Introduction to Cloud Applications (I)

Lecture 3: REST, End-to-End(I), PaaS

Contents

Contents

- Introduction:
  - Course updates.
  - ~~Answering Ansys specific questions (2).~~

- REST (I):
  - Concepts.
  - Resources.
  - ”Path pattern.”

- End-to-End Example including UI and Jupyter Notebook.

- ~~PaaS:~~
  - ~~Concepts.~~
  - ~~Elastic Beanstalk.~~
  - ~~Relational Data Service (Database-as-a-Service).~~

Course Updates
Ansys Specific Questions

Course Updates

- URLs:
  - New SSO AWS URL: https://ansys-na.awsapps.com/start#/
  - Recording of setting up EC2 and MySQL: https://web.microsoftstream.com/channel/ef107fdf-43a8-43bb-a6e8-ca22a51c6455
  - Recording of setting up and connecting to applications on EC2. (TBD)

- Reminders: ***Shut down your resources when not using them!***

- ”Assignment 1:”
  - I am going to write up an optional “assignment.”
  - Give you a chance to “solo” on some of the concepts.

Ansys Specific Questions

- Going to skip for today due to time.
- I will set up a general, perhaps special lecture, to cover topics.

REST (I)

Concepts

REST (https://restfulapi.net/)

What is REST

- REST is acronym for **RE**presentational **S**tate **T**ransfer. It is architectural style for **distributed hypermedia systems** and was first presented by Roy Fielding in 2000 in his famous dissertation.
- Like any other architectural style, REST also does have it's own 6 guiding constraints which must be satisfied if an interface needs to be referred as **RESTful**. These principles are listed below.

Guiding Principles of REST

- **Client–server** – By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.
- **Stateless** – Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.
- **Cacheable** – Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.
- **Uniform interface** – By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components. REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state.
- **Layered system** – The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot "see" beyond the immediate layer with which they are interacting.
- **Code on demand (optional)** – REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented.

Resources

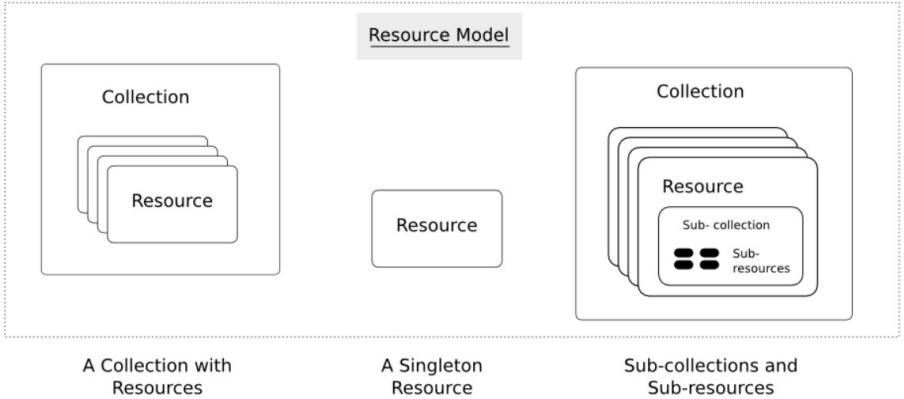
Resources are an abstraction. The application maps to create things and actions.

"A resource-oriented API is generally modeled as a resource hierarchy, where each node is either a *simple resource* or a *collection resource*. For convenience, they are often called a resource and a collection, respectively.

- A collection contains a list of resources of **the same type**. For example, a user has a collection of contacts.
- A resource has some state and zero or more sub-resources. Each sub-resource can be either a simple resource or a collection resource.

For example, Gmail API has a collection of users, each user has a collection of messages, a collection of threads, a collection of labels, a profile resource, and several setting resources.

While there is some conceptual alignment between storage systems and REST APIs, a service with a resource-oriented API is not necessarily a database, and has enormous flexibility in how it interprets resources and methods. For example, creating a calendar event (resource) may create additional events for attendees, send email invitations to attendees, reserve conference rooms, and update video conference schedules. (Emphasis added) (https://cloud.google.com/apis/design/resources#resources)



https://restful-api-design.readthedocs.io/en/latest/resources.html

REST – Resource Oriented

- When writing applications, we are used to writing functions or methods:
  - openAccount(last_name, first_name, tax_payer_id)
  - account.deposit(deposit_amount)
  - account.close()

  We can create and implement whatever functions we need.
- REST only allows four methods:
  - POST: Create a resource
  - GET: Retrieve a resource
  - PUT: Update a resource
  - DELETE: Delete a resource

  That's it. That's all you get.

"The key characteristic of a resource-oriented API is that it emphasizes resources (data model) over the methods performed on the resources (functionality). A typical resource-oriented API exposes a large number of resources with a small number of methods." (https://cloud.google.com/apis/design/resources)

- A REST client needs no prior knowledge about how to interact with any particular application or server beyond a generic understanding of hypermedia.

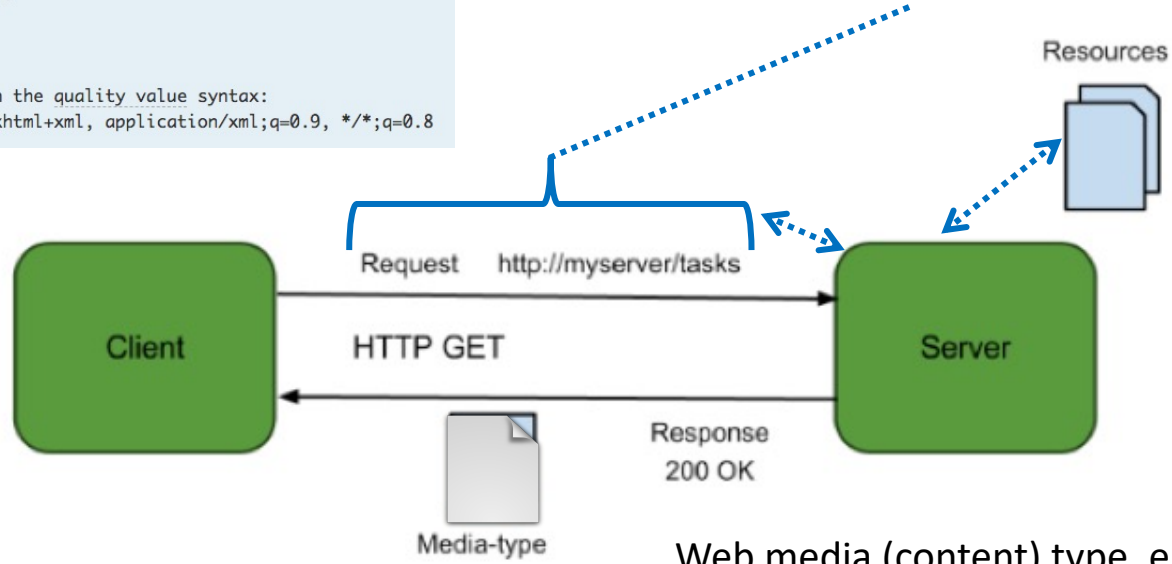
Resources, URLs, Content Types

- Relative URL identifies “resource” on the server.
- Server implementation maps abstract resource to tangible “thing,” file, DB row, … and any application logic.

Accept type in headers.

```
Accept: <MIME_type>/<MIME_subtype>
Accept: <MIME_type>/*
Accept: */*

// Multiple types, weighted with the quality value syntax:
Accept: text/html, application/xhtml+xml, application/xml;q=0.9, */*;q=0.8
```



Client may be

Browser

Mobile device

Other REST Service

… …

Web media (content) type, e.g.

- text/html
- application/json

REST Principles

- What about all of those principles?
  - Client/Server
  - Stateless
  - Cacheable
  - Uniform Interface
  - Layered System
  - Code on demand
- Some of these principles
  - Are simple and obvious.
  - Are subtle and have major implications.
  - Stateless can be baffling, but we will cover later.
- We will go into the principles in later lectures, ...

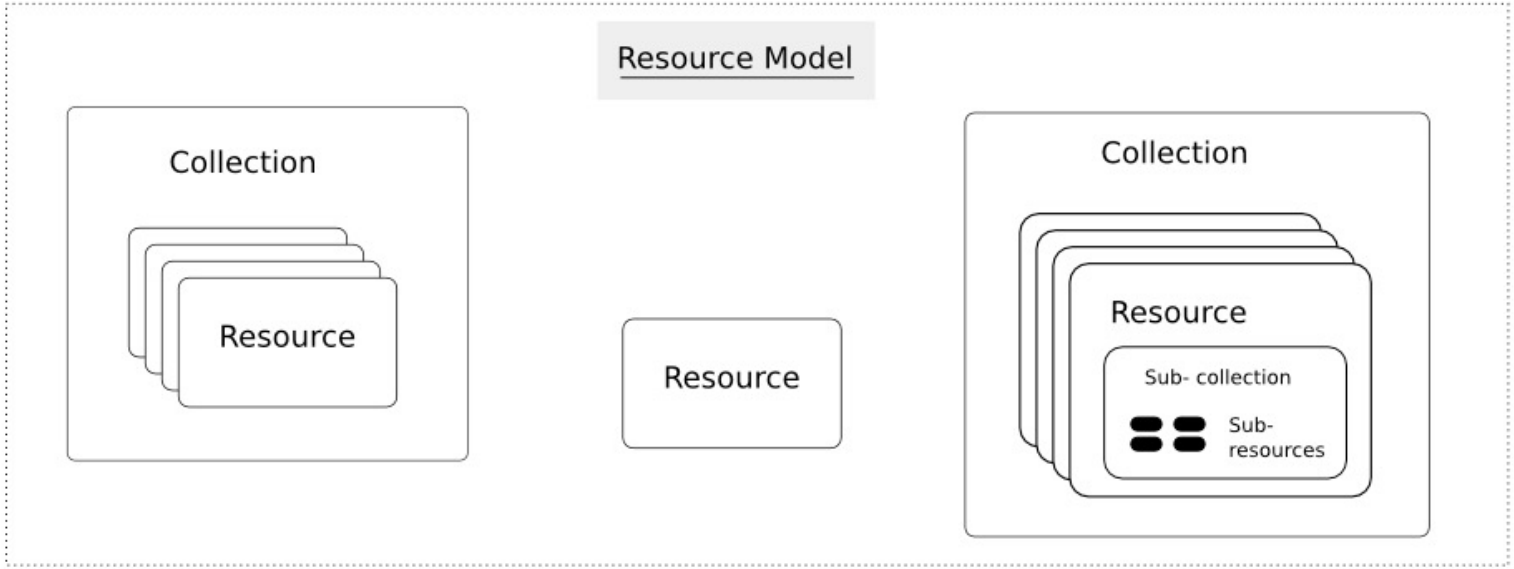
Data Models and Resources

Data Modeling Concepts and REST

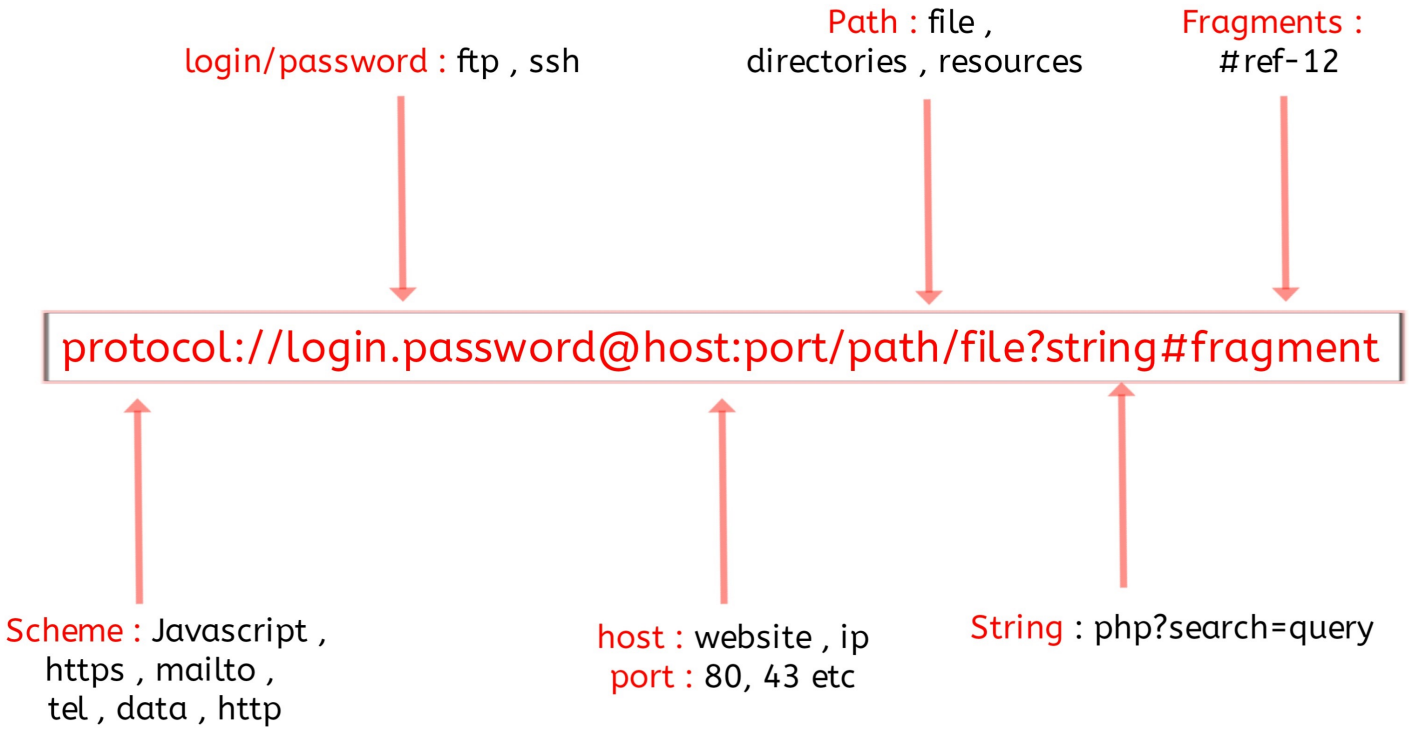
Almost any data model has the same core concepts:

- Types and instances:
  - Entity Type: A definition of a type of thing with properties and relationships.
  - Entity Instance: A specific instantiation of the Entity Type
  - Entity Set Instance: An Entity Type that:
    - Has properties and relationships like any entity, but ...
    - Has at least one *special relationship* – ***contains.***
- Operations, minimally CRUD, that manipulate entity types and instances:
  - Create
  - Retrieve
  - Update
  - Delete
  - Reference/Identify/... ...

REST and Resources



URLs



URL Mappings

- Some URLs: This gets you to the database service (program)
  - http://127.0.0.1:5001/api
  - mysql+pymysql://dbuser:dbuserdbuser@localhost
  - mongodb://mongodb0.example.com:27017
  - neo4j://graph.example.com:7687
- You still have to get into a database within the service:
  - SQL: use lahmansbaseballdb
  - MongoDB: db.lahmansbaseballdb
  - HTTP://127.0.0.1:5001/api/lahmansbaseballdb
  - … …
- And then into things inside of things inside of things ... In the database.

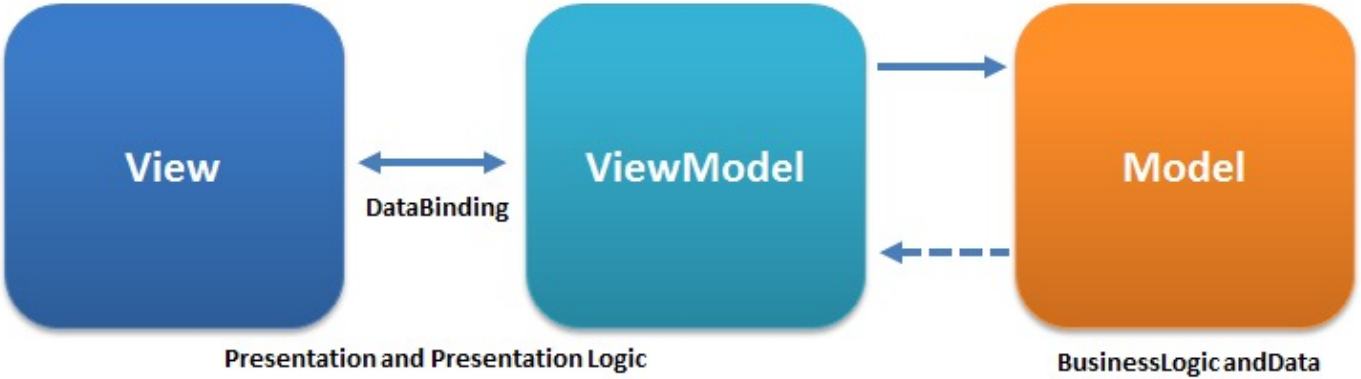
Simplistic, Conceptual Mapping (Examples)

REST Method	Resource Path	Relational Operation	DB Resource
DELETE	/people	DROP TABLE	people table
POST	/people	INSERT INTO PEOPLE (...) VALUES(...)	people table people row
GET	/people/21	SHOW KEYS FROM people ...; SELECT * FROM people WHERE playerID= 21	people row
GET	/people/21/batting	SELECT batting.* FROM people JOIN batting USING(playerID) WHERE playerID=21	batting table
GET	/people/21/batting/2004_1	SELECT batting.* FROM people JOIN batting USING(playerID) WHERE playerID=21 AND yearID=2004 AND stint=1	batting row

Show examples in notebook.

Angular Application (Part of the Big Picture)

View – Model – View Model

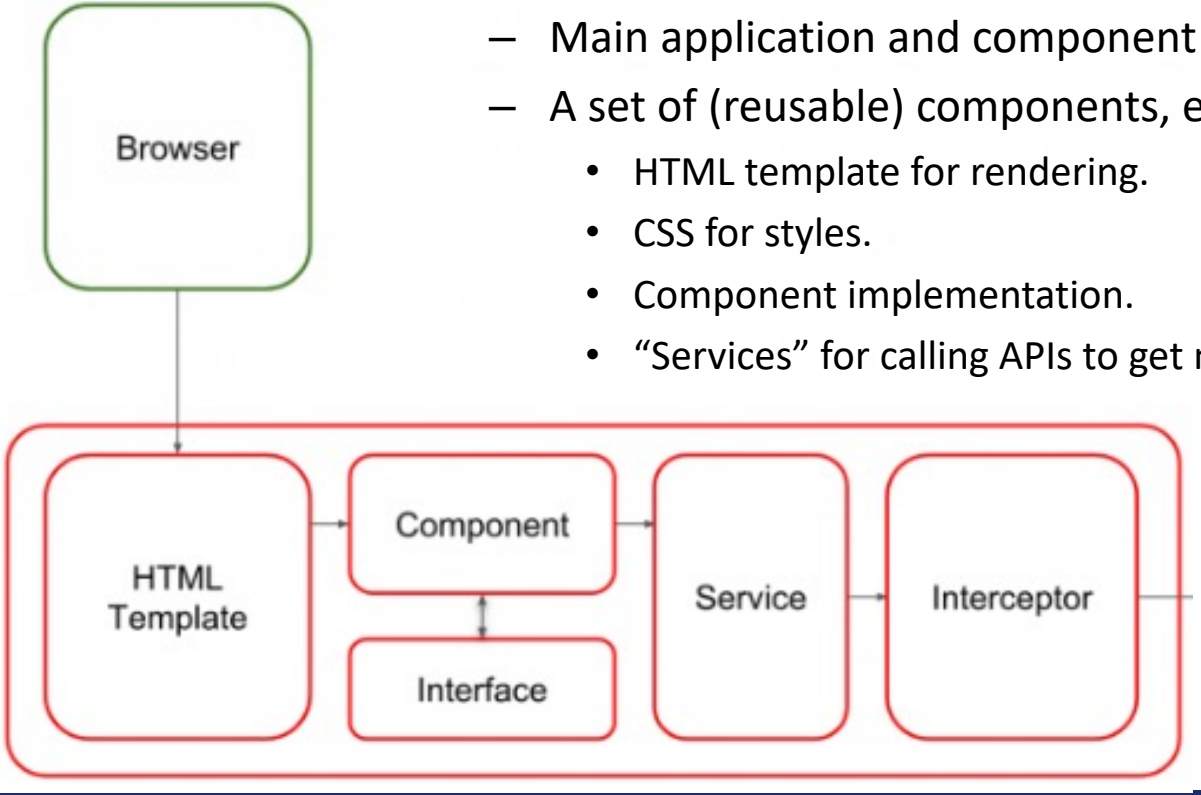


"Model–view–viewmodel (MVVM) is a software architectural pattern that facilitates the separation of the development of the graphical user interface (the view) – be it via a markup language or GUI code – from the development of the business logic or back-end logic (the model) so that the view is not dependent on any specific model platform. The viewmodel of MVVM is a value converter,[1] meaning the viewmodel is responsible for exposing (converting) the data objects from the model in such a way that objects are easily managed and presented ..."
https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel

Angular Application Introduction

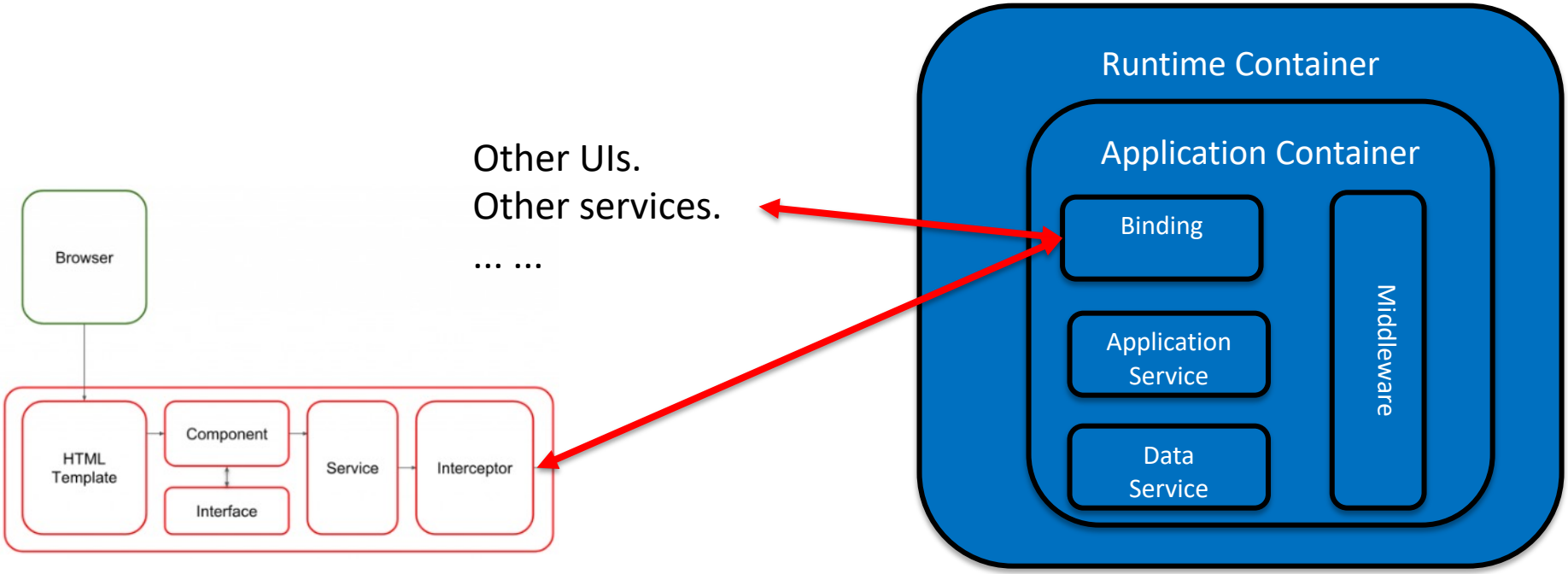
Overly simplistically, an Angular browser application contains

- Main application and component metadata.
- A set of (reusable) components, each of which is composed of:
  - HTML template for rendering.
  - CSS for styles.
  - Component implementation.
  - "Services" for calling APIs to get models.

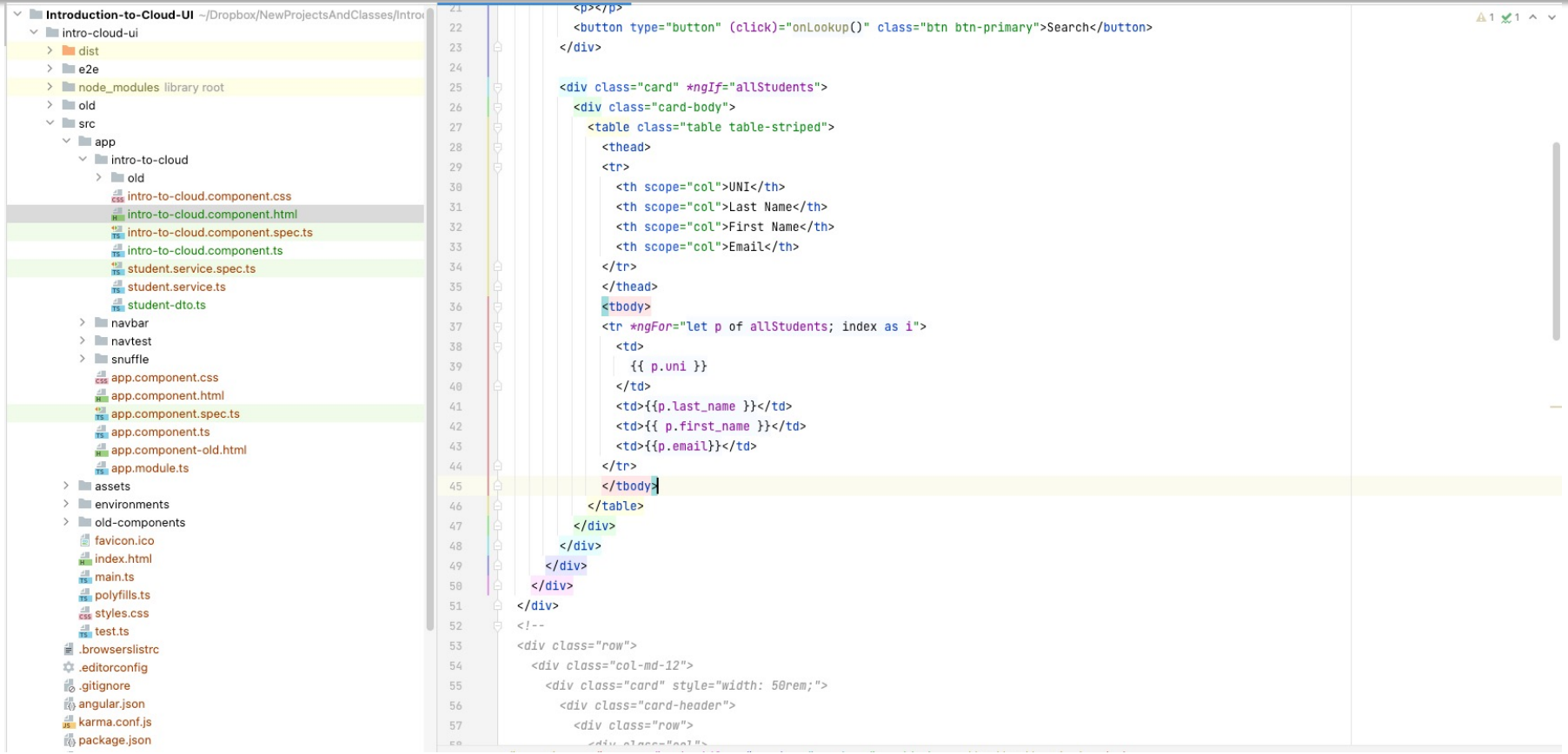


Overall Structure

- API First: All services have a well-defined API.
- ”The UI” is one of many applications that use the API.



Walkthrough



Demo