

Кластер БД Postgresql состоит из трёх нод:

- Master-реплика, работающая на чтение и запись;
- Асинхронная slave-реплика – репликация по асинхронному протоколу и только чтение.
- Асинхронная backup-реплика – нода для бэкапирования базы данных.

3.4. Реализация требований к режиму сопровождения

Для промышленного контура: 24 часа 7 дней в неделю

Для контуров функционального тестирования: 8 часов 5 дней в неделю

3.5. Реализация требований к масштабированию

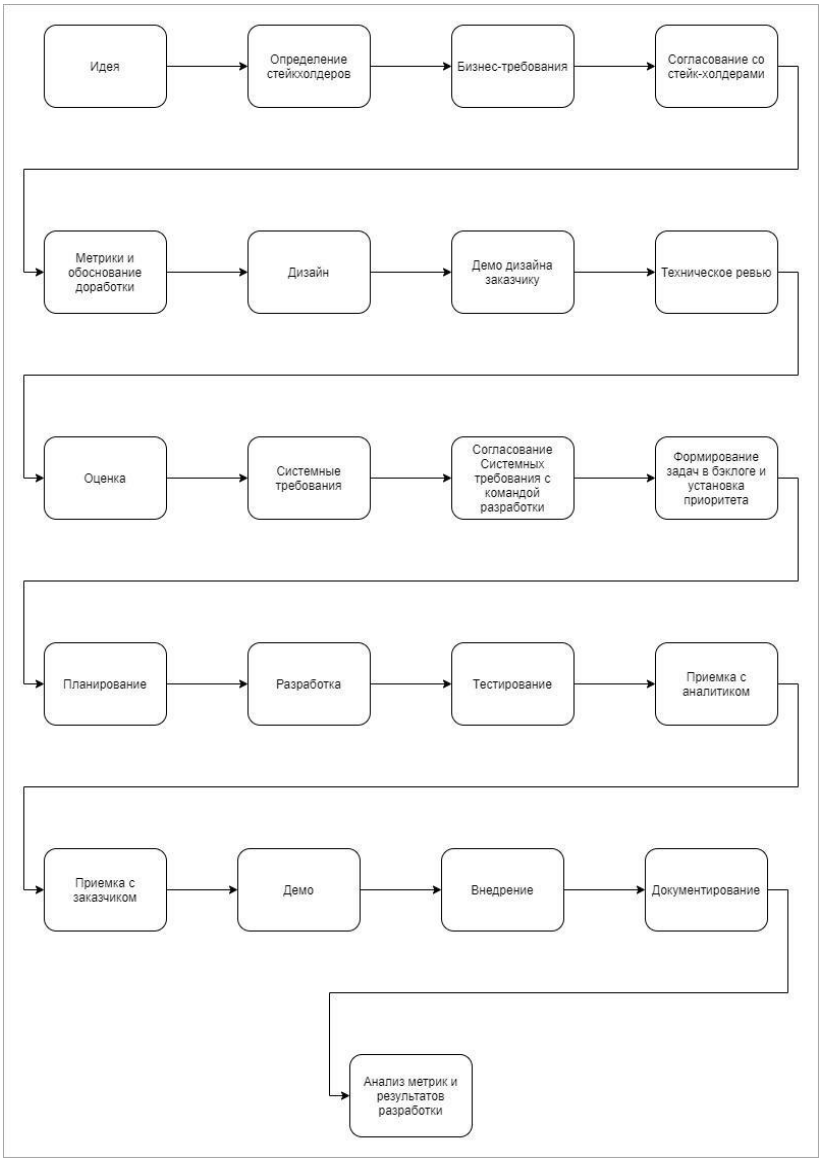
Масштабирование сервиса в кластере осуществляется горизонтально путем добавления дополнительных экземпляров сервисов в кластере без прерывания его функционирования.

Масштабирование базы данных PostgreSQL: Объем базы данных зависит от потока документов и сроков хранения результатов распознавания. В случае создания 1000 (тысячи) заданий на распознавание в день и хранения результатов распознавания, ежедневный прирост объема базы данных будет составлять 200 мегабайт. При этой исходный размер БД после развертывания составляет не более 100 мегабайт.

3.6. Реализация требований к катастрофоустойчивости

Не предъявляются.

4. Процесс разработки ПО



Верхнеуровнево у задачи следующий жизненный цикл: Идея - Бизнес-требования - Дизайн - Функциональные требования - Разработка - Тестирование и доработки - Приёмка - Внедрение - Документирование. Более детальный флоу указан на Рисунке 1;

Любая идея фиксируется в Story (или в другом типе задачи в зависимости от флоу работы на проекте);

Все задачи, над которыми работает аналитик, должны фиксироваться в Jira с указанием ссылки на прорабатываемый материал в Confluence. Статусы задач аналитика в Jira должны быть актуальными в конце каждой недели;

В процессе работы над задачей аналитик может проводить Техническое ревью, чтобы совместно с командой выработать подход к решению задачи; Бизнес-требования содержат указания на проблемы и потребности, схемы бизнес-процессов, которые необходимы в том числе для того, чтобы учесть все ветки процесса;

После формирования бизнес-требования задача передается на проработку дизайнеру, который занимается проектированием. При получении дизайна и его базового описания формируются функциональные требования;

Функциональные требования содержат конкретику - описание элементов дизайна, межсистемного взаимодействия и т.д.;

Все крупные артефакты (бизнес- и функциональные требования на крупный функционал) проходят обязательное ревью руководителем аналитики. На ревью может потребоваться время, которое необходимо учитывать в сроках выполнения задач. Также по результатам ревью задача может быть отправлена на доработку. Все замечания, при этом, указываются в виде комментариев к материалу, закрывает которые руководитель аналитики;

По результатам разработки Функциональных требований аналитик осуществляет “нарезку” задач на группы разработчиков. При этом все задачи должны быть перелинкованы со Story, а также должны быть указаны ссылки на Confluence в задачах Jira, а в материалах Confluence должны быть ссылки на задачи Jira (взаимная линковка облегчает “поиск концов”);

После выставления задач, аналитик может закрыть свою задачу на разработку требований;

В процессе разработки аналитик осуществляет консультирование разработчиков и, при необходимости, дорабатывает требования;

По результатам разработки аналитик участвует в приёмке задачи, а также документирует результаты разработки на уровне, оговоренном в команде.

5. Процесс сопровождения ПО

В результате успешного осуществления процесса менеджмента документации программных средств:

- разрабатывается стратегия идентификации документации, которая реализуется в течение жизненного цикла программного обеспечения или услуги;
- определяются стандарты, которые применяются при разработке программной документации;
- определяется документация, которая производится процессом или проектом; − указываются, рассматриваются и утверждаются содержание и цели всей документации;
- документация разрабатывается и делается доступной в соответствии с определенными стандартами;
- документация сопровождается в соответствии с определенными критериями.