Advanced API Design

Device

PC
Smart TV
Game Console
Mobile …etc

Application

iOS APP
Android APP
Desktop…etc

Browser

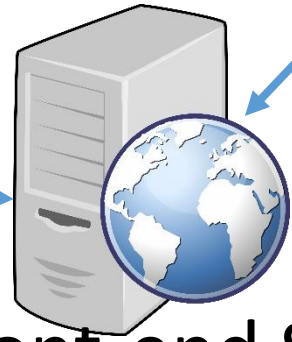
Chrome
Firefox
Safari…etc

Database Server

SQL (MySQL, Oracle..etc)
noSQL (MongoDB, CouchDB ..etc)
Graph (NeoDB)
Files system

API Server

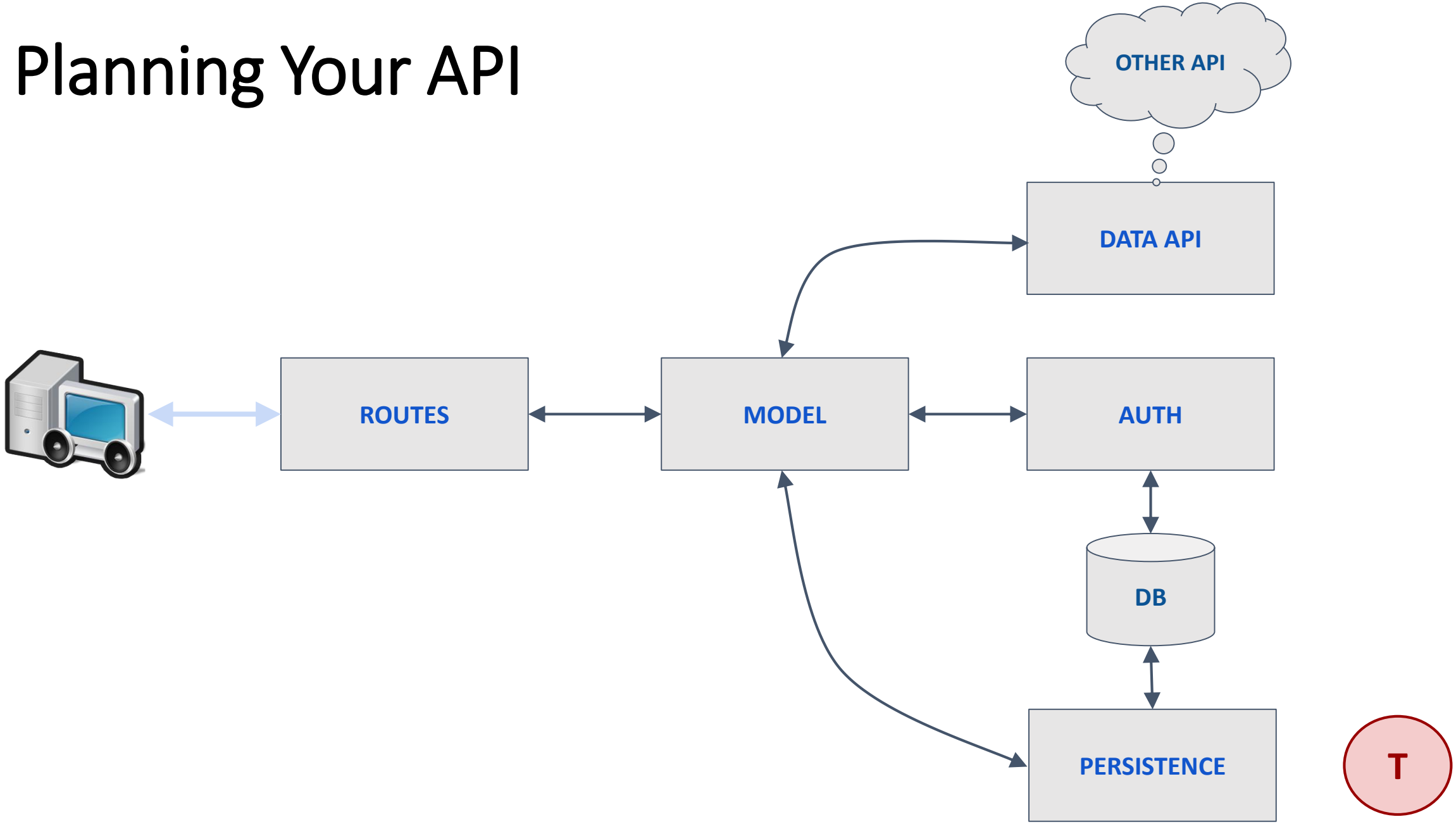
NodeJS Server



Front-end Server

Bootstrap with Jquery, Angular, or Vue.JS…etc served by any HTTP server (Express NodeJS, Python, PHP..etc)

Planning Your API



Assignment Update

- ~~GIT repositories~~
- ~~Plan DB, API and Frontend~~
- ~~Persistent Storage~~
- ~~Basic API with Node.JS~~
- ~~Layout contents with Bootstrap~~
- ~~Create Bootstrap forms~~
- ~~Connecting Frontend with Backend (read/write)~~
- Dynamically change contents of frontend
- API security & Authentications
- Complete API methods (GET, POST, PUT, DELETE, PATCH)
- Unit Testing
- Documenting API

REST Terms Definitions

- Resource: A single instance of an object. For example, a user.
- Collection: A collection of homogeneous objects. For example, users.
- HTTP: A protocol for communicating over a network.
- Consumer: A client computer application capable of making HTTP requests.
- Third Party Developer: A developer not a part of your project but who wishes to consume your data.
- Server: An HTTP server/application accessible from a Consumer over a network.
- Endpoint: An API URL on a Server which represents either a Resource or an entire Collection.
- Idempotent: Side-effect free, can happen multiple times without penalty.
- URL Segment: A slash-separated piece of information in the URL.

Versioning

api/v1/

- No matter what…your code is going to be changed
- an API is a published contract between a Server and a Consumer. If you make changes to the Servers API and these changes break backwards compatibility
- To ensure your application evolves AND you keep your Consumers happy, you need to occasionally introduce new versions of the API while still allowing old versions to be accessible.
- If you are only adding endpoints to your API, you do not to increment your version.

HTTP Methods (Verbs)

- GET (SELECT): Retrieve a specific Resource from the Server, or a listing of Resources.
- POST (CREATE): Create a new Resource on the Server.
- PUT (UPDATE): Update a Resource on the Server, providing the entire Resource.
- PATCH (UPDATE): Update a Resource on the Server, providing only changed attributes.
- DELETE (DELETE): Remove a Resource from the Server.

- HEAD – Retrieve meta data about a Resource, such as a hash of the data or when it was last updated.
- OPTIONS – Retrieve information about what the Consumer is allowed to do with the Resource.

Use URIs to specify your objects, not your actions

POST: Api/v1/users/add
POST: api/v1/users/delete/12
POST: api/v1/users/update/12

GET: /users	- Retrieves a list of tickets
GET: /users/12	- Retrieves a specific ticket
POST: /users	- Creates a new ticket
PUT: /users/12	- Updates ticket #12
PATCH: /users/12	- Partially updates ticket #12
DELETE: /users/12	- Deletes ticket #12

- Methods on Collections:
  - GET Retrieve the collection
  - POST Add a new resource to the collection
- Method on a resource:
  - GET Retrieve the resource
  - PUT Update the entire resource
  - Patch Partially update the resource
  - DELETE Remove the resource

Put vs PATCH

```
PUT /users/1
{
   "username": "skwee357",
   "email": "skwee357@gmail.com"       // new email address
}

PATCH /users/1
{
   "email": "skwee357@gmail.com"       // new email address
}
```

Update Password

- POST: /users/:user_id/reset_password
- You have a collection of users, where the single user is specified by the {user_id}. You would then specify the action to operate on, which in this case is reset_password.
- It is like saying "Create (POST) a new reset_password action for {user_name}".

Handling Errors

- API consumers see the API as a black box
- Each response needs to include a status code

  e.g. `401 UNAUTHORISED`
- Include a human-readable message in the body

```
{
    message: "Basic Authentication required"
}
```

Common Status Errors

- 201 Created after creating a resource; resource must exist at the time the response is sent
- 202 Accepted after performing an operation successfully or creating a resource asynchronously
- 400 Bad Request when someone does an operation on data that's clearly bogus; for your application this could be a validation error; generally reserve 500 for uncaught exceptions
- 401 Unauthorized when someone accesses your API either without supplying a necessary Authorization header or when the credentials within the Authorization are invalid; don't use this response code if you aren't expecting credentials via an Authorization header.
- 403 Forbidden when someone accesses your API in a way that might be malicious or if they aren't authorized
- 405 Method Not Allowed when someone uses POST when they should have used PUT, etc
- 413 Request Entity Too Large when someone attempts to send you an unacceptably large file
- 418 I'm a teapot when attempting to brew coffee with a teapot

Partial Response

- Allows you to supply only the information needed
- Reduces the bandwidth

/books?fields=title,author

/books/1449336361?fields=title,author

Searching Collections

- Search should be considered for collections
- Search query passed as a parameter

  /books?q=javascript

Collection Pagination

- Avoid returning the entire collection!
- Default limit should be decided (first 20 records?)
- Allow developers to specify fewer records
- Also need to specify which block to return.

  /books?limit=15&offset=45

Multiple Data Formats

- If multiple formats are available client needs to specify
- Two options:
  - Accept header

    Accept: application/json
  - Type format in the URL

    /books/1449336361.json

Content Not Changed?

- GET response should include Last-Modified header
- GET request should include If-Modified-Since header
- If match the server responds with 304 NOT MODIFIED and the headers (no response body)
- If content has changed the server responds with 200 OK and requested resource.
-

- GET response should include ETag header
- GET request should include If-None-Match header
- If match the server responds with 304 NOT MODIFIED and the headers (no response body)
- If content has changed the server responds with 200 OK and requested resource.
-

- Last-Modified / If-Modified-Since
- Uses the RFC 2822 date format

  Wed, 21 Oct 2015 07:28:00 GMT
- ETag / If-None-Match

  Hash of the response body

  686897696a7c876b7e

Generating ETAG

- Use the etag module

```
const etag = require('etag')
res.setHeader('ETag', etag(body))
```

- No authentication status stored in API
- Each request must provide the authentication needed
- Basic Access Authentication
- OAuth 2.0

- Don't use query parameters to alter state
- Don't use mixed-case paths if you can help it; lowercase is best
- Don't use implementation-specific extensions in your URIs (.php, .py, .pl, etc.)
- Don't fall into RPC with your URIs
- Do limit your URI space as much as possible
- Do keep path segments short
- Do prefer either /resource or /resource/; create 301 redirects from the one you don't use
- Do use query parameters for sub-selection of a resource; i.e. pagination, search queries
- Do move stuff out of the URI that should be in an HTTP header or a body

- Don't ever use GET to alter state; this is a great way to have the Googlebot ruin your day
- Don't use PUT unless you are updating an entire resource
- Don't use PUT unless you can also legitimately do a GET on the same URI
- Don't use POST to retrieve information that is long-lived or that might be reasonable to cache
- Don't perform an operation that is not idempotent with PUT
- Do use GET for as much as possible
- Do use POST in preference to PUT when in doubt
- Do use POST whenever you have to do something that feels RPC-like
- Do use PUT for classes of resources that are larger or hierarchical
- Do use DELETE in preference to POST to remove resources
- Do use GET for things like calculations, unless your input is large, in which case use POST

- Do use the appropriate status code
- Do use caching headers whenever you can
  - ETag headers are good when you can easily reduce a resource to a hash value
  - Last-Modified should indicate to you that keeping around a timestamp of when resources are updated is a good idea
  - Cache-Control and Expires should be given sensible values
- Do everything you can to honor caching headers in a request (If-None-Modified, If-Modified-Since)
- Do use redirects when they make sense, but these should be rare for a web service

- More Information:

Best Practices for Designing a Pragmatic RESTful API

http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api#method-override