STRV

MICROSERVICE DEVELOPMENT W/ GO

SEMESTRAL PROJECT

Version: 0.1.0
Last Update: 03 2024
Owner: Marek Cermak

INSTRUCTIONS

Design & Implement an API for a Go Newsletter platform. The platform enables registered users to curate and publish their own newsletters that other users can subscribe to. Follow the specifications below for further information.

Once finished, commit your source code to GitHub.
Invite:

- Marek Cermak (CermakM) on GitHub

For deployment, you can use any cloud platform. We recommend free plans on Railway, Render or paid alternatives such as AWS (free tier available), GCP (ditto) and Heroku. But ultimately, you are free to choose whatever cloud provider you want.

Once you complete your test project assignment, please send us the link to your git repository and the URL of the deployed backend API. You don't have to invite us to your deployed cloud solution.

SPECIFICATIONS

API	REST / GraphQL
Language	Go
Database	PostgreSQL*
Deployment	Any

*Do NOT use ORMs (please)

Project description

Design & Implement an API for a Go Newsletter platform. The newsletter app enables newsletter editors to register and manage their own newsletter that consumers can subscribe to. Editors can then publish news in the newsletter and send them to all subscribers.

API should be able to serve both mobile apps and websites using REST / GraphQL API. The API backend might use two different storage services to maintain the data. User accounts are stored in the Postgre SQL database, whereas subscribers are stored in Firebase.

Features Overview

Editor

- Sign up & Sign in
- Create, Rename and Delete a newsletter
- Publish a post to the newsletter
- [optional] List subscribers of the newsletter

Subscribers

- Subscribe to a newsletter using an email address
    - Receive an email (see "Publishing to Newletter") that confirms subscription with a link to unsubscribe
- Unsubscribe from a newsletter
    - There should be a link in every email that will allow the user to unsubscribe from a particular newsletter

Registration and Login

Registration

- All editors should be stored in persistent DB (PostgreSQL)
- Sign up using an email and a password (in the case of Firebase auth, feel free to implement even social auth)

Login

- Editors should be able to sign in using email & password
- Use stateless authorization (JWT)
    - you may use a 3rd party provider such as Firebase, you don't have to implement the authorization yourself
- [optional] Users can request a password reset

Creating a newsletter (authorized Editor)

- A newsletter is owned by a single Editor
- A newsletter requires a name and optionally a description
- A newsletter can be removed
- Newsletters can be stored in the database of your choice – either in PostgreSQL or in Firebase

Subscribing to a newsletter

- A newsletter has a unique link that subscribers can use to subscribe to the newsletter using their email address
- An email confirming the subscription is sent to the subscriber's email address
- Each email to subscribers contains a link to unsubscribe

Publishing to the newsletter

- Editors can publish to the newsletters they administer
- Published issues are sent to the subscribers using a mailing service of your choice, such as Resend, SendGrid or AWS SES
- [optional] Store sent issues in the Firebase

Firebase & Cloud account naming convention

When creating the **Firebase and Cloud accounts** related to the test project, please make sure to name it following this convention:

```
strv-vse-go-newsletter-[last_name]-[first_name]
```

REVIEW PROCESS

In general, the actual implementation is quite open-ended. The reason is that we want to see how you think in terms of backend architecture and development processes, and how you generally deal with the challenges you might face while implementing a backend API.

The following recommendations should help you determine what to focus on since these are the things we will be looking at while reviewing your project.

RECOMMENDATIONS

- Please do not consider the project to be a prototype. Think of it as a final release for your client.
- If you don’t think you can manage to implement some features (i.e. those marked as “optional”), prepare a document and describe how you would implement them, we can discuss it during the interview.
- Don't hesitate to use all the modern packages, technologies and architectures, regardless of the simplicity of the project.
- A good API is a documented API.
- Provide sufficient documentation. Think of this as the documentation that you will hand to the client and the client will use it to maintain the project.

Good luck!