Project Cloud Infantry
System Design

Table of Contents

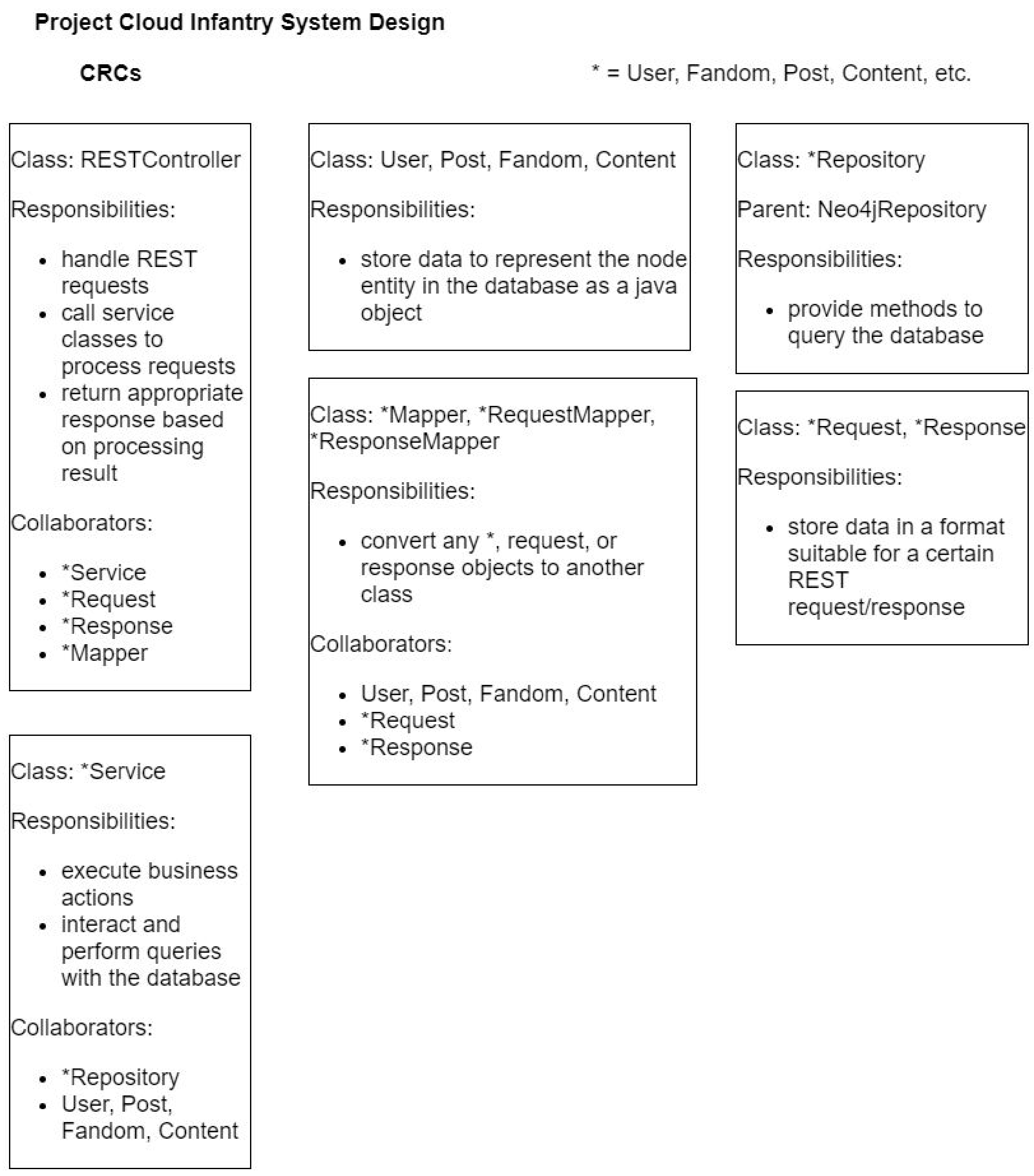
High-Level Description

Project CloudInfantry uses a single-page react app that makes requests to a Spring-Java application that manages a neo4j database.

Official React documentation covers the architecture of the framework, and can be found here: https://reactjs.org/docs/getting-started.html

The Spring-Java application uses spring-data to make querying the database easier and more compatible with Java. Core concepts from the official documentation were used in the architecture of the application. The reference documentation can be found here: https://docs.spring.io/spring-data/neo4j/docs/current/reference/html/#reference

CRCs



Project Cloud Infantry System Design

CRCs

* = User, Fandom, Post, Content, etc.

Class: RESTController

Responsibilities:

- handle REST requests
- call service classes to process requests
- return appropriate response based on processing result

Collaborators:

- *Service
- *Request
- *Response
- *Mapper

Class: *Service

Responsibilities:

- execute business actions
- interact and perform queries with the database

Collaborators:

- *Repository
- User, Post, Fandom, Content

Class: User, Post, Fandom, Content

Responsibilities:

- store data to represent the node entity in the database as a java object

Class: *Mapper, *RequestMapper, *ResponseMapper

Responsibilities:

- convert any *, request, or response objects to another class

Collaborators:

- User, Post, Fandom, Content
- *Request
- *Response

Class: *Repository

Parent: Neo4jRepository

Responsibilities:

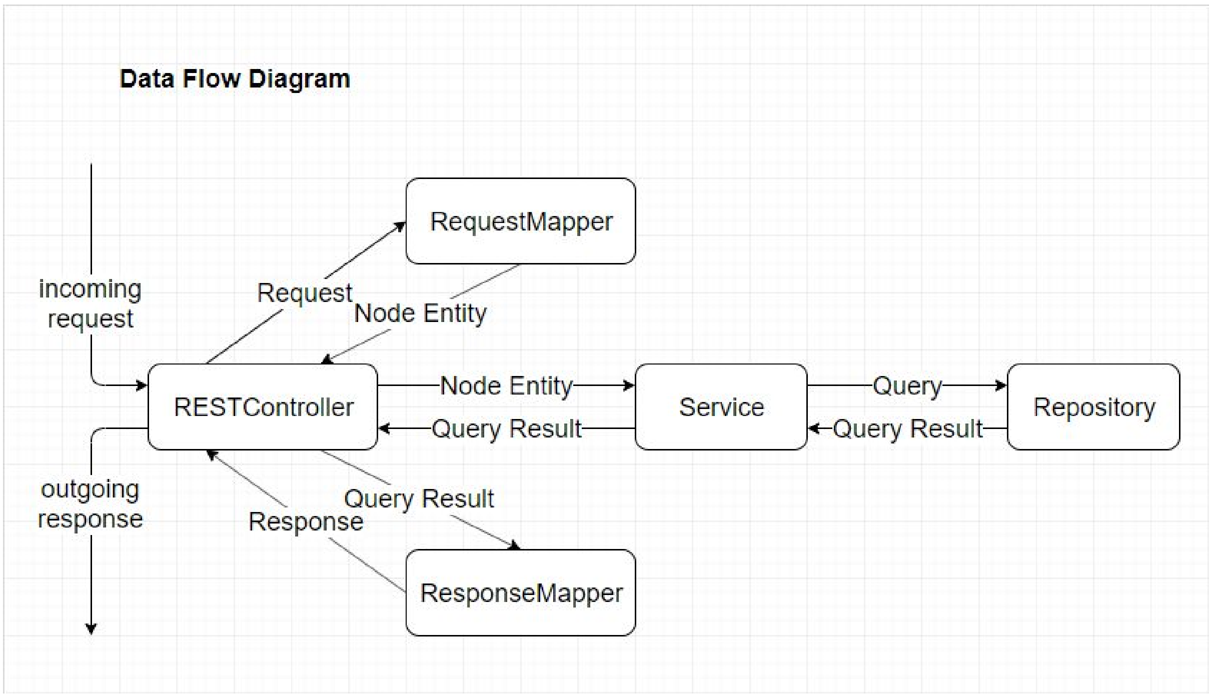
- provide methods to query the database

Class: *Request, *Response

Responsibilities:

- store data in a format suitable for a certain REST request/response

Data Flow



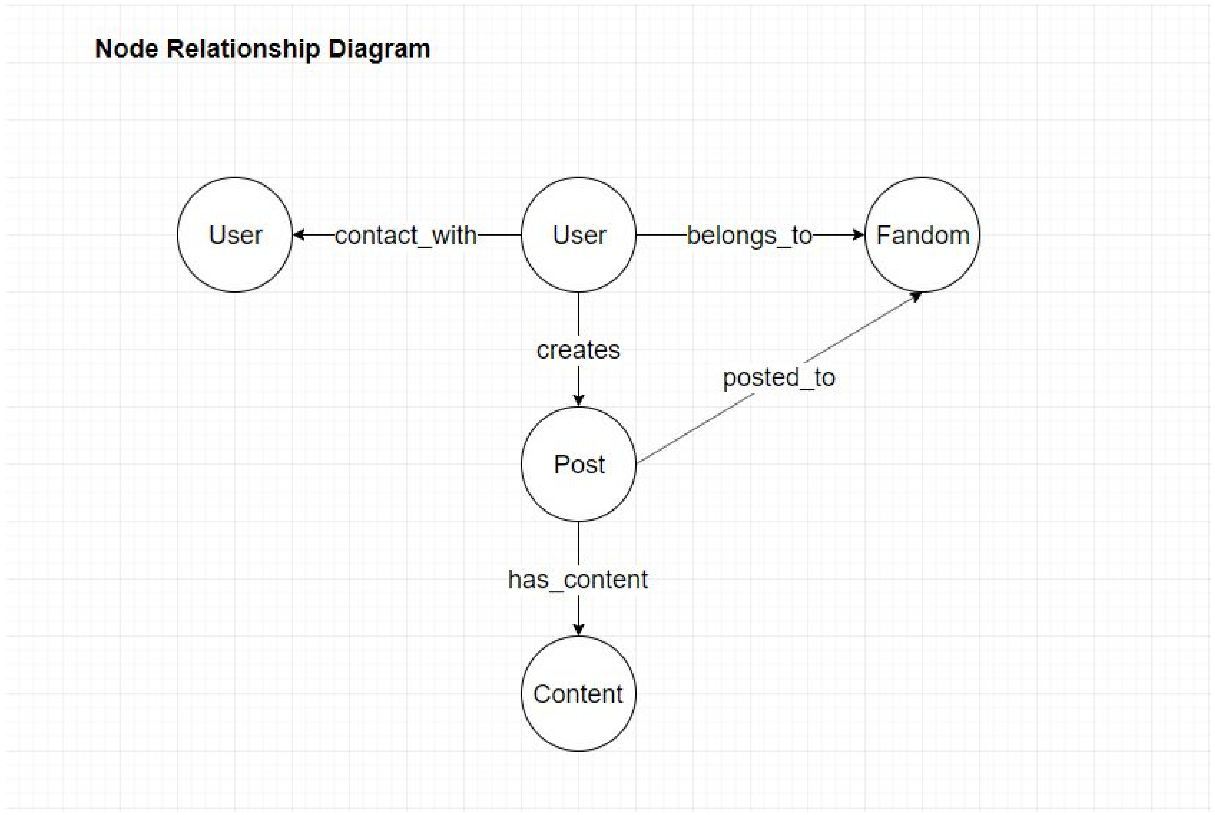
The RESTController receives and handles incoming http requests. It parses the contents and calls methods in the service classes to perform the required action.

The mapper classes convert data to another form to be used in the application. A request mapper will map incoming requests to their corresponding node entities (e.g. data for a new user converts to a User object) and a response mapper maps result data to the required response format.

Service classes interact with a specific repository class to perform business actions. They perform any validation requirements on the request and return any data necessary for the response.

Repository classes contain any queries that will be used on the database and make them accessible from function prototypes.

Node Relationships



Error Handling

From the backend,

```
200 OK if request successful

400 BAD REQUEST if request body is improperly formatted or missing required
information

404 NOT FOUND if <entity> with such id does not exist

500 INTERNAL SERVER ERROR if request was unsuccessful (Java Exception is thrown)
```

On the frontend, errors will be handled on a case-to-case basis as there may be unique ways of addressing errors depending on the component in use.