Verloop Backend Engineering Challenge - Collaborative Story

Thank you for applying to Verloop! This challenge is to help us understand your programming style and expertise.

Challenge Duration

3 days from the time that we send this document to you. But we'd love to hear from you anytime between 1-2 days. We expect this to take 5-6 hours.

Metrics

Admin Stuff

- There's no restriction on the programming language or libraries
- It is okay to Google or Bing (if that's your thing)
- Write code as if you're writing for a production environment at scale. Expect our test suite to hit your API endpoints at thousand requests per second.
- We are very interested in your architecture/design decisions and empathy for your team members in terms of code readability and arrangement.
- Maintain a local .git folder and include it in the zip/7z/tar that you use to submit your solution. Commit atomically so we can follow your chain of thought.
- Please do NOT make the problem or your solution public. E.g don't push this to Github, Gitlab etc.

Bonus

- Write good, useful docstrings and comments
- Adding a Dockerfile
- Adding logging
- ER diagram of the database
- Write functional and unit tests
- Profile your code. What are the bottleneck functions?
- In addition to response time, measure requests/second that your endpoint can handle
- Using one of Python, Go or TypeScript
- Using one of MySQL, PostgreSQL or MongoDB

Submission

Once you are done, email us the solution and findings/notes/logs in a zip file titled <your-name>-hiring-challenge.zip

Add a README with the following details:

1. How to run your code? E.g. Do I need specific packages? Then add a requirements.txt or go.mod
2. If you had more time, what would you do differently?

After Submission

We will typically get back to you within 10 working days (or ~2 weeks). Faster, if you have a cloud deployment or a clearly neat implementation :)

Description

In this challenge you have to write an API server for a collaborative story writing. Participants build the story one word at a time.

- A story is made up of a title and paragraphs.
- Paragraphs are made of sentences.
- A single request can add exactly 1 word to the story.
- The first 2 words added to a story make the title.
- From the 3rd word, the first sentence of the first paragraph begins.
- As people add words, when there are 15 words, a new sentence starts.
- When there are 10 sentences, a new paragraph starts.
- When there are exactly 7 paragraphs, the story ends and a new one is created.

Technical details

The server should have 3 API end points.

- `POST /add` to add a new word.
- `GET /stories` returns list of stories.
- `GET /stories/:id` will return details of the story.

Other details:

- DB connection parameters will be passed in the `VERLOOP_DSN` env variable. For e.g. if you are using PostgresSQL, then expect this value to be of the format:

  ```
  postgresql://[user[:password]@][netloc][:port][,...][/dbname][?param1=value1&...]
  ```

- Debug logs can be turned on and off by passing value of `VERLOOP_DEBUG` env variable. Truthy values must turn debug level logs on.
- There is no concept of users in the system. Anyone with the endpoint URL, should be able to add a word to the story.
- The words in `/add` requests, do not have any order associated with them. For e.g. if hundreds of POST requests come in concurrently, then application may process all of those words in any arbitrary order.

POST /add

Request:

```
{
    "word": "verloop"
}
```

Response:

```
201 Created
```

```
{
    "id": 1,
    "title": "verloop",
    "current_sentence": ""
}
```

Request:

```
{
    "word": "hiring"
}
```

Response:

```
200 OK
```

```
{
    "id": 1,
    "title": "verloop hiring",
    "current_sentence": ""
}
```

Request:

```
{
    "word": "hi!"
}
```

Response:

```
200 OK
```

```
{
    "id": 1,
    "title": "verloop hiring",
    "current_sentence": "hi!"
}
```

Request:

```
{
    "word": "wanna fraandship?"
}
```

Response:

```
400 Bad Request
```

```
{
    "error": "multiple words sent"
}
```

GET /stories

Optional query parameters

param	type	values
limit	uint32	-
offset	uint32	-
sort	enum	created_at, updated_at, title
order	enum	asc, desc

Response:

```
200 OK
```


```
{
    "limit": 100,
    "offset": 0,
    "count": 1,
    "results":[
        {
            "id": 1,
            "title": "verloop hiring",
            "created_at": "2020-08-01T00:00:00Z",
            "updated_at": "2020-08-01T00:01:00Z"
        }
    ]
}
```

GET /stories/1

Response:

```
200 OK
```


```
{
    "id": 1,
    "title": "verloop hiring",
    "created_at": "2020-08-01T00:00:00Z",
    "updated_at": "2020-08-01T00:01:00Z",
    "paragraphs":[
        {
            "sentences":[
                "hi!"
            ]
        }
    ]
}
```