REST APIS

API

- Application Programming Interface
- Define a stable interface that allows access to a hidden entity.
  - *Interface defines methods*
  - *Legal inputs*
  - *What will be returned*
- Allows the internal representation of the entity to change without affecting others that depend on it.
- Suppose I promise that method ***foo*** is going to return a list of people. I notice that the people are returned in alphabetical order. Can I depend on that?

API - Example

- Besides having pages that allow you to interact with Twitter, it has an API that exposes certain pieces of information.
- Reference for just a tiny bit of it.
- Some other application can work with that API and provide customized interactions or perform data analysis.
- There is always tension over the amount of access that third parties are allowed.
  - *Data is worth money*
  - *Privacy is important*
    - Knowing who I follow may leak information about me and allow me to be targeted.
    - Facebook got in trouble for allowed too much leakage of data that was then used for political purposes.

REST

- Representational State Transfer
- A way of defining a resource through the use of multiple endpoints.
- Roy Fielding's dissertation (2000)

Core Ideas

- https://restfulapi.net/rest-api-design-tutorial-with-example/
- Separate out the user interface
- Stateless
  - *Request must have all the information*
  - *No stored context on the server*
  - *Session state completely client side*
- Cacheable
  - *Response can be reused for later equivalent requests*

Core Ideas

- Uniform Interface
  - *We define interface with constraints*
    - Identification of resources
    - Manipulation of resources via representations
    - Self descriptive messages
    - Hypermedia as engine of application state
- Layered System
    - No visibility beyond the layer an object is currently interacting with
- Code on Demand
  - Extend functionality by downloading scripts or applets

Resources

- *A key idea is that we are going to have named resources that will be exposed to the outside world in a consistent fashion.*
- *Nouns, not verbs*
- *May be singular or a collection.*
- *A instance of a resource may have a subcollection*
- *We identify a particular item in the collection by an id.*

- *Reference*

Resources Example

- */library*
  - A library
- */library/patrons*
  - A collection of library patrons
- */library/patrons/{patronID}*
  - A single patron
- */library/patrons/{patronID}/checkOut*
  - A single patrons checkout of a book
- */library/patrons/{patronID}/checkedBooks*
  - A single patrons books that have been checked out
- */library/books*
  - A collection of library books
- */library/books/{bookID}*
  - A single book

Resources

- *Document – singular collection. (eg Library). It may have fields with values and links to other resources.*
- *Collection – a server managed directory of resources. (eg. /Library/books)*
- *Store – a client managed resource repository. API allows client to add/access/delete. (eg. /Library/patrons/{patronId}/bookLists.) A URL designated by the client when added to the store.*
- *Controller – A model of a procedural concept. It can have inputs and outputs. (eg. /Library/patrons/{patronId}/checkout*

Naming Conventions

- *A forward slash indicates a hierarchical relationship between resources.*
- *Use hyphens to improve readability*
- *Don't use underscores*
- *Use lower case. The path portion of a URI is case sensitive.*
- *Don't include file extensions*
- *Don't use CRUD function names in resources.*

HTTP verb

- *We will use the HTTP request type to determine the CRUD operation.*

Resources Example

- *GET /library/patrons*
  - Get a list of patrons
- *POST /library/patrons*
  - Create a new patron

- *GET /library/patrons/{patronID}*
  - Get patron with the given ID
- *PUT /library/patrons/{patronID}*
  - Update patron with the given ID
- *DELETE /library/patrons/{patronID}*
  - Delete patron with the given ID

Search Results

- Provide capabilities to manage what is returned as list of items in a collection using query parameters
  - *General filter*
  - *Filter based on an attribute*
  - *Ordered*
  - *Paginated*

Resources Example

- *GET /library/books*
  - A collection of library books

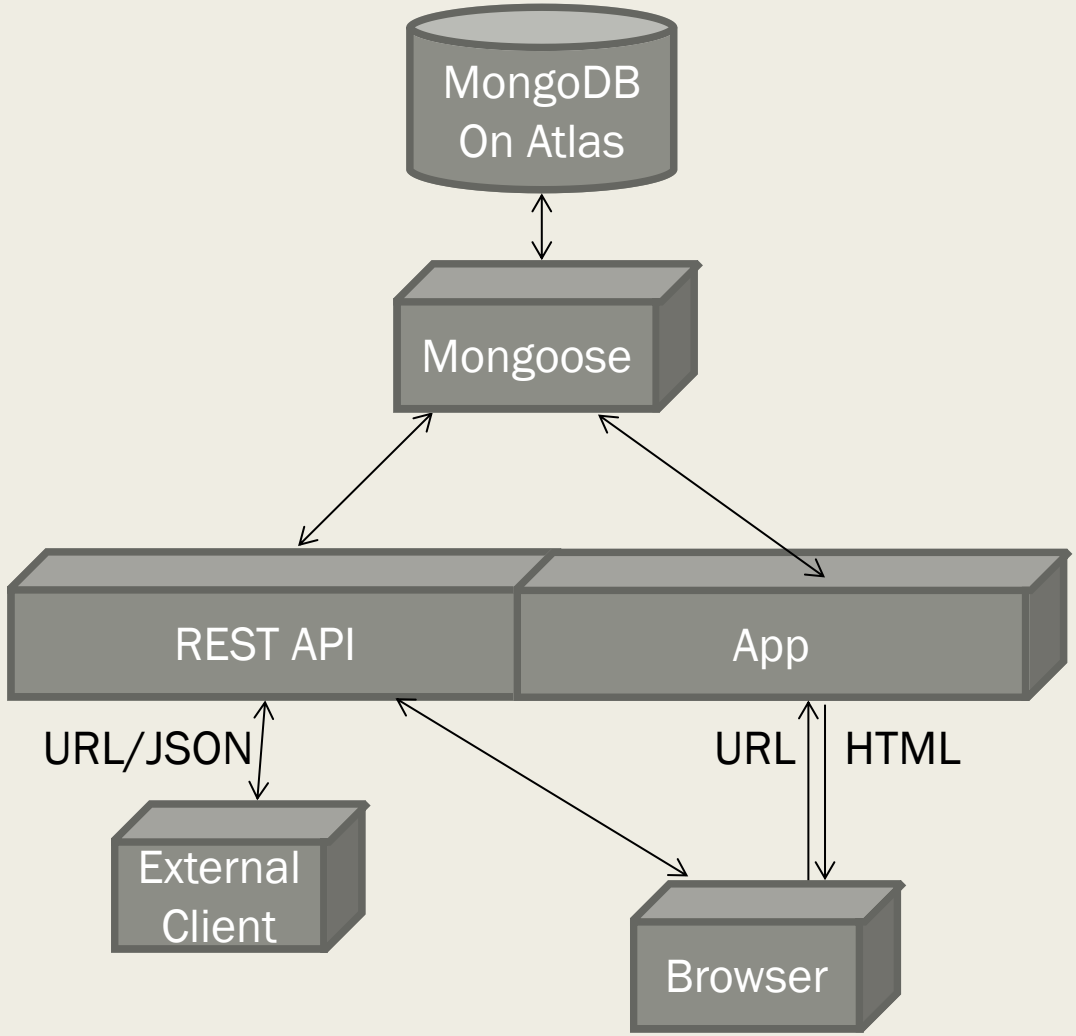
- *?genre= select based on genre*
- *?sortBy= sorted on an attribute*
- *?skip= number of results to skip*
- *?page= number of results to provide*

Relations

- Should we include in book a reference to the person that has checked out the book?
  - *Zero,One to One relation*
  - *If we include it, then there is a relation that we have to maintain in checkout*
  - *If we don't include it, then we probably want a specialized search on /library/books that goes through all the patrons to see who has the book.*

Testing Endpoints

- Postman
  - *Allows you defined collections of HTTP*
  - *Allows you to provide authentication*
  - *Allows the HTTP type/query parameters/body to be specified.*
  - *Displays the result returned as text*
- Curl
  - *Command Line utility*
  - *Allows you to requests*
  - *General filter*
  - *Filter based on an attribute*
  - *Ordered*
  - *Paginated*
- Swagger
  - *Set of tools that allow you to create and document APIs*
  - *Inspector allows for the creation of automated tests on the results of the endpoints in your API*



- Fetch is client side only.
- Server side will use Objects compiled from Schema.

References

- Restful server with mongoose and express
- MDN Tutorial This is a multipart tutorial that creates a library website using Mongoose and Express
- Handling Errors in Express

- It is useful to look at each of them in comparison. As you look at the differences, ask why.