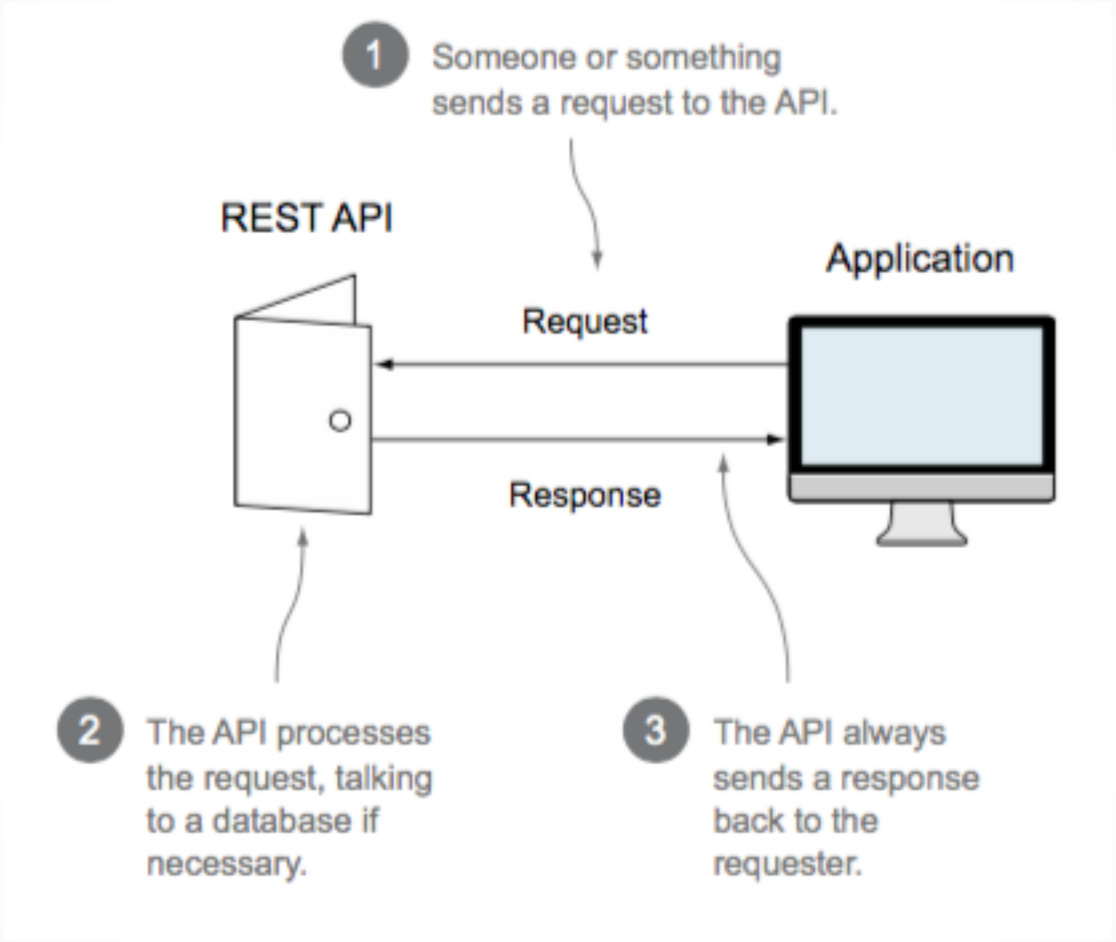
Preparing a REST API

Rules of REST APIs, API patterns, Typical CRUD operations

Rules for a REST API

- **Recall**:
  - REST – Representational State Transfer
  - REST is stateless—it has no idea of any current user state or history
  - API – Application Programming Interface
  - REST API – stateless interface to your application
- **Standards**:
  REST APIs have an associated set of standards
  - Generally best to stick to them
  - You're doing things the right way if you decide to make your API public


1 Someone or something sends a request to the API.
REST API
Application
Request
Response
2 The API processes the request, talking to a database if necessary.
3 The API always sends a response back to the requester.

Request URLs

- Request URLs for a REST API have a simple standard
  - Think about your DB collections
  - Will typically have a set of URLs for each collection
  - May also have a set of URLs for each set of subdocuments
  - Each URL in a set will have the same basic path, and some may have additional parameters
  - Within a set of URLs you need to cover a number of actions, generally based around the standard CRUD operations

Common actions

- Create a new item
- Read a list of several items
- Read a specific item
- Update a specific item
- Delete a specific item

URL paths and params for an API

Action	URL path	Parameters	Example
Create new contact	/contacts		/api/contacts
Read list of contact	/contacts		/api/contacts
Read specific contact	/contacts	contactid	/api/contacts/123abc
Update specific contact	/contacts	contactid	/api/contacts/123abc
Delete specific contact	/contacts	contactid	/api/contacts/123abc

Request methods used in a REST API

Request method	Use	Response
GET	Read data from DB	Data object answering request
POST	Create new data in DB	New data object as seen in DB
PUT	Update a doc in DB	Updated data object as seen in DB
DELETE	Delete an object from DB	Null

Request method links URL to desired action

Action	Method	URL path	Parameters	Example
Create new contact	POST	/contacts		/api/contacts
Read list of contact	GET	/contacts		/api/contacts
Read specific contact	GET	/contacts	contactid	/api/contacts/123abc
Update specific contact	PUT	/contacts	contactid	/api/contacts/123abc
Delete specific contact	DELETE	/contacts	contactid	/api/contacts/123abc

API URLs for subdocuments

- Subdocuments are treated in a similar way, but require an additional parameter

- E.g.:
  - **Action**: Create a new review for a product
  - **Method**: POST
  - **URL path**: /products/productId/reviews/reviewId
  - **Parameters**: productId, reviewId
  - **URL**: /api/products/123/reviews/abd

Responses and status code

- If you make a request, a good API will always respond and not leave you hanging
- Every single API request should return a response
- For a successful REST API, standardizing the responses is just as important as standardizing the request format.
- There are two key components to a response:
    - The returned data
    - The HTTP status code

Returning data from an API

- Your API should return a consistent data format
- Typical formats for a REST API are XML and JSON
- Our API will return one of three things for each request:
  - A JSON object containing data answering the request query
  - A JSON object containing error data
  - A null response

10 Most popular status codes

- A good REST API should return the correct HTTP status code

Status Code	Name	Use case
200	OK	A successful GET or PUT request
201	Created	A successful POST request
204	No Content	A successful DELETE request
400	Bad Request	n unsuccessful GET, PUT, or POST request due to invalid content
401	Unauthorized	Requesting a restricted URL with invalid credentials
403	Forbidden	Making a request that isn't allowed
404	Not Found	Unsuccessful request due to invalid parameter in URL
405	Method not allowed	Request method not allowed for given URL
409	Conflict	Unsuccessful POST request when another object with the same data already exists
500	Internal server error	Problem with the server or DB server

Setting up API in express

- We’ve already got a good idea about the actions we want our API to perform, and the URL paths needed to do so

- We need to setup controllers and routes to cause express to do something with an incoming URL
    - Controllers will do the actions
    - Routes will map incoming requests to appropriate controllers
    - Need to require the routes in app.js
    - Need to tell application when to use the routes
    - Define actions in the controllers

Next steps

- Either in the **routes** or controllers file, specify the following:
    - The request method
    - The required URL parameters
    - The definition of the full API routes
- In controller:
    - Return JSON and response status code from an Express request
    - Use Mongoose model to read data from mongoDB
    - Use Postman REST client to test requests to the API

Reading data from MongoDB

- Mongoose models have several methods available to them to help with querying the database.

- Here are some of the key ones:
  - **find** - General search based on a supplied query object
  - **findById** - Look for a specific ID
  - **findOne** - Get the first document to match the supplied query
  - It's good to run query with exec()
  - Be sure to catch errors and return appropriate response

Limiting return paths

- Limiting the data being passed around is better for bandwidth consumption and speed

- Mongoose does this through a **select** method chained to the **model** query

```
Product
  .findById(req.params.productid)
  .select('name reviews')
  .exec(
      function(err, product) { // do error checking-product
        var review;
        review = product.reviews.id(req.params.reviewid);
        // do error checking for review
} );
```

-

Resources

- http://www.restapitutorial.com/httpstatuscodes.html

- Getting MEAN with Mongo, Express, Angular, and Node
  Simon Holmes
  November 2015
  ISBN 9781617292033
  440 pages printed in black & white