Software Architecture

Part II

What we covered last time…

- Software architecture
- System quality attributes
- Design issues and trade-offs
- System decomposition
- Service-oriented architectures (SOA)

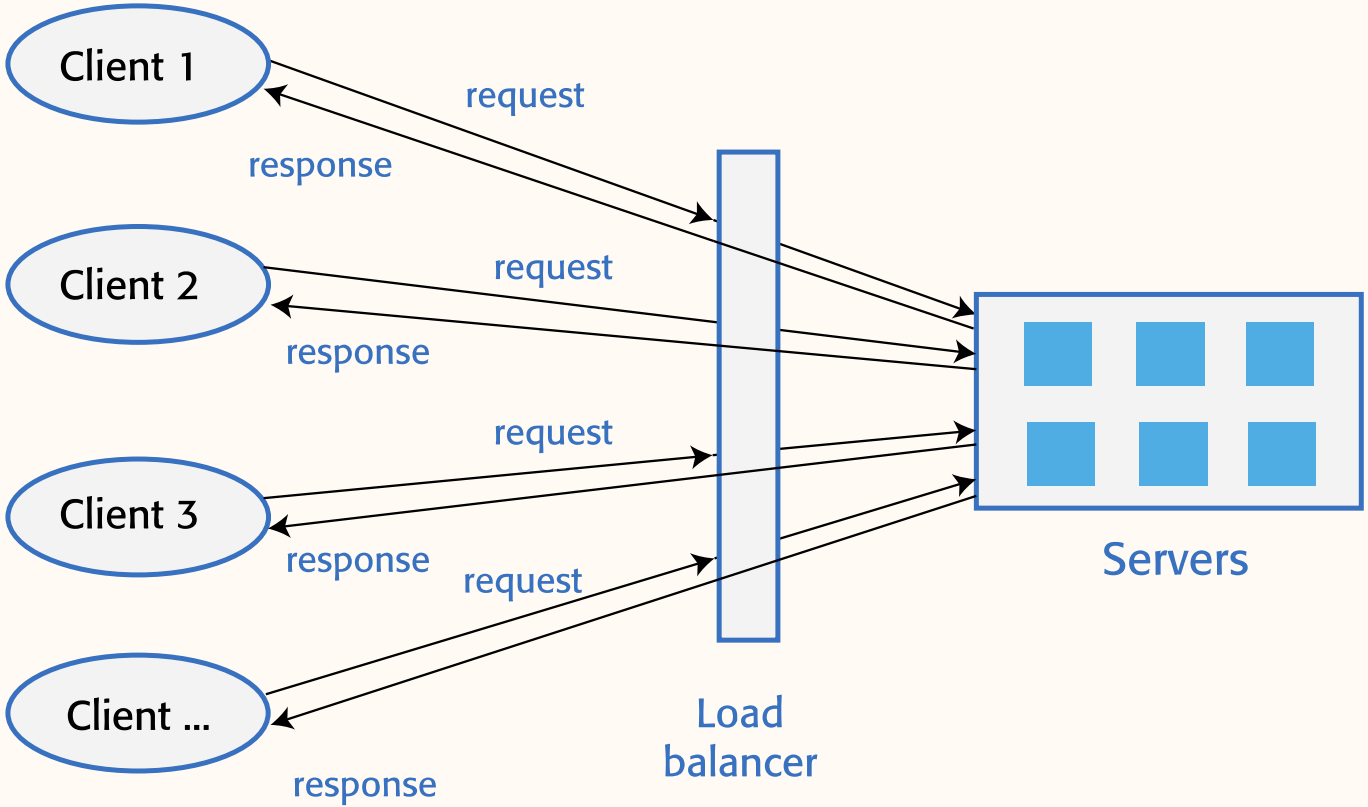
Outline for today

- Distribution architecture
- Decision-making: Choosing an architecture
- HTTP and REST
- MVC pattern
- Decision-making: Technologies

Distribution architecture

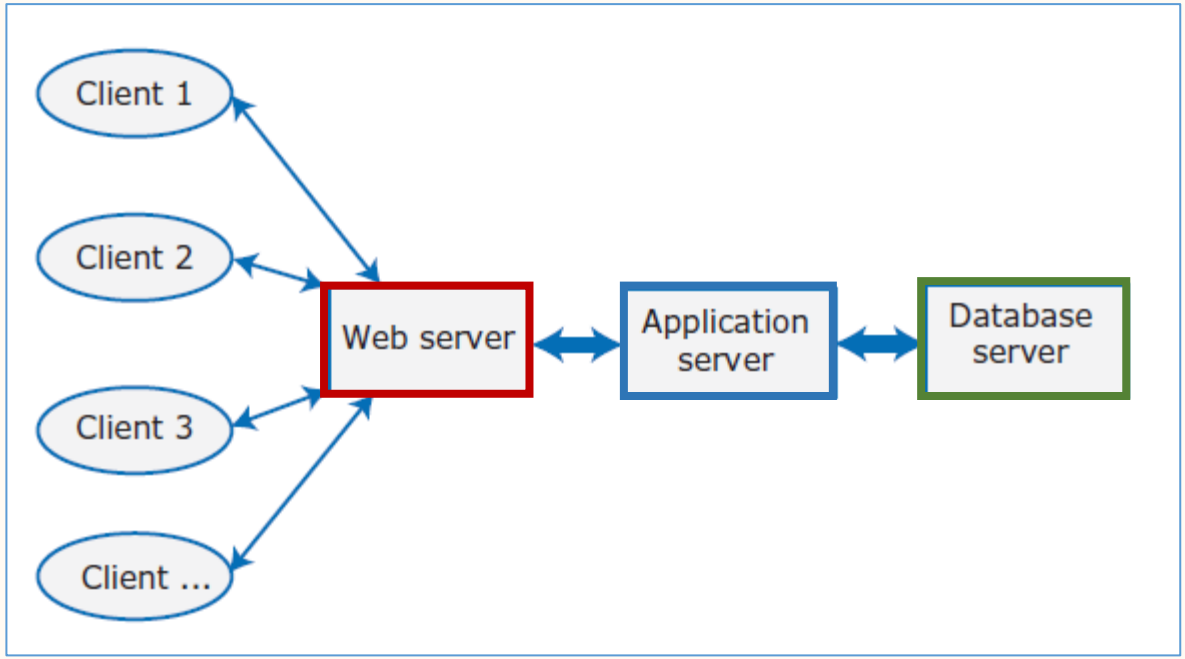
- The arrangement of servers for system
  - What software components are allocated to each
  - How servers interact to provide service to users
- Decided during architectural design process
- Majority of today's software products are web-based → **client-server** architecture
  - UI implemented on user's device
  - Functionality distributed →client & server(s)

Client-server architecture



Multi-tier client-server arch.

- Helps implement **separation of concerns**
- If there is one shared code-base, this is known as a **monolithic** architecture

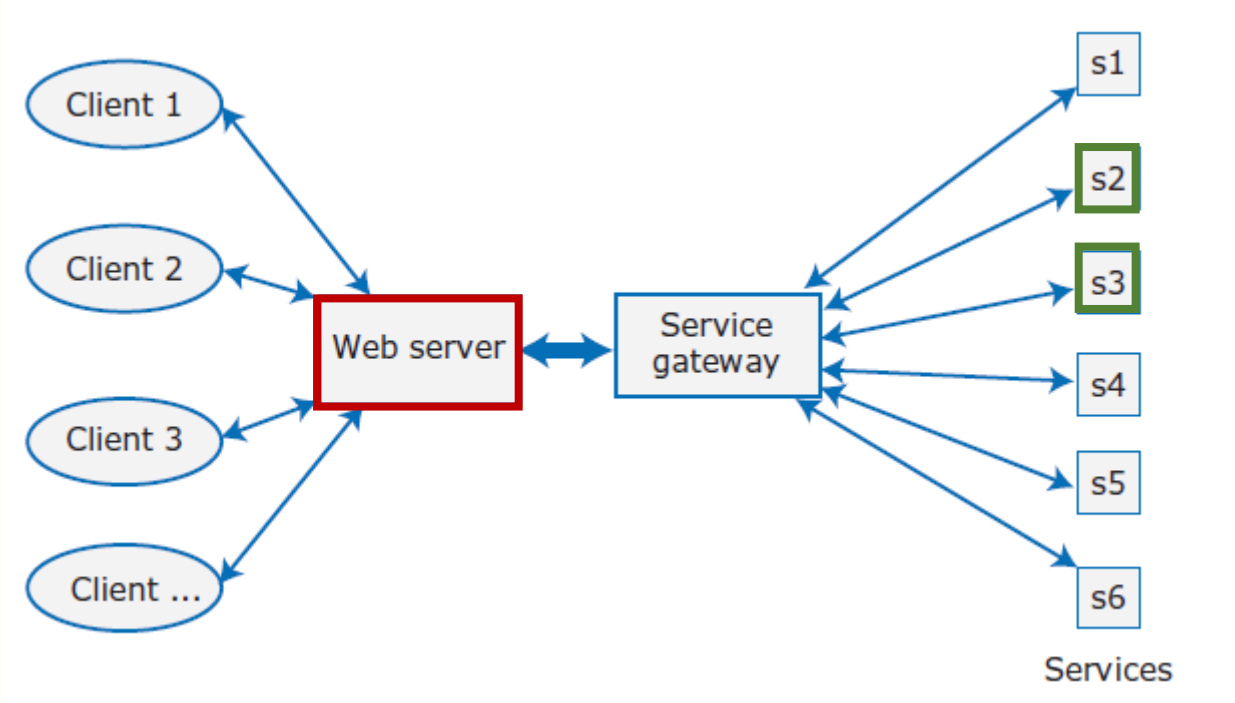


Types of servers

- **Web server**
  - Communicates with clients using HTTP protocol
  - Processes HTTP requests from the client
  - Delivers web pages to client(s) for rendering
- **Application server**
  - Responsible for application-specific operations
- **Database server**
  - Manages system data: transfers to/from database

Service-oriented architecture

- Specialized form of multi-tiered c/s arch.



Outline

- Distribution architecture
- Decision-making: Choosing an architecture
- HTTP and REST
- MVC pattern
- Decision-making: Technologies

Choosing an architecture

- What sort of architecture should you choose?
- Issues to consider:
  - Data type & data updates
  - Frequency of change
  - System execution platform
- Note: these are tradeoffs, not rules
  - May end up with conflicting advice as different issues are considered

Data type & data updates

- If using structured data updated regularly
  - Usually best: Single shared database that provides locking and transaction management
  - Not good: Data distributed across services
    - Need to keep data consistent when updated
    - Add overhead to system if not in one place
- Otherwise: data *could* be distributed

Frequency of change

- If system components change regularly
  - Usually best: Isolating components as separate services to simplify changes
  - Not good: Monolithic architecture
    - Need to redeploy entire system when any part changes
    - Requires separate services to be updated in lock-step
- Otherwise: monolithic architecture *could* make deployment, troubleshooting simpler

System execution platform

- If plan to run system on the cloud with users accessing it over the Internet
  - Usually best: Service-oriented architecture because scaling out is much simpler
- If plan to run system on local servers
  - Maybe more appropriate: Multi-tier architecture

Outline

- Distribution architecture
- Decision-making: Choosing an architecture
- HTTP and REST
- MVC pattern
- Decision-making: Technologies

Review of HTTP protocol

- Text-based request/response protocol
- Client sends **request** to server
  - **Method** (instruction) such as GET or POST
  - Identifier of a resource (**URL**/**URI**) for instruction
  - Optional additional **data**
- Server sends **response**
  - **Status** indicating success or reason for failure
  - Optional additional **data**

Structuring the additional data

- Often represented as human-readable text
- Two widely-used representations
  - XML—markup language, tags identify each item
  - JSON—represents object way JavaScript does ➔
    - More compact
    - Faster to parse/process
    - Easier for people to read

```
{
"book": [
{
“title”:   "Software Engineering",
"author": “Ian Sommerville”,
“publisher”: “Pearson Higher Education”,
“place”: “Hoboken, NJ”,
 “year”: “2015”,
"edition": “10th”,
 “ISBN”: “978-0-13-394303-0”
},
 ]
}
```

HTTP response status codes

- 1##—Informational response: server still processing
- 2##—Successful completion, expected response
- 3##—Redirection to another resource
- 4##—Client error (on client side of conversation) such as page not found, unauthorized access, …
- 5##—Server error: client request was valid but server failed to complete request

Some common status codes

- Basic success / fail codes:
  - 200 - OK
    - Success!
  - 404 - Not Found
    - File or page requested isn’t found by the server
- Server errors:
  - 500 - Internal Server Error
  - 503 - Service unavailable

Review of **API**

Application Programming Interface

- A way for programs to communicate
- "***Contract between an information provider and information user***", establishing…
  - Content required from the consumer (the call/request)
  - Content provided by the producer (the response)

REST API

- (aka RESTful API)
- API that…
  - Conforms to REST architectural constraints and
  - Allows for interaction with RESTful web services
- REST = REpresentational State Transfer
  - Server response ***transfers a representation of the state of the requested resource*** to client

Use HTTP verbs

- RESTful API's generally use HTTP
- Four major types of HTTP requests used in RESTful APIs
- These map to database CRUD operations
  - POST Create
  - GET Read
  - PUT Update
  - DELETE Delete

RESTful principles

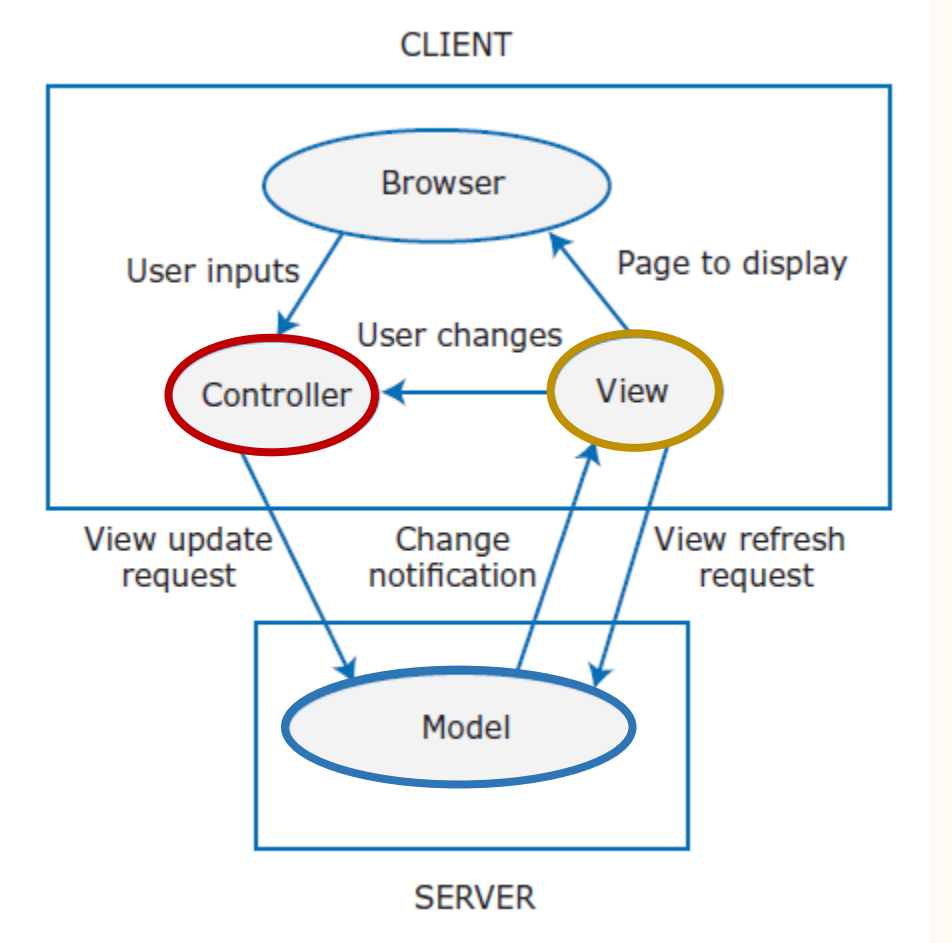
- Uniform interface
  - Consistent API request formats
  - Consistent mapping of resources to URIs
- Statelessness
  - Each request includes all info needed to process
  - Means server can’t create client “session”
- Decoupled client/server; layered architecture
- Cacheability

Outline

- Distribution architecture
- Decision-making: Choosing an architecture
- HTTP and REST
- MVC pattern
- Decision-making: Technologies

MVC Pattern

- Important design pattern for apps in general, but also for client-server apps
- Key point: client interfaces updated when data on server changes

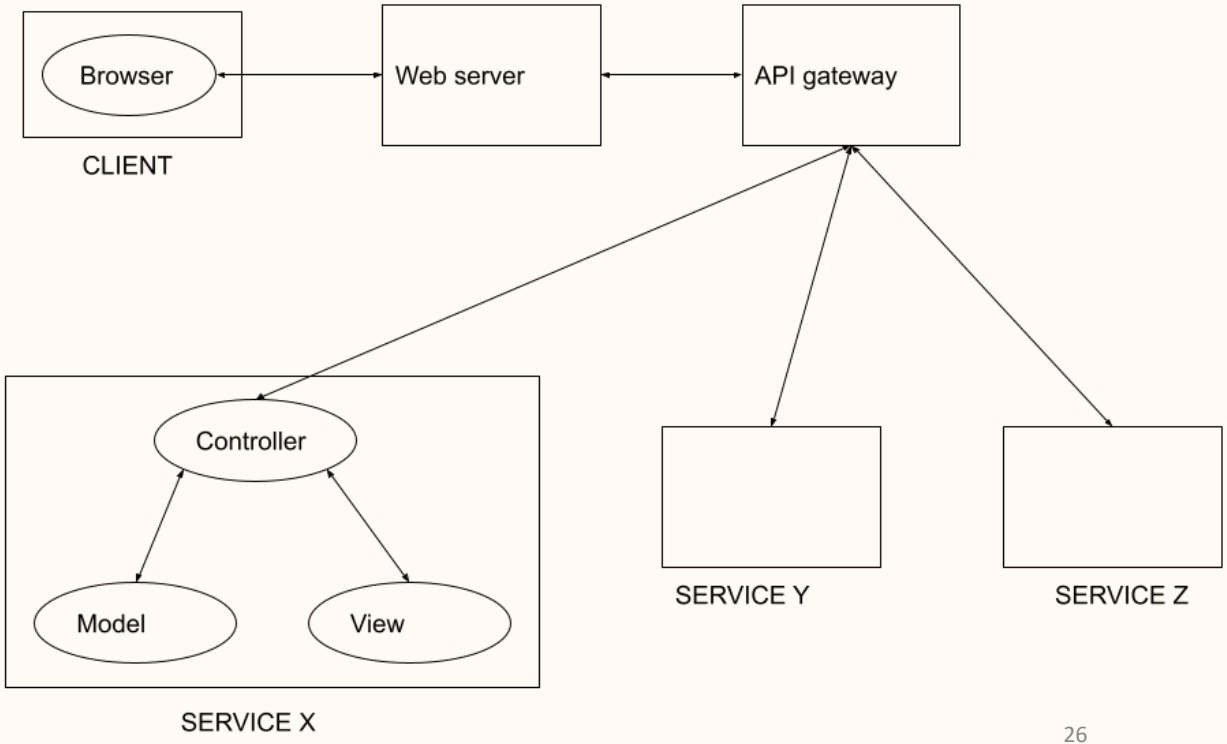


MVC Pattern

- **Model**
  - System data
  - Associated business/domain logic
    - Code/rules for how data is created, stored, & changed
- **View**
  - Each client has own view of the data
  - Includes HTML page generation, forms mgmt.
- **Controller**
  - Translates user inputs → model update requests

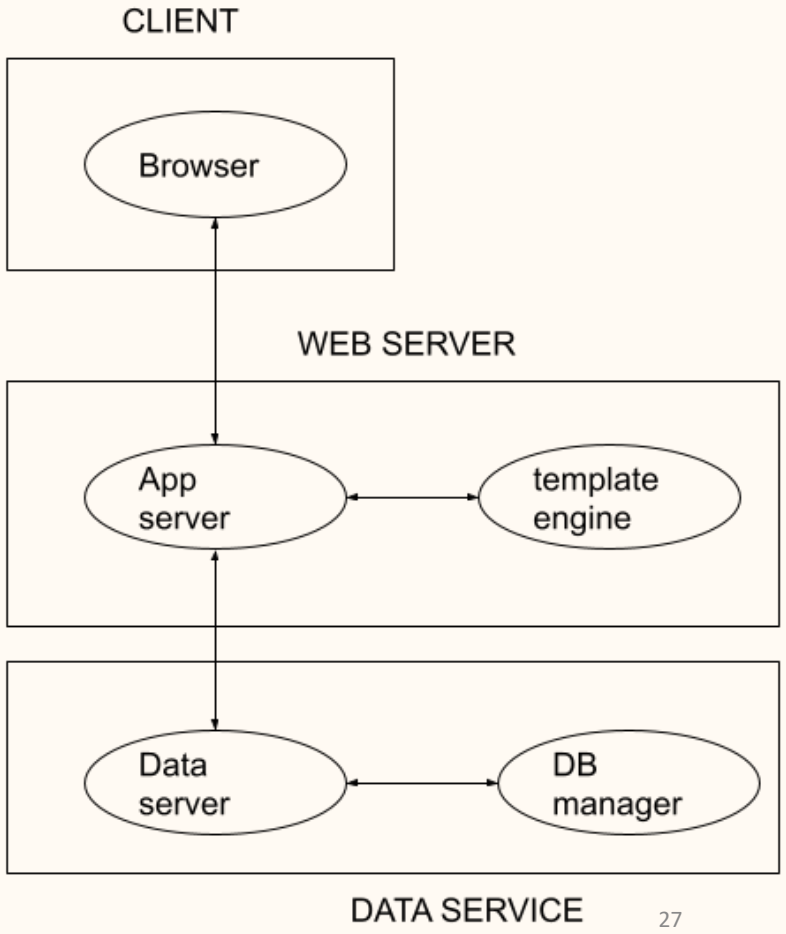
MVC-like pattern for services

- Individual services can implement MVC pattern
- Not all services need to have views



Architecture for our prototype

- Not strictly MVC or microservices
- The app server handles controller and view logic
- The data service uses a REST API
  - (i.e. it is a RESTful service)



Outline

- Distribution architecture
- Decision-making: Choosing an architecture
- HTTP and REST
- MVC pattern
- Decision-making: Technologies

Technology choice review

- Database — SQL or noSQL?
- Platform — Mobile app and/or web?
- Server — on premises or public cloud?
- Open source — to use or not to use?
- Development tools — are the tools opinionated?

Developing for the cloud

- Cloud computing is becoming ubiquitous
- Key decision to make: Design system to…
  - Run on individual servers or
  - Run on the cloud
- Possible to rent server from Azure, AWS, …
  - But does not take full advantage of the cloud

Developing on the cloud

- To develop on the cloud
  - Design architecture as a **service-oriented system**
  - Use platform APIs provided by the cloud vendor
- These allow for automatic **scalability** and system **resilience**
- More on this in the next chapter

Changing norms

- "When I wrote this book in 2018,…"
  - Distribution arch. of most business software products was multi-tier, client–server
  - User interaction implemented using MVC
- "Increasingly being updated to…"
  - Use service-oriented architectures
  - Run on public cloud platforms
  - SOA will become norm for web-based products

Summary

- Distribution architecture
- Decision-making: Choosing an architecture
- HTTP and REST
- MVC pattern
- Decision-making: Technologies