Module 3: SaaS Architecture

...

Patterns

Due Dates

CHIP 3.3 due Sep 9th

Module 3 self-checks due this Friday on Sep 10

Patterns in Software Development

- Software Design Pattern: a general architectural solution to family of similar problems obtained by generalizing from the experience of users who have solved those.
- Software Architecture Pattern: how subsystems are connected together to meet the application's functional and non-functional requirements.

Software architecture shows the system's structure and hides the implementation details, focusing on how the system components interact with one another.

Software design, on the other hand, concentrates on the system's implementation, often delving into considerable detail

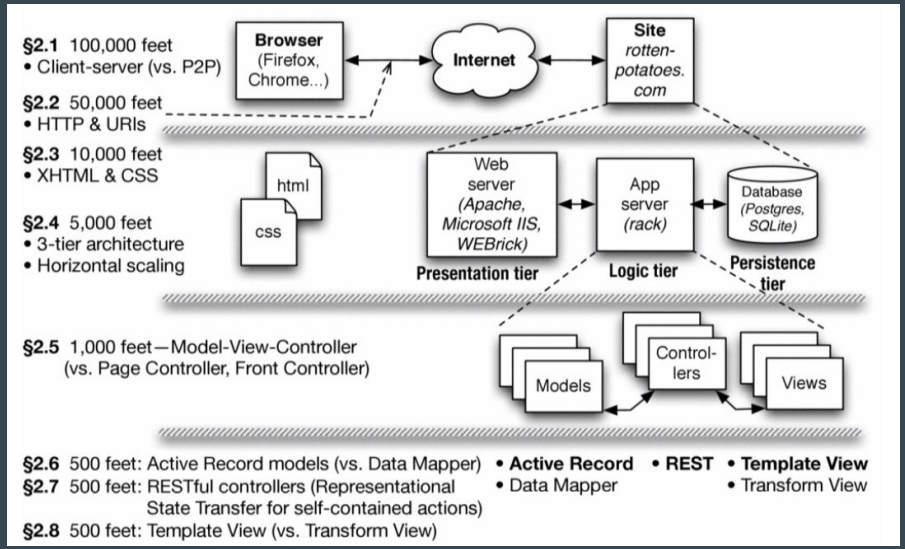
Question 1

Compare and contrast client-server architecture to peer-to-peer architecture

What are the benefits of each?

Patterns to know:

- Patterns in internet based services:
  - Client-Server
  - Peer-to-Peer
- Saas apps follow the client-server design pattern where a client makes requests and server responds to many client requests. Client-Server captures the reusable behavior of specialized servers and clients without specifying how servers and clients should be implemented.
- Peer-to-Peer design would cause all entities to act as clients and servers. While flexible, it makes it difficult to implement software to do either job well.



Saas Web Apps are examples of client-server pattern

Web Infrastructure, HTTP, Restful APIs

1.1 Requests

Without looking at any documentation, discuss what the following HTTP requests might do (no specific answer format required, just a general idea). *Note*: These links do not work, but determine what the request type and arguments would do if they did exist.

- GET https://github.com/orgs/cs169/repos
- GET https://github.com/repos/cs169/homework2
- POST https://github.com/orgs/cs169/repos

1.1 Requests

Without looking at any documentation, discuss what the following HTTP requests might do (no specific answer format required, just a general idea). *Note*: These links do not work, but determine what the request type and arguments would do if they did exist.

- GET https://github.com/orgs/cs169/repos
- GET https://github.com/repos/cs169/homework2
- POST https://github.com/orgs/cs169/repos

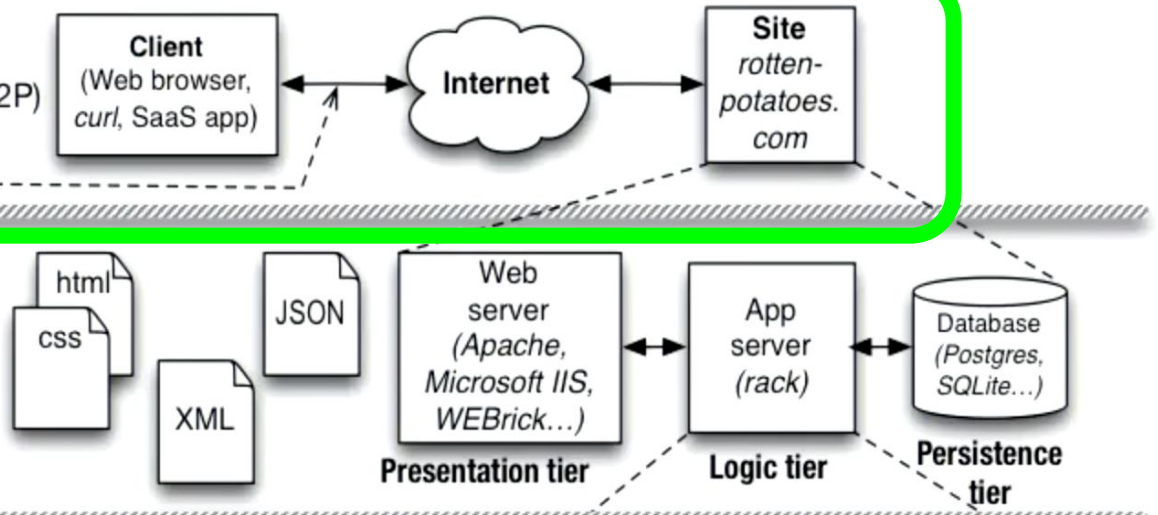
- GET https://github.com/orgs/cs169/repos
  Retrieves all the repositories for the organization "cs169"
- GET https://github.com/repos/cs169/homework2
  Retrieves repository called "homework2" for the organization "cs169". (Food for thought: How should this API respond if the repository doesn't exist?)
- POST https://github.com/orgs/cs169/repos
  Publish/create a repository for organization "cs169". (Food for thought: If this repository is to be created, what other information should be sent? How might we send this information with the post request? If we use JSON, how could we format the necessary information? No right answers, but important questions to ask!)

§2.1 100,000 feet
- Client-server (vs. P2P)

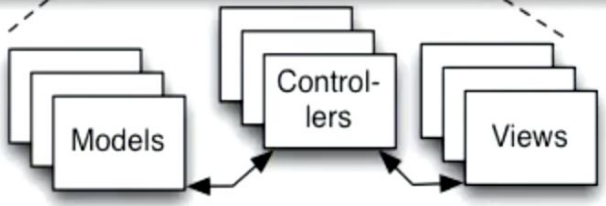
§2.2 50,000 feet
- HTTP & URIs

§2.3 10,000 feet
- HTML & CSS, JSON, XML

§2.4 5,000 feet
- 3-tier architecture
- Horizontal scaling



§2.5 1,000 feet—Model-View-Controller (vs. Page Controller, Front Controller…)



§2.6 500 feet: Active Record models (vs. Data Mapper)
- **Active Record**
- Data Mapper

§2.7 500 feet: RESTful routes & controllers for self-contained actions)
- **REST**

§2.8 500 feet: Template View (vs. Transform View)
- **Template View**
- Transform View

Overview of Web Infrastructure

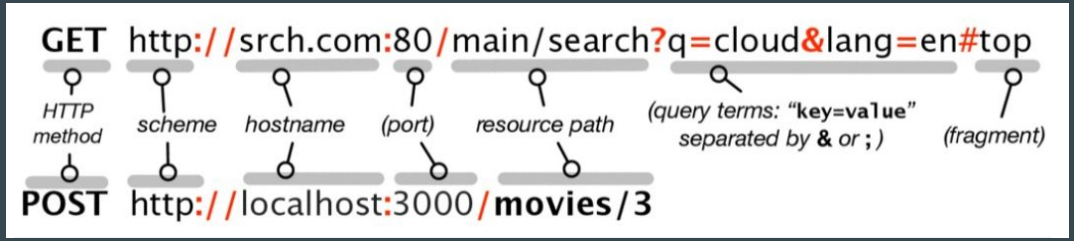
Web Before: collection of static pages that are delivered to a universal viewer
Web Now: collection of programs that can be invoked remotely to return any kind of data (not just a static web page to be viewed)

Remote procedure calls where a web client can call a function that gets executed on another computer and returns an answer to the client
Web = collection of services

TCP/IP → HTTP

- Allows communication of arbitrary character strings between a pair of network agents.
- In TCP/IP network each computer has an IP Address. Browsers automatically contact DNS and get the IP address.
- Referring to localhost points to the very computer app is running on.
- Multiple agents on a network can be running at the same IP address, so TCP/IP uses port numbers to distinguish different network agents at the same IP address.
- All protocols based on TCP/IP like HTTP must specify the host and port number while opening a connection. Once one agent opens a connection to another agent using hostname and port number as specified in TCP/IP, the communication in HTTP is then initiated.
- An HTTP server process must be listening for connections on that host and port number.

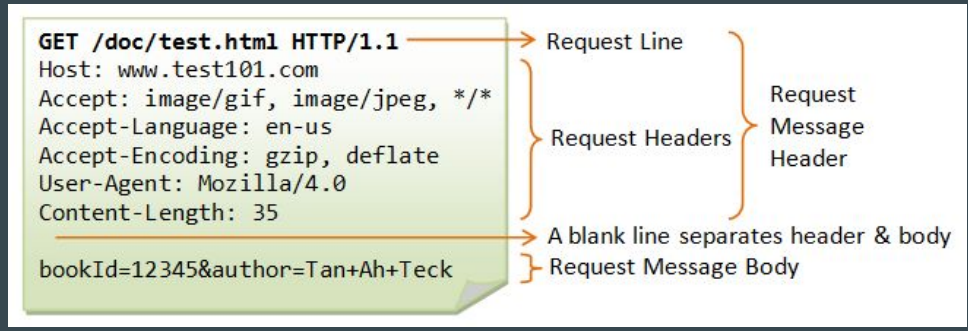
HTTP Request

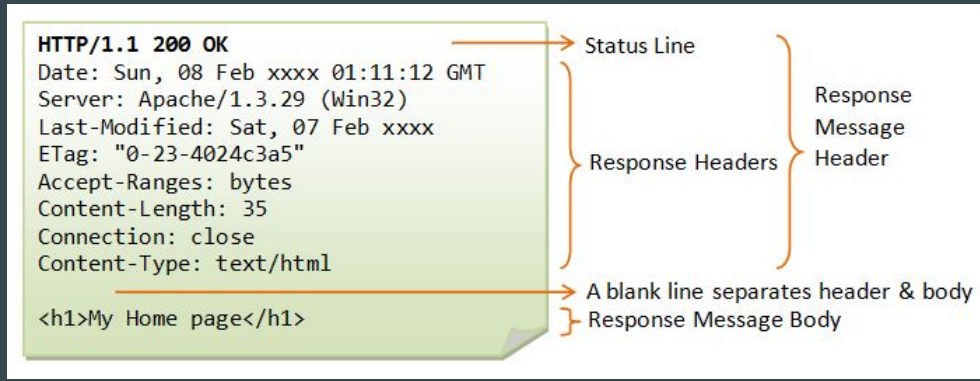


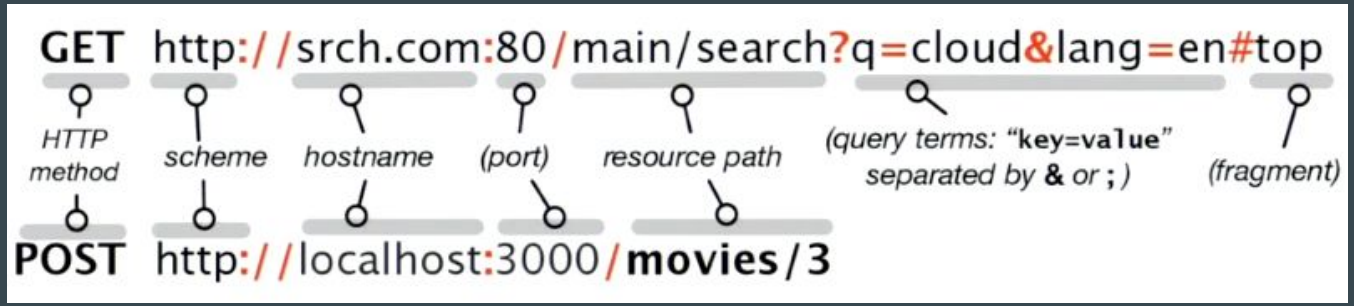
- HTTP request = HTTP method (GET,POST, PUT, DELETE) + URI
- URI (Uniform Resource Identifier) is the string typed in address bars that shows:
  - Name of the communication scheme to retrieve info: HTTP
  - Hostname with optional port number: “cs169.com” or “localhost:3000”
  - Resource on that host that user wants to retrieve: “movies/15/reviews”
- URI means a resource available on the internet.
- HTTP is a stateless protocol:
  - independent requests every time
- Simplify server design at expense of application design:
  - now remembering the client requires cookie generation and client has to include right cookies with each request it sends
- HTTP is a request-response protocol

HTTP Communication (CHIP 3.3 Review)

1. HTTP Server is listening to localhost:3000
2. Web Client requests Rotten Potatoes home page from a Web Server
   a. Web Client's browser creates HTTP request using GET method and URI: localhost:3000 to contact web server listening to same computer, same port.
   b. Web Server receives the HTTP request
3. HTTP Server obtains Rotten Potatoes app and sends this content back to Client
   a. Server returns content encoded in HTML using HTTP
   b. Client receives the content as a HTTP reply
4. Web Client displays content according to directives
5. Web Client closes its HTTP connection
6. Web Server is still listening for clients.







URI endpoint received HTTP request
Route: Method + URI (base URI + path to resource) ***
HTTP methods: GET POST PUT PATCH DELETE
GETs should have no side-effects

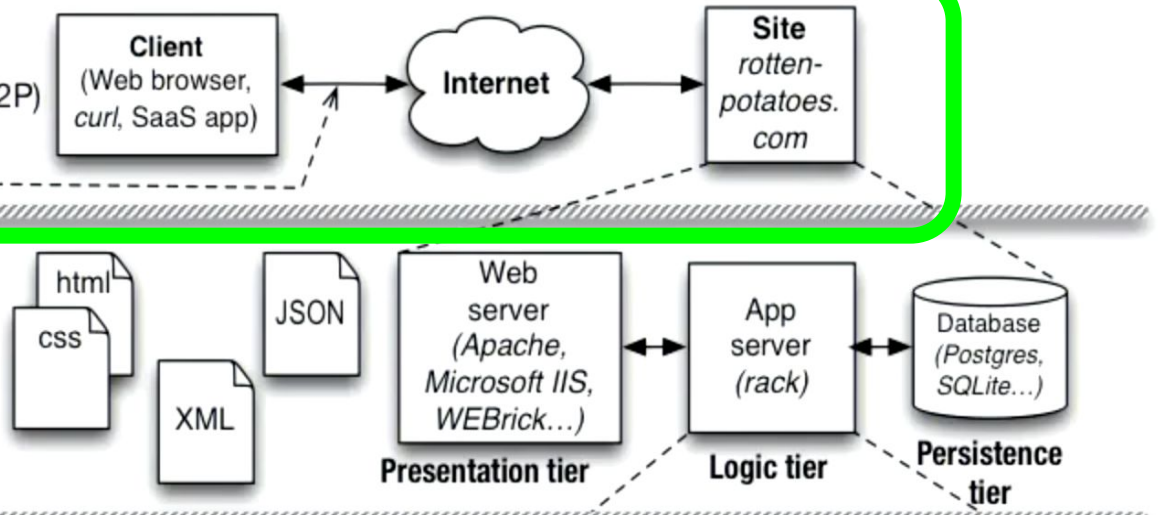
For GET method: arguments passed in URI query
For POST method: args passed in body (why?)

§2.1 100,000 feet
- Client-server (vs. P2P)

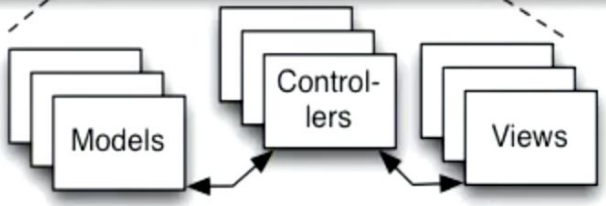
§2.2 50,000 feet
- HTTP & URIs

§2.3 10,000 feet
- HTML & CSS, JSON, XML

§2.4 5,000 feet
- 3-tier architecture
- Horizontal scaling



§2.5 1,000 feet—Model-View-Controller (vs. Page Controller, Front Controller…)



§2.6 500 feet: Active Record models (vs. Data Mapper)
- **Active Record**
- Data Mapper

§2.7 500 feet: RESTful routes & controllers for self-contained actions)
- **REST**

§2.8 500 feet: Template View (vs. Transform View)
- **Template View**
- Transform View

Service Oriented Architecture

SOA Benefit & Cost Analysis

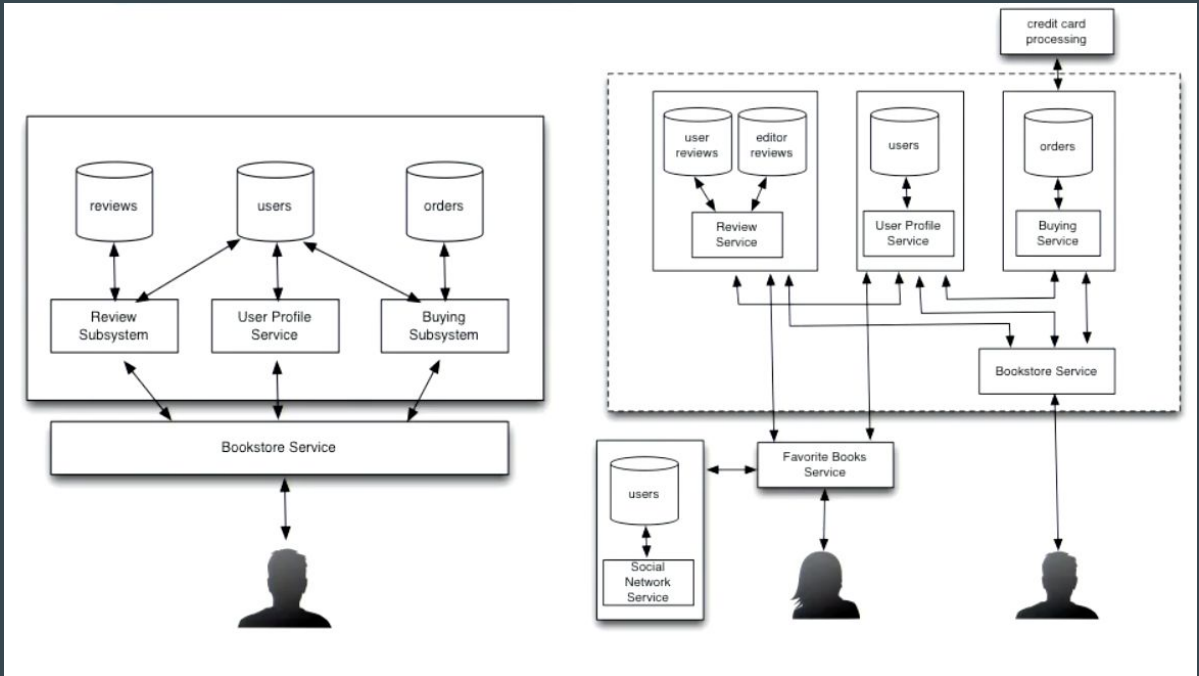
Reusable: can combine existing services where each microservice is implemented with their own language/framework (implementation hidden behind API)	**Performance:** for each microservice, need to dig into the software stack of a network interface due to layers of APIs, which can cause performance penalty: State is split across services
Easy testing: each microservice does only one thing, testing that one thing in isolation is easier Allows developers to use best tool for job for each microservice	**Dependability:** Partial Failure is possible in SOA since microservices can fail independently from others, so it is not just a measure of "app is working" vs "app isn't working".In cases like this testing can be harder (integration and system tests, validation tests)
Agile Friendly: Agile works best with small-medium teams, since SOA allows small teams to develop microservices and then combining them.	**Development Work:** each microservice has to build their own interface instead of a single monster interface for the entire application, REST simplifies this each microservice essentially being end-to-end services that can stand on their own
Same team is responsible for developing, testing and operating the microservice so customer-feedback cycles can be done quickly and efficiently	**Dev/Ops:** developers need to know about operations to be able to manage functionality and performance. If "you build you run" then you need to know ops deeper

Question 3

What are the benefits of using SOA (Service Oriented Architecture)?

SOA: Service Oriented Architecture

- set of independent composable services that can be combined as black boxes

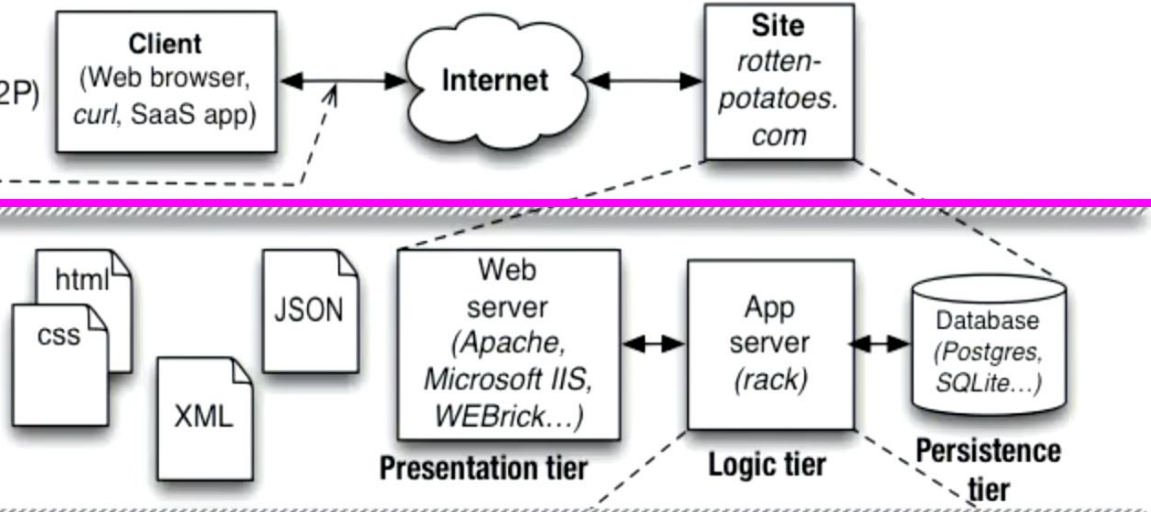


§2.1 100,000 feet
- Client-server (vs. P2P)

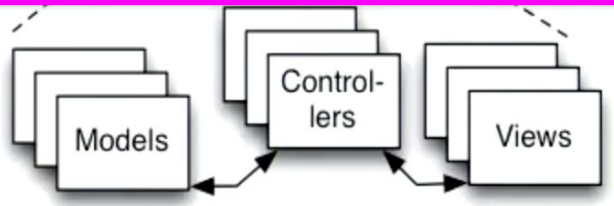
§2.2 50,000 feet
- HTTP & URIs

§2.3 10,000 feet
- HTML & CSS, JSON, XML

§2.4 5,000 feet
- 3-tier architecture
- Horizontal scaling



§2.5 1,000 feet—Model-View-Controller (vs. Page Controller, Front Controller…)

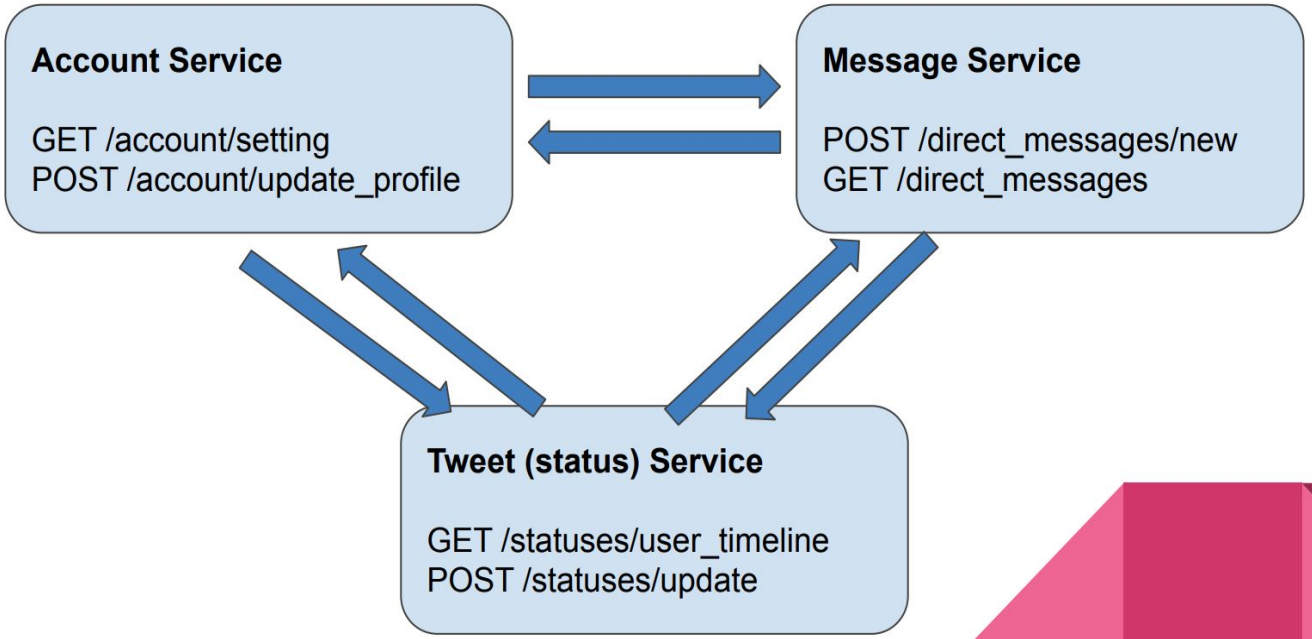


§2.6 500 feet: Active Record models (vs. Data Mapper)
- **Active Record**
- Data Mapper

§2.7 500 feet: RESTful routes & controllers for self-contained actions)
- **REST**

§2.8 500 feet: Template View (vs. Transform View)
- **Template View**
- Transform View

Twitter in SOA example

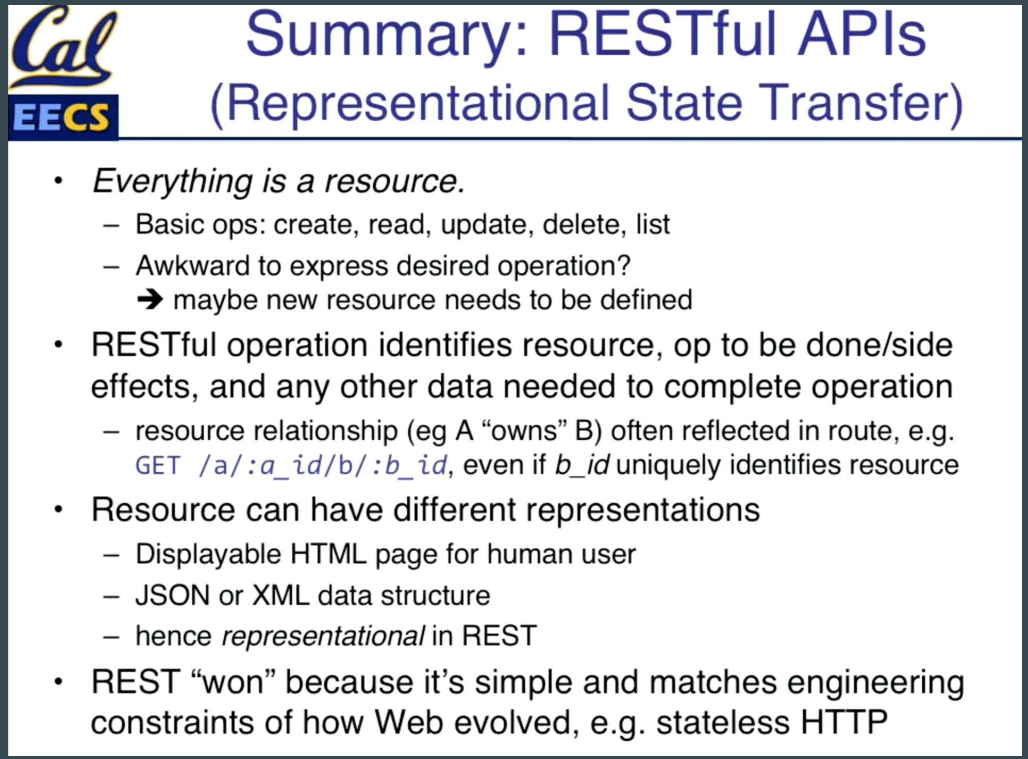


APIs

A set of well-defined methods for interacting with the data of a software system. Defines what kind of requests are acceptable to access their servers and what kind of requests should be expected

API documents show info about:

- identify function to be called
- pass arguments (variable names, formatting)
- how to receive return value (return types)
- errors, exceptions

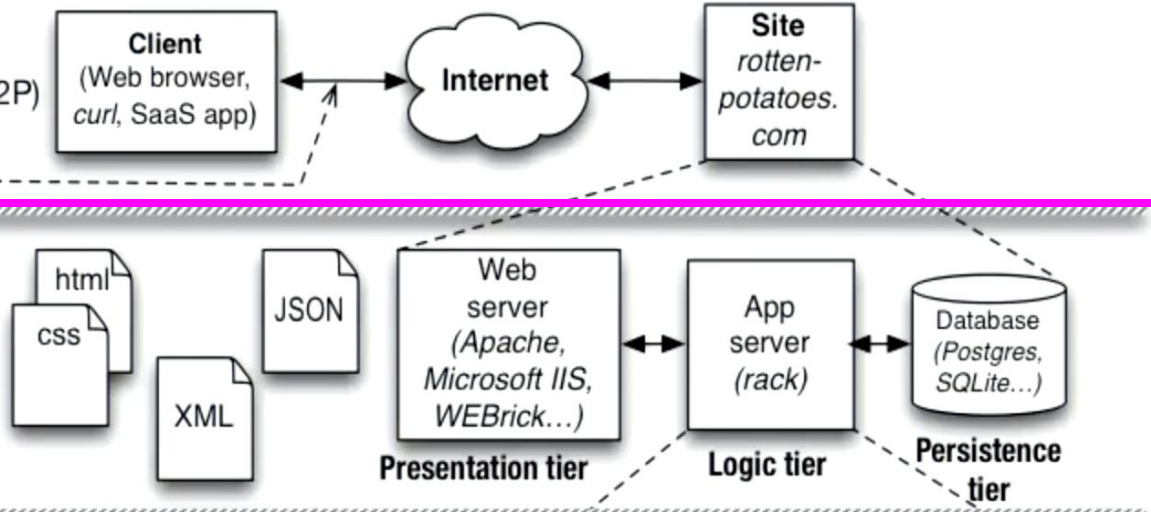


§2.1 100,000 feet
- Client-server (vs. P2P)

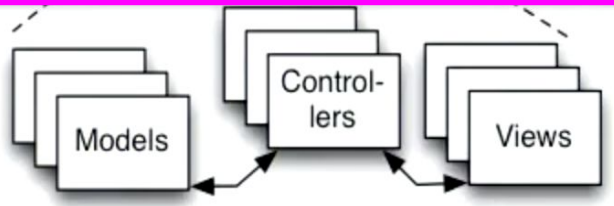
§2.2 50,000 feet
- HTTP & URIs

§2.3 10,000 feet
- HTML & CSS, JSON, XML

§2.4 5,000 feet
- 3-tier architecture
- Horizontal scaling



§2.5 1,000 feet—Model-View-Controller (vs. Page Controller, Front Controller…)



§2.6 500 feet: Active Record models (vs. Data Mapper)
- **Active Record**
- Data Mapper

§2.7 500 feet: RESTful routes & controllers for self-contained actions)
- **REST**

§2.8 500 feet: Template View (vs. Transform View)
- **Template View**
- Transform View

REST

...

Question 4

Which of the following are RESTful routes?

Order confirmation: GET /confirm

Welcome Page: GET /user/mchou/welcome

Return Item 606: POST /user/mchou/return/606

Add item 301: POST /add/301

REST

- Identify all entities manipulated by apps as **resources**.
- Provide a limited set of **operations** to be performed on a resource.
- Design the routes so that any HTTP request would contain all info necessary to identify both **a particular resource** and the **action** to be performed on it.
- REST is an organizing principle for self-contained requests and resources where various operations can be performed.
- Each controller action describes a single self-contained operation on one of the app's resources.
- Apps designed in accordance with this are said to expose RESTful APIs and the URIs that map to particular actions are said to be RESTful URI
- Rails doesn't mandate RESTful routes but it assumes REST by default.
- **Restfulness imply Service-Oriented Architecture -> WHY?**

RESTful Routes vs Non-RESTful Routes

	Non-RESTful site URI	RESTful site URI
Login to site	POST /login/dave	POST /login/dave
Welcome page	GET /welcome	GET /user/301/welcome
Add item ID 427 to cart	POST /add/427	POST /user/301/add/427
View cart	GET / cart	GET /user/301/cart
Checkout	POST /checkout	POST /user/301/checkout

- Non-Restful routes and requests rely on results of the previous requests
- RESTful clients can immediately go to line 3 but Non-Restful Clients have to perform other lines first to set up the implicit information that line 3 depends.
- RESTful routes relies on no other information than that is sent with the HTTP request: the resource and action is clearly defined regardless of previous requests.

Restful Routes

HTTP Verb	Path	Controller#Action	Used for
GET	/photos	photos#index	display a list of all photos
GET	/photos/new	photos#new	return an HTML form for creating a new photo
POST	/photos	photos#create	create a new photo
GET	/photos/:id	photos#show	display a specific photo
GET	/photos/:id/edit	photos#edit	return an HTML form for editing a photo
PATCH/PUT	/photos/:id	photos#update	update a specific photo
DELETE	/photos/:id	photos#destroy	delete a specific photo

Attendance

https://tinyurl.com/discussion-week-3

Passcode: CLIENT_SERVER

Module 3 Exercises:

SOA, Restful APIs & HTTP

Design RESTful API

Goal: Build an login/authentication server

What API calls we need?

Display Login Template?

Send Credentials?

Need to go find my credentials?

Need to update my credentials?

Design RESTful API

Goal: Build an authentication server

What API calls we need?

Display Login Template? **GET /login**

Send Credentials?

Need to go find my credentials?

Need to update my credentials?

Design RESTful API

Goal: Build an authentication server

What API calls we need?

Display Login Template? **GET /login**

Send Credentials? **POST /login**

Need to go find my credentials?

Need to update my credentials?

Design RESTful API

Goal: Build an authentication server

What API calls we need?

Display Login Template? **GET /login**

Send Credentials? **POST /login**

Need to go find my credentials? **GET /login/update**

Need to update my credentials?

Design RESTful API

Goal: Build an authentication server

What API calls we need for:

Display Login Template? **GET /login**

Send Credentials? **POST /login**

Need to go find my credentials? **GET /login/update**

Need to update my credentials? **POST /login/update**

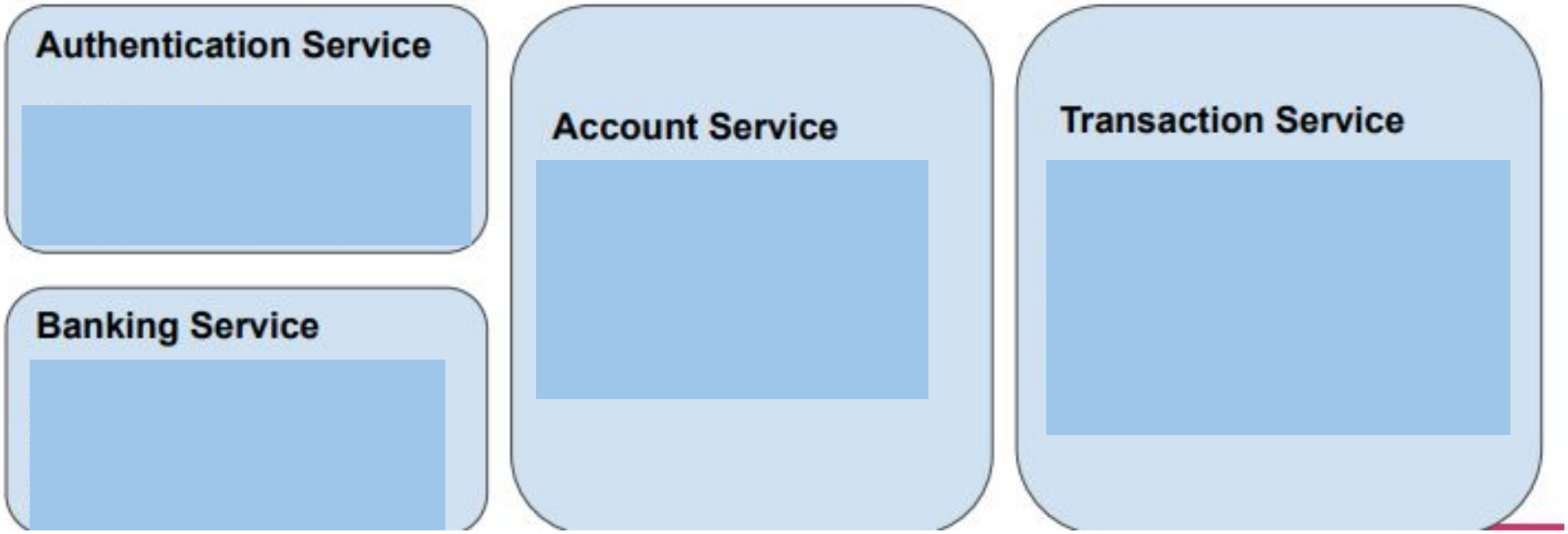
Activity: Design Venmo as SOA

Online platform where users can send money to each other.
Objectives: Design functional services and expose necessary APIs for each service.

Requirements

a. Users are able to log in and see their profiles
b. Users can transfer their balance to their checking/saving account
c. Users can make transactions (making payments or requesting payments)
d. Users can see the list of transactions
e. Anything you feel necessary. Explain why.

Example Solution



Authentication Service

GET /login
POST /login
POST /login/update

Banking Service

POST /charge_debit
POST /transfer
POST /charge_credit

Account Service

GET /account/profile
GET /account/balance
POST /account/update
GET /account/search

Transaction Service

GET /transaction/stream
GET /transaction/list
POST /transaction/send
POST /transaction/request
POST /transaction/fulfill

```ruby
require 'sinatra'

get '/login' do
  get_login_template()
end

post '/login' do
  attempt_login(params[:password], params[:username])
end

post '/login/update' do
  edit_login(params[:password], params[:password_confirmation], params[:username])
end

post '/charge_debit' do
  user = getUserById(params[:userId])
  charge_debit(user, params[:amount])
end

post '/transfer' do
  user = getUserById(params[:userId])
  transfer_to_debit(user, params[:amount])
end

post '/charge_credit' do
  user = getUserById(params[:userId])
  charge_credit(user, params[:amount])
end

```

2 API Implementation

Finish the following ruby implementation for a simple todo API. Assume you have access to a params hash with any necessary query parameters, along with a format_as_json(object) function. Return results as JSON when applicable.

```ruby
    class User
      attr_reader :id
      attr_accessor :todo

      def initialize
        @id = app.get_new_user_id
        @todo = Todo.new
      end
    end

class Todo
  def initialize
    @items = Array.new
  end

  def add item
    if not @items.include? item
      @items << item
    end
  end

  def delete item
    @items.delete item
  end

  def view_item item_id
    # assume each item instance has an accessible id field,
    # and all items/ids are unique
    item = _____
    format_as_json item unless item.nil?
  end
end
```

```ruby
get '/user/:id/todo' do
  user = get_user_by_id params[:id]
  _____
  _____
end

get '/user/:id/todo/:item_id' do
  user = get_user_by_id params[:id]
  _____
  _____
end

post '/user/:id/todo' do
  user = get_user_by_id params[:id]
  request.body.rewind
  raw_item = JSON.parse request.body.read
  _____
  _____
end

delete '/user/:id/todo/:item' do
  user = get_user_by_id params[:id]
  _____
  _____
end
```

2 API Implementation

Finish the following ruby implementation for a simple todo API. Assume you have access to a params hash with any necessary query parameters, along with a format_as_json(object) function. Return results as JSON when applicable.

```ruby
    class User
      attr_reader :id
      attr_accessor :todo

      def initialize
        @id = app.get_new_user_id
        @todo = Todo.new
      end
    end

class Todo
  def initialize
    @items = Array.new
  end

  def add item
    if not @items.include? item
      @items << item
    end
  end

  def delete item
    @items.delete item
  end

   def view_item item_id
     # assume each item instance has an accessible id field,
     # and all items/ids are unique
     item = @item.find {|i| i.id = item_id } # SOLUTION
     format_as_json item unless item.nil?
   end
end
```

```ruby
get '/user/:id/todo' do
  user = get_user_by_id params[:id]
  user.todo.to_json # SOLUTION
end

get '/user/:id/todo/:item_id' do
  user = get_user_by_id params[:id]
  todo = user.todo # SOLUTION
  todo.view_item params[:item_id] # SOLUTION
end

  post '/user/:id/todo' do
    user = get_user_by_id params[:id]
    request.body.rewind
    raw_item = JSON.parse request.body.read
    user.todo.add raw_item # SOLUTION
    201 # SOLUTION (Return 201 to indicate resource created)
  end

  delete '/user/:id/todo/:item' do
    user = get_user_by_id params[:id]
    user.todo.delete params[:item] # SOLUTION
    200 # SOLUTION (Return 200 to indicate resource deleted)
  end
```

1.2 API Documentation

Read through GitHub's API documentation, which can be found at https://developer.github.com/v3. Find answers to the following questions.

- How can I get publicly available information about a user?
- How can I list all pull requests of a repository?
- How can I get all closed pull requests of a repository?
- How can I create a new pull request?
- Which input values are required to create a new pull request?

- How can I get publicly available information about a user?
  Github Documentation: https://developer.github.com/v3/users/get-a-single-user
  Example Request: curl https://api.github.com/users/cs169
  Behavior: Should grab the information of the user with the name "cs169".
- How can I list all pull requests of a repository?
  Github Documentation: https://developer.github.com/v3/pulls/list-pull-requests
  Example Request: curl https://api.github.com/repose/rails/rails/pulls
  Behavior: Should list all PRs of the "rails" repo owned by the "rails" organization/account
- How can I get all closed pull requests of a repository?
  Github Documentation: https://developer.github.com/v3/pulls/list-pull-requests
  Example Request: curl https://api.github.com/repos/rails/rails/pulls?state=closed
  Behavior: Should list all closed PRs of "rails" repo owned by the "rails" organization/account.
- How can I create a new pull request?

  Github Documentation: https://developer.github.com/v3/pulls/create-a-pull-request
  Example Request: This request is different from the previous ones in that it is a POST, not GET request (since we're trying to create something). Therefore, we must modify the curl request by 1. Changing the request type to POST and 2. Sending the needed information as arguments in the URL or as a JSON (depends on the API). In this case, it might look like:
  curl -X POST https://api.github.com/repos/:owner/:repo/pulls
- Which input values are required to create a new pull request?
  From the previous answer's Github Documentation hyperlink, it looks like the required parameters are: title, head, and base

Questions?

Attendance

https://tinyurl.com/discussion-week-3