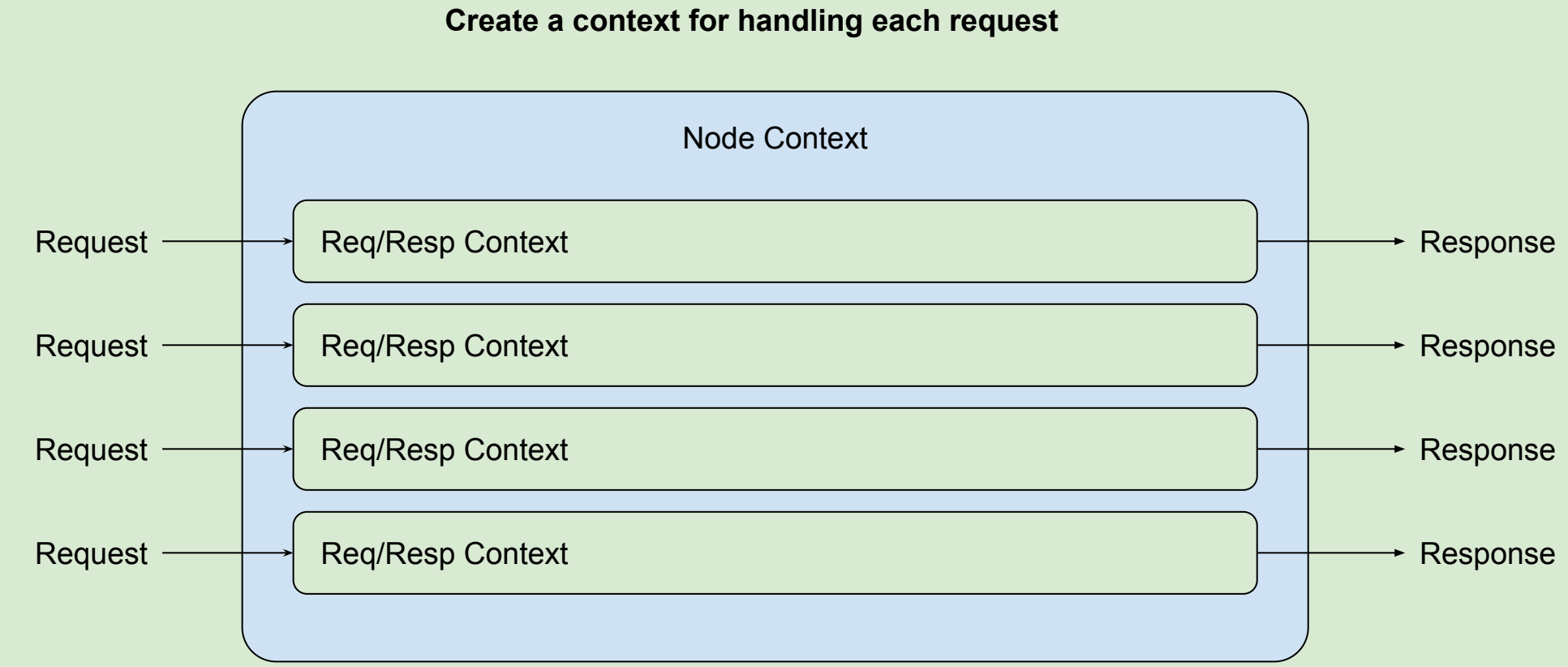
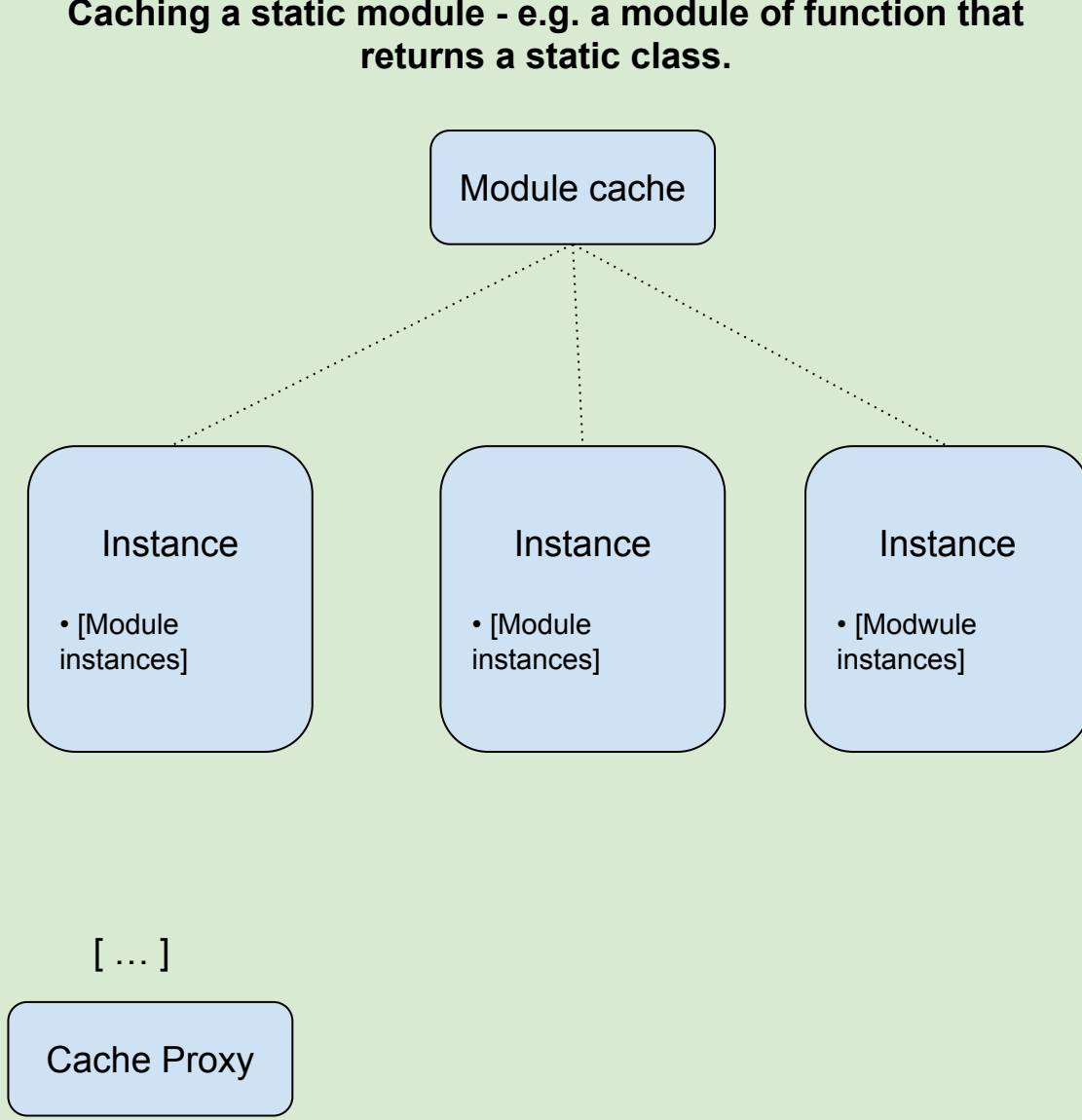
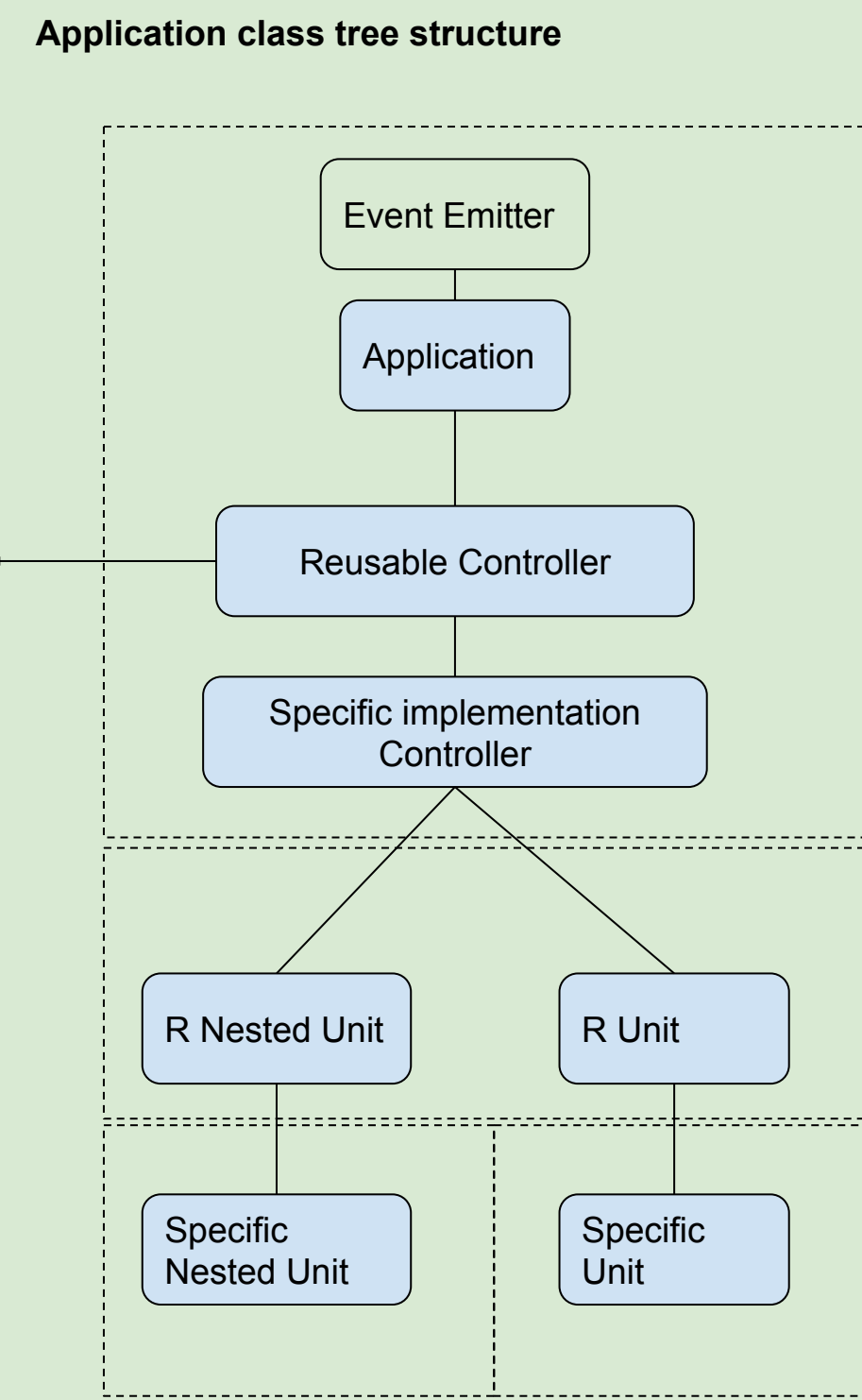


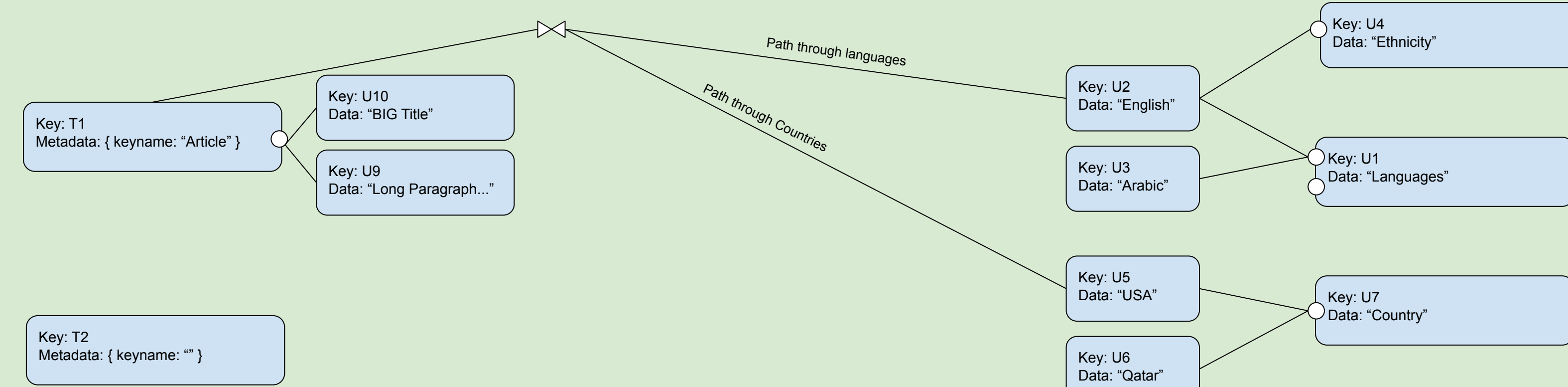
- Load modules
  - Create koa server
  - Add event listener (add static class to Application cache)
- Add event listeners
  -
- Initialize Application
  - Connect to database
  - Run event listeners:
    - Initialize database content
  - Port server:
    - Nested unit module new cache instance.
    - Apply koa middleware
      - New instance of nested unit.
    - Create http server



**Content**

Schema → resolver functions to fetch database.  
Writing api as a series of type definitions and attaching resolvers/functions to each one.  
Relational query - instead of hitting/calling 2 apis the data could be retrieved by a single call. Sort of combination of queries/collections/documents into a single trip. GraphQL makes it flexible.

- Creating endpoints for data
- Define unit structure of data which has mapping keynames to data
- Validation of data types and structure.



#### Schema field picking/filtering:

##### Nested Unit execution result:

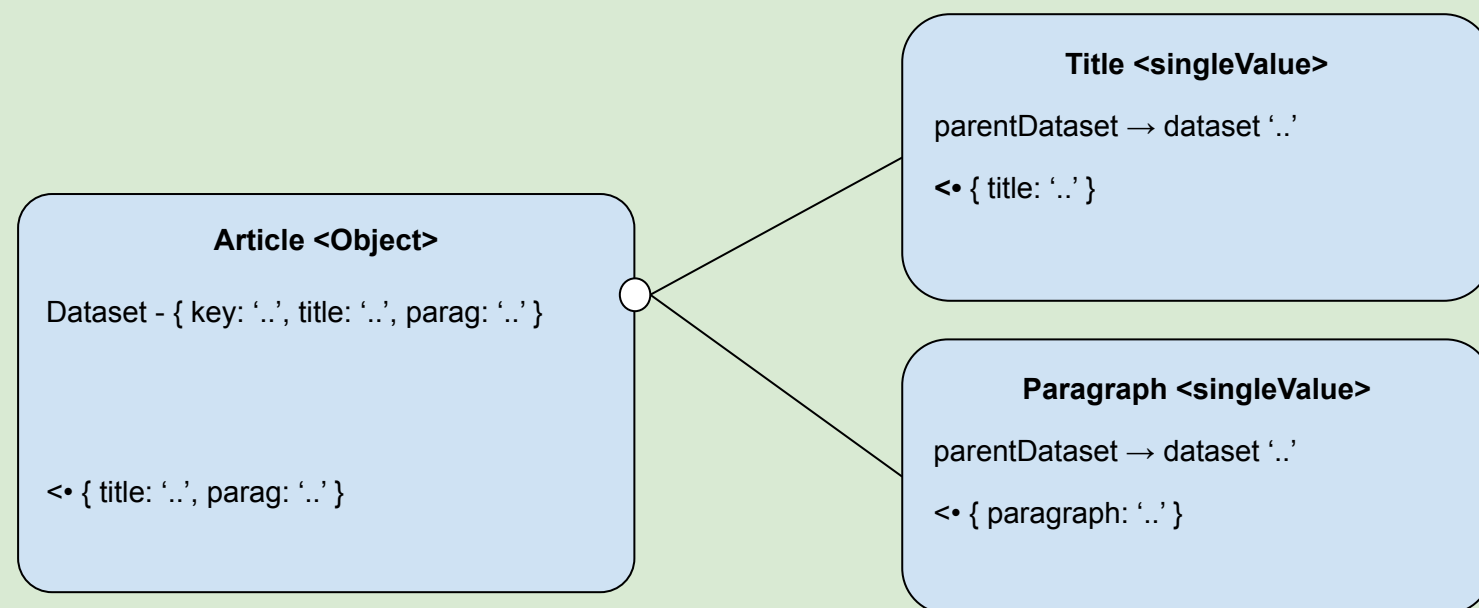
```
Fieldname: 'article'
Dataset: { paragraph }
Subsequent: [ Level 1
{
  Fieldname: title
  Dataset: { title }
  Subsequent: [ Level 2
    {
      Dataset: { short }
    }
    {
      Dataset: { long }
    }
  ]
}
]
{
  Dataset: paragraph
  Subsequent: []
}
```

**Request:**  
localhost/endpoint/?key=a1  
Entrypoint: "Article"  
Key: "a1"

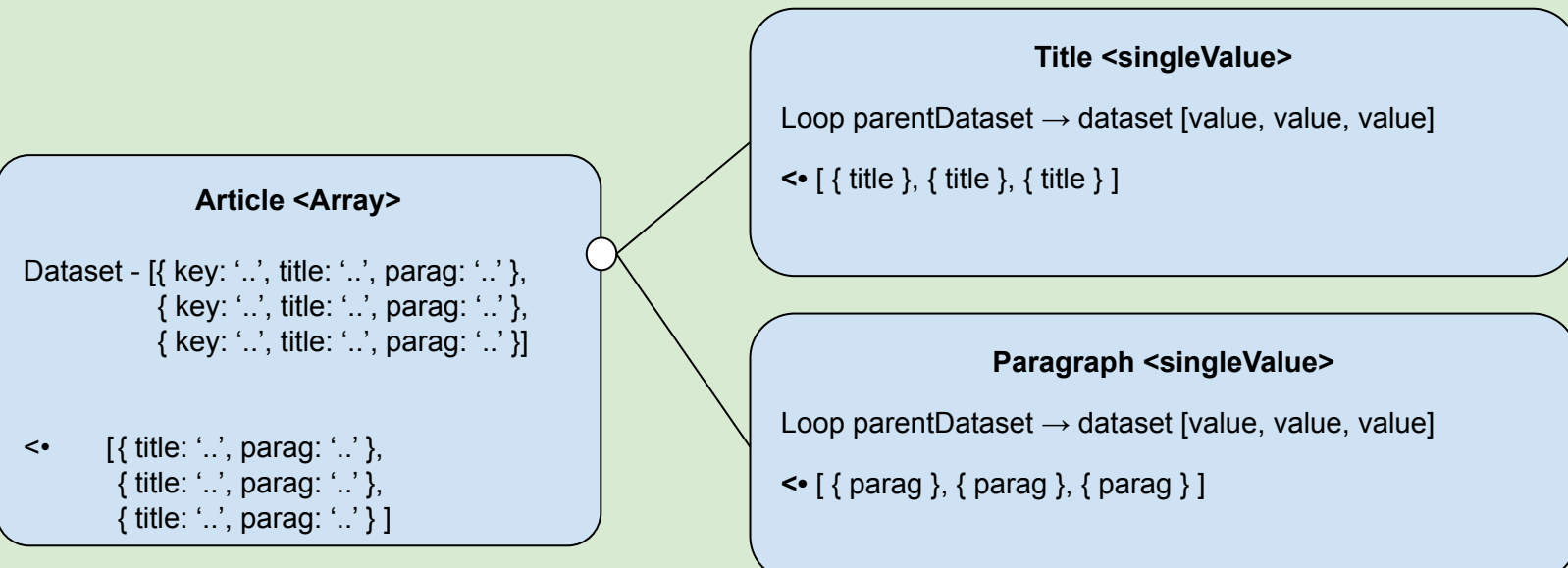
Request body data:  
{  
 Fieldname: article  
 subfield: [ Level 1  
 {  
 Fieldname: title  
 subfield: [ Level 2  
 { short },  
 { long }  
 ]  
 },  
 { paragraph },  
 { array... }  
 ]  
}

##### Request close to GraphQL format:

```
{Article
  Title
  Paragraph
  Array
}
```



Array handling Option 1 - deal with arrays in child using parent dataset



Array handling Option 2 - deal with arrays as own dataset (if dataset is an array execute nested units differently, looping through each object).

