Финальная работа

Тема: разработка облачного хранилища

Перед вами подробное описание вашего первого самостоятельного проекта, который вы сможете положить в портфолио.

Ниже вы найдёте технические подробности, которые помогут вам создать работающее приложение. Если у вас появятся вопросы или вы найдёте ошибку в техническом задании, обратитесь к куратору.

Описание проекта

Ваш университет разрабатывает бесплатное облачное хранилище файлов для студентов и преподавателей. Каждый студент при зачислении получает возможность зарегистрироваться и хранить в нём документы и фотографии, необходимые для учебного процесса. По сути, это аналог Google Drive или Яндекс Диска, но без рекламы и в инфраструктуре университета.

Приложение будет иметь клиент-серверную архитектуру с чётким разделением на frontend и backend. Вы будете разрабатывать backend-часть. Для взаимодействия с клиентом, который будет представлен React-приложением, вы разработаете REST API — пригодятся все знания, которые вы получили в курсе.

Проект достаточно объёмный. Чтобы не запутаться и подобрать подходящие инструменты, разобьём его на отдельные задачи и будем «есть слона по частям».

Задача 1. Подготовка инструментов

Цель задачи

В любом деле важна подготовка. Для эффективной разработки нужно подготовить окружение и выбрать удобный редактор кода.

Что нужно сделать

1. Установите среду разработки PhpStorm или Visual Studio Code, если они ещё не установлены на вашем компьютере. Можете использовать любой редактор кода, но в этих двух самый богатый функционал.
2. Установите XAMPP с официального сайта. В phpmyadmin создайте пустую базу данных cloud_storage.
3. Рекомендуем использовать версию PHP не ниже 7.4.
4. Вы будете разрабатывать REST API, поэтому полезно установить Postman. Эта утилита позволит легко производить отладку запросов в API.

Задача 2. Настройка виртуального сервера

Цель задачи

Настроить локальный сервер и убедиться в его работоспособности.

Что нужно сделать

1. Выберите каталог (папку), где будет расположено ваше приложение. Можно использовать каталог веб-сервера по умолчанию или выбрать свой. В Linux Ubuntu/Debian он по умолчанию находится в /var/www/html. Если используете Windows или Mac и Wamp или XAMPP соответственно, адрес каталога ищите в документации, он может меняться от версии к версии.
2. Конфигурационный файл вашего виртуального сервера должен выглядеть примерно так (используется каталог по умолчанию — **/var/www/html**):

```
<VirtualHost *:80>
    DocumentRoot /var/www/html
    ServerName cloud-storage.local
    ServerAlias www.cloud-storage.local
    ErrorLog "/var/log/apache2/cloud-storage.local-error.log"
      CustomLog "/var/log/apache2/cloud-storage.local-access.log"
common
    <Directory /var/www/html/>
        Options +Indexes +Includes +FollowSymLinks +MultiViews
        AllowOverride All
        Require all granted

        <IfModule mod_rewrite.c>
            Options -MultiViews
            RewriteEngine On
            RewriteCond %{REQUEST_FILENAME} !-f
            RewriteRule ^(.*)$ /index.php [QSA,L]
        </IfModule>
</Directory>
```

</VirtualHost>

3. После внесения правок в файл конфигурации перезапустите локальный сервер.
4. Создайте в каталоге /var/www/html (или в том, который указали) файл test.php с кодом:

```
<?php
    phpinfo();
```

   В строке адреса браузера напишите 127.0.0.1 и попробуйте загрузить страницу. Если всё настроено правильно, вы увидите страницу с информацией об установленной конфигурации PHP.

Задача 3. Реализация роутинга

Цель задачи

Реализовать роутинг, используя единую точку входа в приложение.

Что нужно сделать

1. В каталоге веб-сервера создайте файл index.php. Он будет единственной точкой входа в приложение.
2. Убедитесь, что активирован apache_mod_rewrite.
3. В файле index.php создайте ассоциативный массив, где в качестве ключей будут указываться относительные URL, например http://mysite.com/**funds/** (здесь **/funds/** — это относительный URL), а в качестве значений — другой вложенный массив с названиями HTTP-методов (например, GET) и вызовом функции класса соответствующего контроллера.
4. Проверьте все запросы, которые приходят в index.php, на соответствие массиву списка URL. Если такой URL присутствует в массиве, вызовите соответствующий метод.
5. Реализуйте загрузку классов контроллеров в виде автозагрузки (autoload).
6. По мере написания новых контроллеров содержимое массива $urlList будет дополняться.

Задача 4. Регистрация и авторизация пользователя

Цель задачи

Реализовать регистрацию, авторизацию пользователя и механизм сброса пароля.

Что нужно сделать

1. Создайте таблицу базы данных User, где будут храниться данные пользователя. Набор полей должен включать email и пароль в зашифрованном виде, а также роль: администратор или обычный пользователь. Остальные поля, например возраст и пол, можете добавить по желанию.

2. Создайте контроллер для сущности «пользователь» User.php. Этот контроллер будет обслуживать следующие API endpoints:

GET /users/list	Получить список пользователей (массив) с ограниченным количеством полей данных пользователя (выводить только общие данные: возраст, пол и так далее)
GET /users/get/{id}	Получить информацию о конкретном пользователе
PUT /users/update	Обновить профиль авторизованного пользователя (свой профиль)

   Реализуйте контроллер в виде класса и сделайте так, чтобы каждый эндпоинт обслуживался одноимённым методом.

3. В этот же контроллер добавьте методы login, logout и reset_password.

   Метод POST login принимает на вход email (логин) и пароль, производит проверку комбинации email и пароля в базе данных и в случае успеха создаёт новую сессию, идентификатор которой отправляется пользователю в виде cookie. Если логин и пароль неверны, отправляется ошибка.

   Метод GET logout завершает текущую сессию, а метод GET reset_password высылает ссылку на сброс пароля на указанный email пользователя.

Задача 5. Список пользователей

Цель задачи

Реализовать возможность управления списком пользователей для администратора.

Что нужно сделать

1. Если залогиненный пользователь имеет роль «администратор», то он может видеть список других пользователей на соответствующей странице.
2. Администратор может производить изменение и удаление пользователей.
3. Для этих действий реализуйте контроллер Admin, в котором опишите соответствующие методы, а затем зарегистрируйте их в массиве URL в index.php.
4. Обратите внимание на необходимость проверки роли пользователя на стороне backend.
5. Список эндпоинтов API:

GET /admin/users/list	Список пользователей
GET /admin/users/get/{id}	Информация по конкретному пользователю
DELETE /admin/users/delete/{id}	Удалить пользователя
PUT /admin/users/update/{id}	Обновить информацию пользователя

Задача 6. Мои файлы

Цель задачи

Реализовать возможность управления списком файлов пользователя.

Что нужно сделать

Реализуйте эндпоинты REST API, которые позволят выводить список файлов пользователя, получать ссылку на конкретный файл для скачивания, удалять файл, переименовывать файл, создавать подпапки и перемещать в них файлы.

Файловую систему можно реализовать двумя способами.

Первый способ — хранить структуру директорий (папок) и список файлов в базе данных, а в файловой системе операционной системы сохранять файлы в единственную папку, при этом шифруя имена таким образом, чтобы не было совпадений. Соответствие зашифрованных и реальных имён файлов при таком способе придётся хранить в таблице базы данных. При помощи той же базы данных можно хранить структуру папок и соответствие расположения файла определённой папке.

Второй способ — использовать файловую систему компьютера. В этом случае пригодятся функции PHP для работы с файловой системой.

Методы обработки файлов опишите в контроллере File.php.

GET /files/list	Вывести список файлов
GET /files/get/{id}	Получить информацию о конкретном файле
POST /files/add	Добавить файл
PUT /files/rename	Переименовать файл
DELETE /files/remove/{id}	Удалить файл
POST /directories/add	Добавить папку (директорию)
PUT /directories/rename	Переименовать папку
GET /directories/get/{id}	Получить информацию о папке (список файлов папки)
DELETE /directories/delete/{id}	Удалить папку

Задача 7. Возможность поделиться файлом

Цель задачи

Реализовать возможность делиться доступом к файлу с другим пользователем системы.

Что нужно сделать

Реализуйте API для возможности предоставить доступ к файлу определённому пользователю системы.

1. Опишите соответствующие методы в контроллере File.php.

2. Поиск пользователя производится по email. Для этого реализуйте эндпоинт /user/search/{email} в контроллере User.
3. В контроллере File.php реализуйте эндпоинты:

GET /files/share/{id}	Получить список пользователей, имеющих доступ к файлу
PUT /files/share/{id}/{user_id}	Добавить доступ к файлу пользователю с id user_id
DELETE /files/share/{id}/{user_id}	Прекратить доступ к файлу пользователю с id user_id

Критерии оценки

- Backend реализован в виде REST API.
- В REST API реализована авторизация с помощью токена.
- Frontend реализован в виде отдельного веб-приложения и подключается к backend с помощью REST API.
- В качестве базы данных используется MySQL или MariaDB.
- Параметры окружения, такие как параметры подключения к базе данных, вынесены в отдельный файл конфигурации.
- Ограничение на размер загружаемого файла — 2 Гб.
- В программе реализована система обработки ошибок, в частности обработка ошибок файловой системы.
- Код опубликован в Git-репозитории, через pull (merge) request.
- Если для корректной работы программы необходимы исходные данные в базе данных — реализован скрипт, который вносит эти данные на стадии развёртывания проекта.
- Проект запускается в среде XAMPP + PHP 7.4 и выше + MySQL 5.7 и выше.
- Соблюдены стандарты PSR-12.
- Методы классов имеют строгую типизацию и аннотации.
- Разработанная архитектура приложения расширяемая, понятная и прозрачная. Хороший пример — MVC (Model View Controller) в самом простом виде.