Form3 coding exercise design

Given the task to design an API that meets the following criteria:

- The API should be RESTful
- The API should be able to
    - Fetch a payment resource
    - Create, update, and delete a payment resource
    - List a collection of payment resources
    - Persist resource state

I propose the following five endpoints:

GET /v1/resources

This endpoint will list the available resources supporting two (optional) query parameters:

Query parameter	Default value	Type	Description
count	10	Int	The maximum number of resources returned.
after	null	ID	A resource ID indicating the start of the returned resources: All returned resources will come after this resource.

All query parameters are handled optimistically, which means that if an invalid count or ID is given, the query will fall back on its default values. There is also currently no upper limit on the count.

If successful, the endpoint produces a status code *200* and a JSON object with the following properties:

Property name	Type	Description
data	PaymentSummary[]	An array of payment summaries.
links	PageLinks	Page links which can be used for pagination.

The *PaymentSummary* type has the following properties:

Property name	Type	Description
id	ID	A unique resource identifier.
links	SelfLinks	Links to the resource.

The *PageLinks* type has the following properties:

Property name	Type	Description
self	URL	A link to the current page.
next	URL | null	A link to the next page. If this is *null* the end of the list has been reached.

The *SelfLinks* type has the following properties:

Property name	Type	Description
self	URL	A link to the current page.

Besides status code 200, the endpoint will send status code *500* upon internal server errors.

POST /v1/resources

This endpoint creates a new resource. There is currently minimal input validation other than the content type must be *application/json* and that the submitted object must be valid JSON, if this is not the case, the endpoint will respond with status code *400*. The input follows the shape of the examples provided in the assignment, where all fields, but version and ID can be submitted. All unsupported fields are ignored and all omitted fields are replaced by default Go values. This is an obvious place to improve the API in future works.

If successful, the API will respond with a status code *201* and the created resource, of type *Payment*, in the response body. The response will have the following properties:

Property name	Type	Description
id	ID	The unique resource identifier.
version	int	The resource version, which is incremented upon modification.
organisation_id	string	
type	“Payment”	
attributes	PaymentAttributes	
links	SelfLinks	Links to the current resource.

The *PaymentAttributes* type is omitted from this document but follows the shape of the examples provided.

PUT /v1/resources/{resourceID}

This endpoint updates the given resource identified by the *resourceID*. If the resource does not exist, the API will respond with a *404* status code.

The endpoint has the same requirements wrt input as the *POST /v1/resources* endpoint, but will upon success respond with a no-content *204* status code.

Calling this endpoint will increment the version number of the resource, and all attributes will be replaced by the input. Any omitted fields will fall back on the default Go values. This is not a *PATCH* endpoint, so omitting fields should probably result in a *400* status code. This is left as future work.

GET /v1/resources/{resourceID}

This endpoint retrieves the given resource identified by the *resourceID*. If the resource does not exist, the API will respond with a *404* status code.

Upon success, the endpoint will respond with status code *200* and a response body of type *Payment* (see *POST /v1/resources*).

DELETE /v1/resources/{resourceID}

This endpoint deletes the given resource identified by the *resourceID*. If the resource does not exist, the API will respond with a *404* status code.

Upon success respond with a no-content *204* status code.