**­­**Uploading of profile photos in the App.

**Why uploading photos?**

Having a profile picture for every user is an essential part of a successful app, so in Givantk app we made uploading of profile picture mandatory.

*This approach was highly inspired by the study performed by the researcher Stijn Baert from Ghent university, Belgium. In his article, which is called “Facebook profile picture appearance affects ­­recruiters’ first hiring decisions” he discussed the effect of facebook profile pictures on the hiring choices.*

The purpose of the study was to investigate whether the publicly available information on Facebook about job applicants affects employers’ hiring decisions.

The study concluded that Candidates with the most beneficial Facebook picture obtain approximately 38% more job interview invitations compared to candidates with the least beneficial picture.

So this study points out to the major importance of pictures on building trust, and obtaining higher hire rates.

**Where to save images?**

First we thought of saving the images on our backend server, but we had a technical issue, because whenever the server sleeps, we found that all the images are deleted because the server that we are using depends on what’s called ephemeral filesystem, where each dyno gets its own ephemeral filesystem, with a fresh copy of the most recently deployed code.

During the dyno’s lifetime its running processes can use the filesystem as a temporary scratchpad, but no files that are written are visible to processes in any other dyno and any files written will be discarded the moment the dyno is stopped or restarted.

For example, this occurs any time a dyno is replaced due to application deployment and approximately once a day as part of normal dyno management.

And as you can see this is an unreliable process, because all of the images will be deleted after few hours, and users will have to upload it again, which will make the user very frustrated.

**Using of Amazon Simple Storage service:**

****

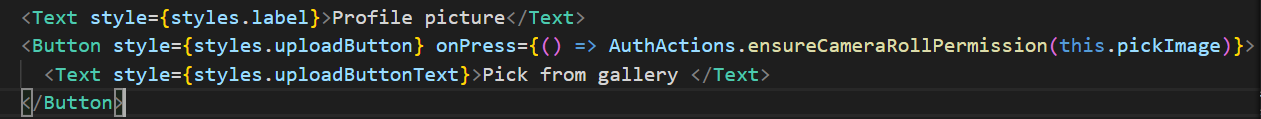
The alternative for this was using **Amazon Simple Storage Service** (Amazon S3), which is an object storage service that offers industry-leading scalability, data availability, security, and performance.

This means customers of all sizes and industries can use it to store and protect any amount of data for a range of use cases, such as websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics.

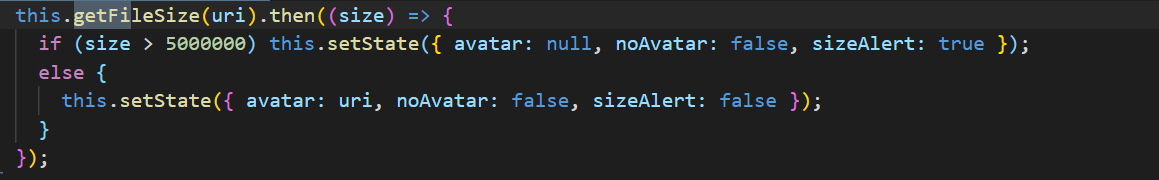
Amazon S3 provides easy-to-use management features so you can organize your data and configure finely-tuned access controls to meet your specific business, organizational, and compliance requirements. And it’s also designed for 99.999999999% (11 9's) of durability, and stores data for millions of applications for companies all around the world.

**How we integrated it with front and backend?**

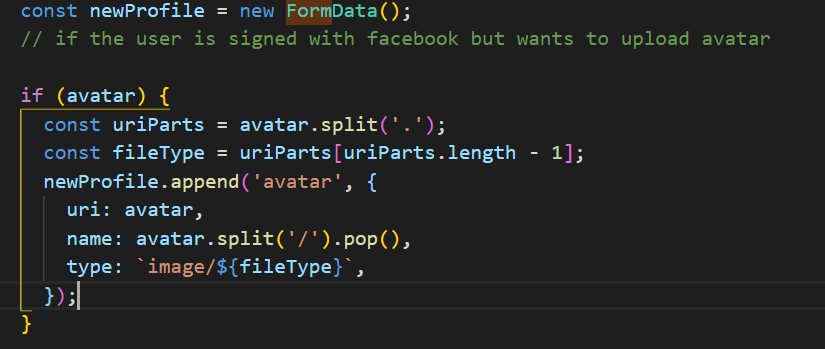
In the front-end we took the image from gallery or access the camera with the aid of image picker module from expo package. This step happens during the making of the user profile.



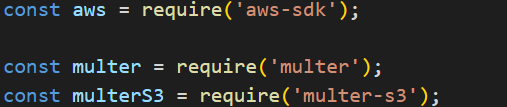
Using file system module imported from expo package, we check whether the image is smaller than 5 mega bytes

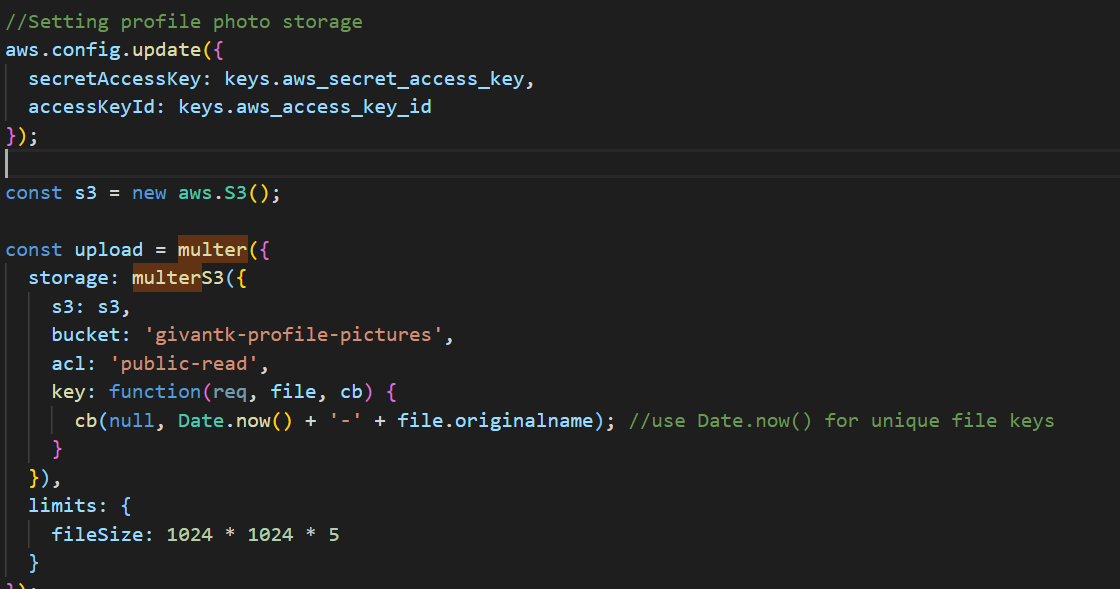


The next step was to send it to the backend as a “multi/part form data”



In the backend we used the multer package to integrate with amazon s3

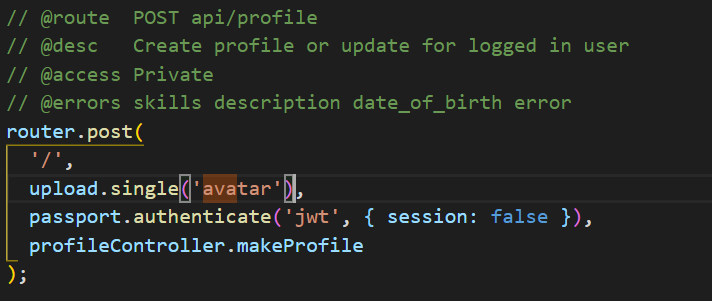




The pictures above demonstrate the following:

We first set the amazon aws configurations, by updating the secret access key, and the access key id.

Then we create a new s3 instance, and use it inside multer function, and set the name of bucket in which we store the pictures in, and the access control list type, which in this case is “public-read” as everybody is supposed to see these photos.



After that we send the following parameter to the router of making profile, which is (upload.single(‘avatar’)), this will result in the appearance of a file property in the request handled by the controller.

Also this file property has another property called location which holds the url on the Amazon Aws Server which displays this photo, we simply access it, then save it in the database in order to send it back to the frontend whenever a user image is to be displayed.

And here’s How the image appears in the frontend:

