HTTP

* Hyper Text Transfer Protocol
* The main way (protocol) information sent on the internet.
* Request response based system.
  + Send an HTTP Request
  + Always receive an HTTP Response
* HTTP Request anatomy
  + URL Uniform Resource location (where the request is going)
    - URL <http://cnn.com/news>
    - URI Uniform Resource Identifier
      * /news
  + Verb
    - GET, PUT, POST, DELETE, OPTIONS…
  + Body (payload)
    - GET requests have no body
    - The body can hold data in any format but JSON is most common.
  + Version of HTTP
  + Headers
    - Key Value pairs that contain information about the request.
    - Meta-information
      * Authentication keys
      * Access Tokens
* HTTP Response Anatomy
  + Version of HTTP
  + Headers
  + Body (optional)
  + Status code
    - Numeric indicator on how a request was processed
      * 100’s information
      * 200’s successes
        + 200 generic success (OK)
        + 201 resource created.
      * 300’s redirects
      * 400’s Client errors
        + 400 bad request
        + 403 forbidden
        + 404 not found
      * 500’s Server errors
        + 500 internal server error

If you get 500’s from an API you are using that API/web server was not designed very well.

* JSON
  + JavaScript Object Notation
  + BY FAR the most popular way to send information across the web
  + FORMATTED STRING
    - A data type that and programming could process and easily parse into it’s own objects
  + {“fname”:”Adam”,”age”:19}

REST

* REpresentational State Transfer
  + The response you get back is a REPRESENTAITION of the object NOT the object itself.
    - Get back a JSON NOT a Java Object or Python Object
    - The representation could be in any format
      * 95% JSON, 4.9% XML, .1% anything else
* A type of web server architecture
  + The most popular type of web server\*
* Resource
  + Collection of objects that you want to keep track of
    - Doctors
    - Pets
    - Apartments
* 6 REST constraints that a Web Server must to be considered truly RESTful in design.
  + Uniform Interface
    - The standards by which you name the endpoints/routes/uris of the application.
      * Resources are often nested if applicable.
        + /doctors/5/appointments
    - What HTTP verbs do to each one of those endpoints.
  + GET /doctors
    - Return All doctors
  + GET /doctors/12
    - Return the doctor with the ID of 12
  + POST /doctors
    - Create a new doctor
  + PUT /doctors/14
    - Update the doctor with an ID of 14
  + DELETE /doctors/20
    - Delete the doctor with an ID of 20
* All requests to a REST API should be self-contained.
  + One HTTP request is not dependent on another for correct processing.
    - Request A does not have a direct impact on Request B’s success
    - I should only have to make a single request to update a doctor
      * NOT put /doctors/4 (edit name) a second request put/doctors/4 (edit specialty)

Javalin

* Web Servers can be written in any programing language.
* Javalin is one Java library for writing web servers.
  + There are plenty of other (Spring MVC/WEB)
* Very similar to express in Node.js
* Create a Javalin object Javalin app = Javalin.create()
  + Attach routes to that object
  + app.get(“/doctors”,DoctorController.getDoctorsHandler);
  + app.httpVerb(“route”, handler lambda);
* Event Driven program
  + Listen for an event (http request)
    - When that event occurs you execute a lambda.
* Path parameters
  + “/doctors/:id”
    - Ctx.getPathParam(“id”)
* Query Parameters
  + “/doctors?specialty=neurology”
    - Ctx.getQueryParam
* Query vs Path param
  + Query parameters are mostly a filter.
    - No result is just an empty data.
      * There could be no doctors of that specialty.
  + Path parameters are identifiers.
    - No result is a 404
      * A specific resource does not exist.

App layering

* Controllers can call ANY services that it needs.
  + Primarily call the service sharing its name
* Services can call ANY daos that it needs.
  + Primarily call the DAO sharing its name.
* DAOs should only communicate with the database concerning the object is is named for.

SQL

* Structured Query Language
  + The main programming language for relational databases
* Database
  + Persists information
    - Relational databases persist information in tables connected to each other
* Versions of SQL
  + Postgres
  + MySQL
  + OracleSQL
* Schema
  + The layout of a database
    - Names of columns and tables
  + The tables and any rules regarding those tables
    - Constraints on tables like primary keys, foreign keys, not null etc….
    - An ERD Entity Relationship Diagram
      * Visual representation of the schema
  + Anything not the data itself
* Constraints
  + Any restrictions put on a column
    - Primary Key
    - Foreign key
    - Not null
    - Check
* SQL Sub Languages
  + DDL
    - Data Definition language
    - They define the SCHEMA of the database
      * CREATE, DROP or ALTER
  + DML
    - Data Manipulation language
    - They modify data in the table.
    - UPATE, INSERT, DELETE
  + DQL
    - Data Query Language
    - How you search for information
    - SELECT
  + DCL
    - Data control Language
    - Controlling ACCESS and permission to a database
      * Creating users for a database
        + Users = developers actually querying and editing the database.
    - GRANT, REVOKE
    - In the real world no developer would have 100% complete unrestricted access to a database.
  + TCL
    - Transaction control Language
    - Commit, ROLLBACK
* Multiplicity
  + Tables have relationships to other tables.
    - One – one
      * Do not see very often.
        + Why is not just one table?
    - One – many
      * Player - team
    - Many -many
      * Player – game
      * Requires a junction table
* Primary key
  + A unique identifier for a record
  + Unique and not null
  + Usually numeric and auto generated but not a requirement.
* Foreign key
  + A column on a child table.
  + References a primary key on a parent table.
  + The foreign key is the many in a one – many relationship.

Normalization

* Process of removing redundancy in a database.
* Done to free up space.
* Transactions faster to process on highly databases.
  + Updates and insertions take longer the more columns a table has.
* 1nf
  + Primary key
  + All columns must be atomic
    - The data in that can not be broken down any more meaningfully.
    - No column should EVER contain array like information
      * Phone\_numbers BAD
* 2nf
  + 1nf
  + No functional dependencies
  + Could not calculate one columns’ values based on another column(s)
* 3nf
  + 2nf
  + No transitive dependencies.
  + The data in a column could not be found elsewhere in the database.
    - No amount a joins could get you that information
    - It would be redundant to put hometown in my player table because I could the team a player plays for and then look up the home town on that table.
* Normalization is not always better.
  + Highly normalized data is tough to query.
  + Have to perform many joins to get what you want sometimes.
* Joins
  + Combine table horizontally based on a predicate.
  + Perform some sort of matching.
  + Left join
  + Right join
  + Inner join
  + Cross join
* Set operations
  + Stack tables vertically on top of each other
    - No predicate or matching involved.
  + Union
    - Stack records even if there are duplicates.
  + Intersect
    - ONLY stacks the common records
* Order by is a clause to organize the results get back
  + Order by desc
  + Order by asc

JDBC

* Java Database Connectivity
* Java library for connecting and interacting with SQL database.
* Do you have to download the appropriate JDBC driver for your database.
  + Postgres
  + MySQL
* Key interfaces
  + Connection
  + Statement (NEVER USE)
  + PreparedStatement
  + ResultSet
* SQL Injection
  + Tricking an application into executing an unintended SQL statement
  + Usually done by passing in weirdly formatted SQL strings into a field