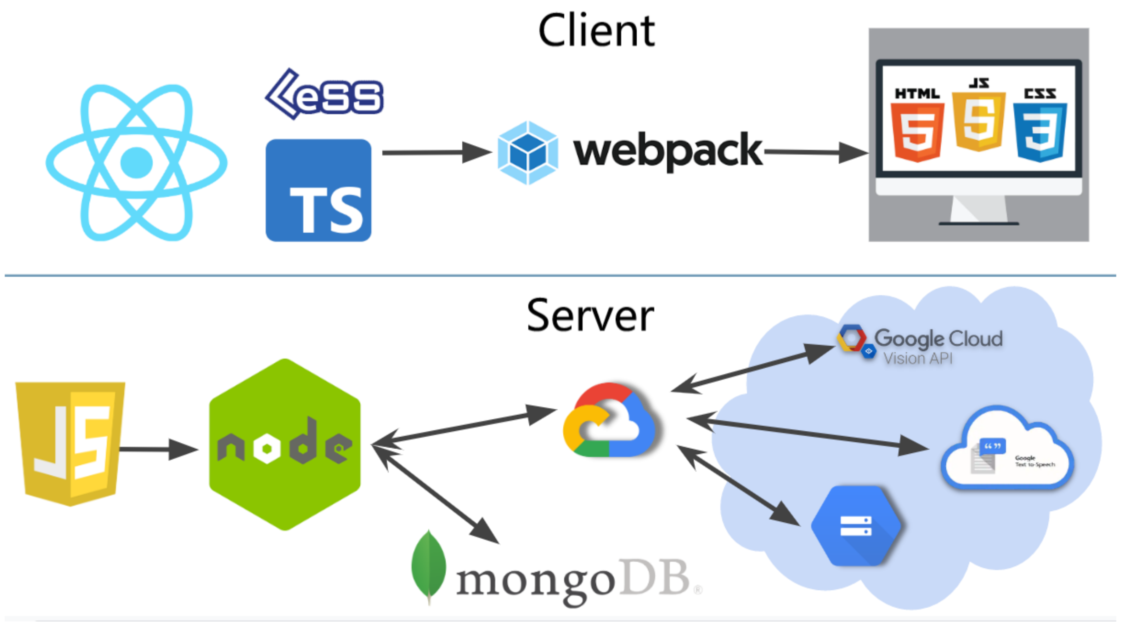
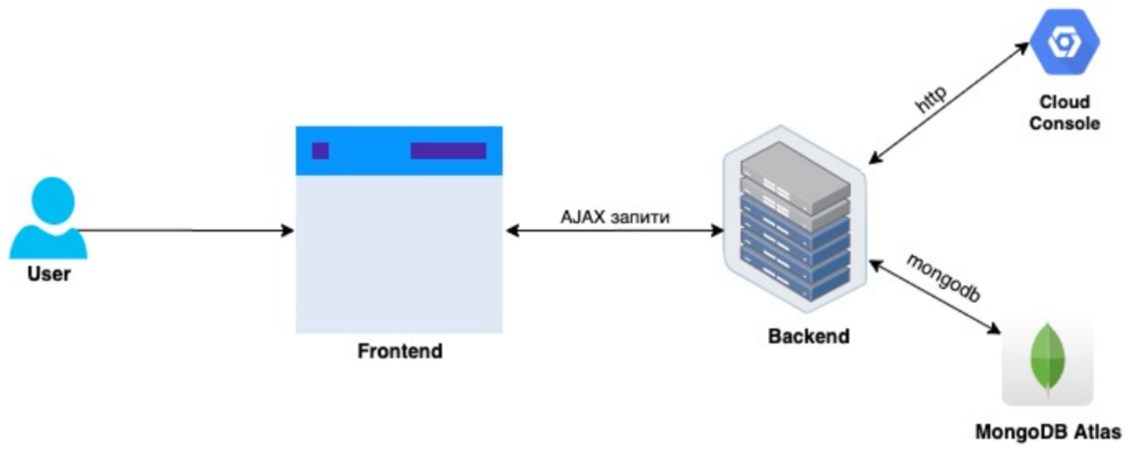
Architecture



Picture 1. Technologies

The web service consists of two parts: client (Frontend) and server (Backend). The user will interact with the client part, and it, in turn, with the server part, to provide and process all the necessary data (Picture 2).



Picture 2. Scheme of the web service

The client side sends **http** requests to get the required data from the backend. This is done using AJAX technology.

HTTP is a data transfer protocol used in computer networks. The name is shortened from Hyper Text Transfer Protocol, the protocol for transferring hyper-text documents HTTP belongs to the protocols of the OSI model of the 7th application level.

The server has an implemented HTTP-Server that waits for incoming requests. During authentication through Google Login, the server part generates cookies for the user, in which his identifier is encrypted. The browser stores this data.

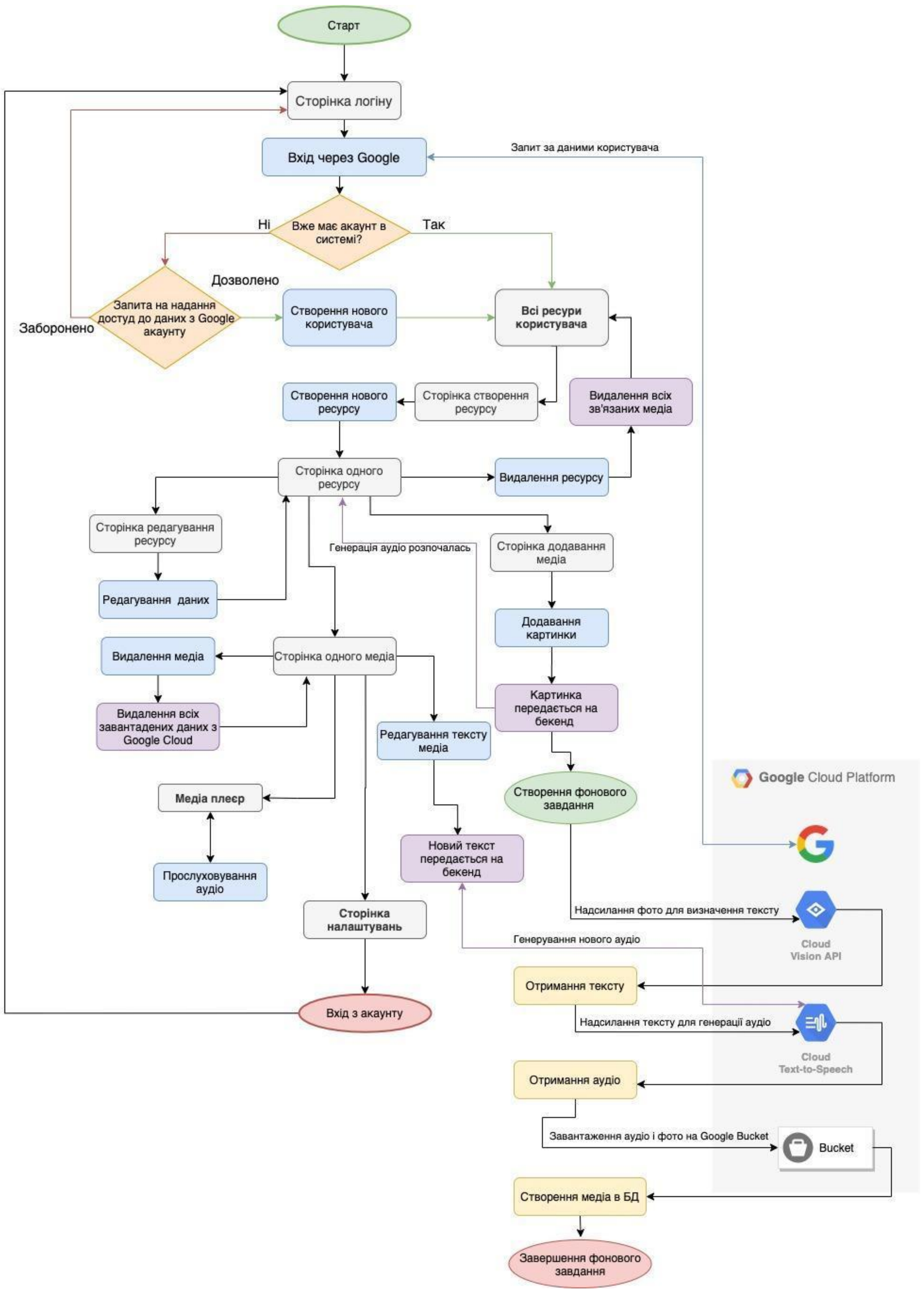
Cookie is a small piece of data sent by a web server and stored on the user's computer.

The client part is implemented in such a way that cookies are sent to the server with each HTTP request, so it has the ability to authorize the user by reading and decrypting them. Accordingly, by implementing a certain set of requests on the server, the client part of the service can get all the data the user needs.

The advantage of separating the client part from the server part is the low level of dependence of these two systems. Because they communicate over a defined protocol with a limited number of requests. It also makes it possible to implement a mobile version that will communicate with the server as well as the client browser version.

Considering the above, the client part will be implemented as SPA. Single Page Application(SPA) is a program that runs inside the browser and does not require reloading the page during use. You use this type of application every day. For example, it is Gmail, Google Maps, Facebook or GitHub.

Since the web service has quite a lot of pages on the client side, it is quite difficult to describe all possible scenarios of interaction with the service. The basic algorithm of the web service is shown in the Picture 3.


Старт
Сторінка логіну
Вхід через Google
Запит за даними користувача
Ні
Вже має акаунт в системі?
Так
Дозволено
Запита на надання доступ до даних з Google акаунту
Заборонено
Створення нового користувача
Всі ресури користувача
Створення нового ресурсу
Сторінка створення ресурсу
Видалення всіх зв'язаних медіа
Сторінка одного ресурсу
Видалення ресурсу
Сторінка редагування ресурсу
Генерація аудіо розпочалась
Сторінка додавання медіа
Редагування даних
Додавання картинки
Видалення медіа
Сторінка одного медіа
Картинка передається на бекенд
Видалення всіх завантадених даних з Google Cloud
Редагування тексту медіа
Google Cloud Platform
Створення фонового завдання
Медіа плеєр
Новий текст передається на бекенд
Прослуховування аудіо
Сторінка налаштувань
Надсилання фото для визначення тексту
Cloud Vision API
Генерування нового аудіо
Отримання тексту
Вхід з акаунту
Надсилання тексту для генерації аудіо
Cloud Text-to-Speech
Отримання аудіо
Завантаження аудіо і фото на Google Bucket
Bucket
Створення медіа в БД
Завершення фонового завдання

Picture 3. Web service algorithm

Service procedures

The basis of a web service is the REST API that it implements. The web service works on the basis of Node.js, accordingly compiles and executes JavaScript code. The npm Express module is used to develop the HTTP server. It runs a process that will accept requests from a specific port on your computer or server, process them, and return data. It is most convenient to work with data in JSON format.

The following main services for requests were implemented, which I divided according to the entities and services of the system. Namely:

- Auth – authorization and authentication service;
- Resources – a service that operates the user's resources;
- Media - a service that operates all the user's media;
- Images is a service that is responsible for uploading images when creating or editing a resource or media.

1. Service Auth

Serves the following ways:

- GET: api/auth/login - after the request, the user goes to the Google Login page, where he confirms the web service's access to his personal data.
- GET: api/auth/google/callback – the Google service makes a request after the user has successfully granted rights to read personal data. After they are saved, the user will be redirected to his resource page with cookies, where his data is encrypted for further data authentication.
- GET: /api/auth/logout – the user receives new cookies that are no longer active, so the user's session is deactivated.

- GET: /api/auth/logout-force – This request permanently deletes all records from the web service that the user has ever added. This includes:
    - All its resources are from the database;
    - All media that were linked from its resources from the database;
    - All images uploaded by him, which were linked to resources from the Google Cloud Storage service;
    - All images and audio uploaded by him were linked to media records from Google Cloud Storage.

2. Resources service

Serves the following ways:

- GET: api/resources - returns all resources created by the user
- POST: /api/resources - accepts JSON with the specified data about the resource, for example:
  ```
  {
     "title": "Best book 4",
     "notes": "Just for me",
     "image": "https://encrypted-tbn0.gstatic.com/image"
  }
  ```
  and creates a resource for the user
- PUT: /api/resources - accepts new data to update resource information in JSON format and returns the already updated resource to the user.
- DELETE: api/resources/:id – this request contains the :id parameter in the path. Upon receiving this service, the service deletes the corresponding resource, its image from Google Cloud Storage, all media associated with it and, accordingly, their images and audio files.
- GET: /api/resources/:id - this request contains the :id parameter in the path. By reading this parameter, the service finds the resource by its identifier and displays all information about the given resource, such as the title, notes, and attached media.

3. Images service

Serves the following ways:

- POST: /api/images/upload - accepts an image using the Form-Data interface. After that, it uploads it to the cloud service Google Cloud Storage and sends the URL link of this picture to the server.
- DELETE: /api/images – accepts JSON type data, example:
  ```
  {
  "data":"https://storage.googleapis.com/heard/storage/a5kb3w9n7z.jpg"
  }
  ```
  The value of the data key is a URL link to the image that will be deleted from Google Cloud Storage.

4. Media service

Serves the following ways:

- DELETE: /api/media/:id - this request replaces the :id parameter in the path. Receiving this identifier, the service deletes the corresponding media record from the database. The audio file generated for it and the picture attached by the user are also deleted.
- GET: /api/media/:id - by reading the id parameter, the service finds the resource by its identifier and returns the media data to the user, as well as the data of its parent resource.

- PUT: /api/media - accepts update data in JSON format and updates the media in the database with the received data. Since the media contains an audio recording, if the user edited the read text from the picture, a new audio recording should also be generated according to the new text. Therefore, to generate text, a request is made to the Google TextToSpeech service to receive new audio. The new binary audio file is then uploaded to Google Cloud Storage. The received URL to the audio file link is stored in the database, the audio duration in milliseconds is also calculated, so that the client part does not make additional requests to read this information when downloading the audio file. Updated media information is sent to the user.
- POST: /api/media - accepts data about the new media as input - a picture and the ID of the parent resource. The data is of type Form-Data, this type is used to transfer binary data by the client side.

Adding media in a web service is the most important functionality, as the usability of the service depends on it. That is why it has been optimized so that the main Node.js process is not blocked by the addition of new media and can serve many users at the same time. That is, so that the user can simultaneously listen to existing media and add new ones in the background.