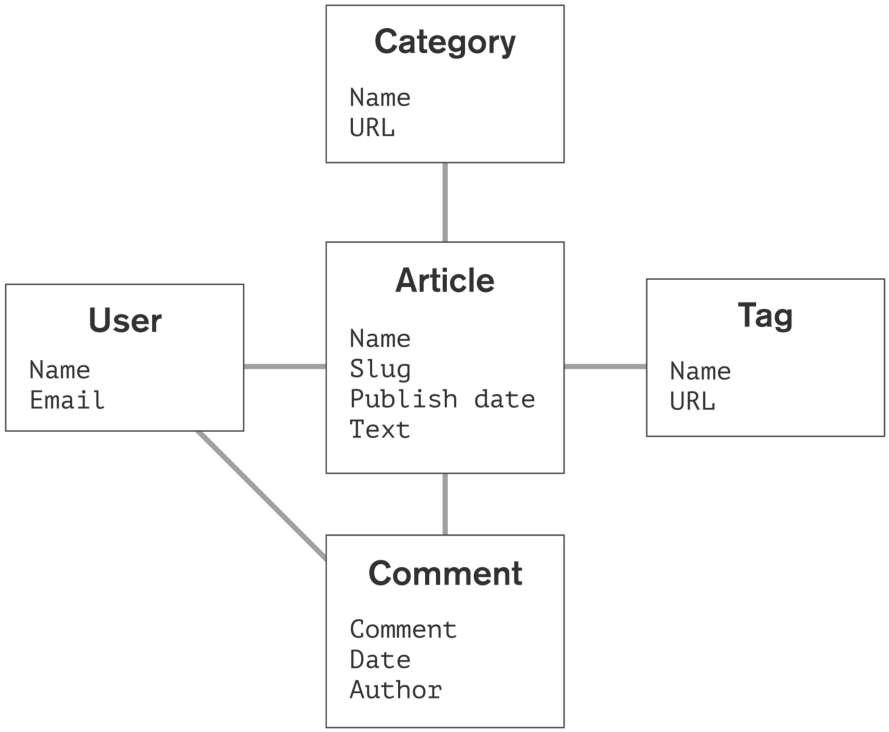
MongoDB
Web Architecture

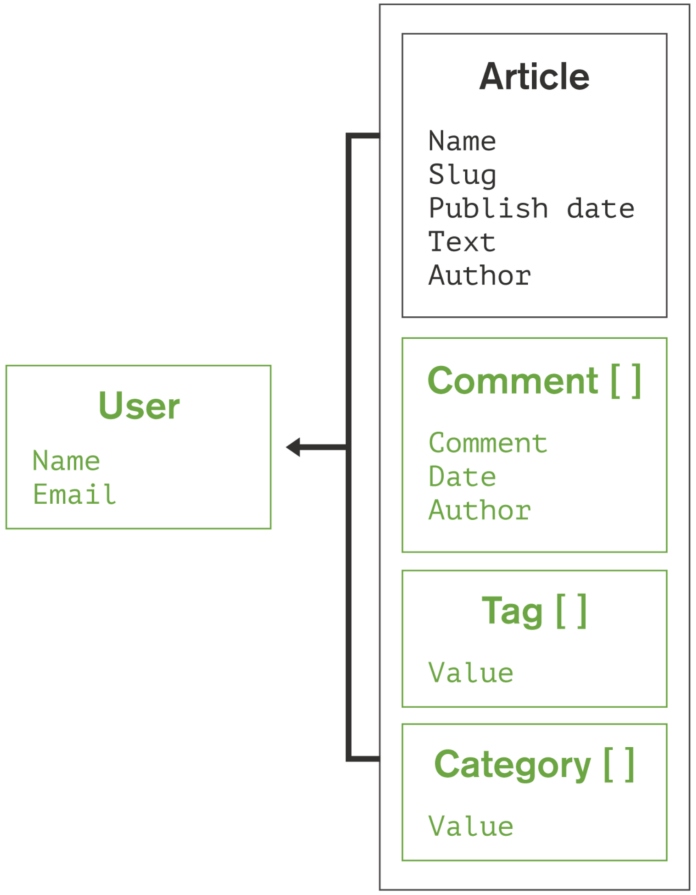
MongoDB

- MongoDB is an open-source, NoSQL database that uses a JSON-like (BSON) document-oriented model.

- Data is stored in collections (rather than tables).
  - Uses dynamic schemas rather than a predefined structure. Fields can be added/removed at will.
  - Works with many programming languages.
  - Caching: Most recent kept in RAM.
  - No transactions, but allows atomic operations.

Example





Relational data model

Data as documents

From: MongoDB_Architecture_Guide.pdf https://www.mongodb.com/json-and-bson

BSON?

- BSON is Binary JSON
- MongoDB encodes JSON documents in binary format behind the scenes
- BSON extends JSON to provide additional data types (int, long, date, floating point)

Schema Enforcement?

- JSON blobs provide flexibility but not what is always wanted
- Consider: `<h1>Hello {{person.informalName}}</h1>`
  - Good: typeof person.informalName == 'string' and length < something
  - Bad: Type is 1GB object, or undefined, or null, or ...
- Would like to enforce a schema on the data
  - Can be implemented as validators on mutating operations
- Mongoose - Object Definition Language (ODL)
  - Take familiar usage from ORMs and map it onto Mongoose
  - Effectively masks the lower level interface to MongoDB with something that is friendlier

Document validation

- Dynamic schemas provide great flexibility
- Controls to maintain data quality are also important
- MongoDB provides document validation within the database

How does it work?

Using: var mongoose = require(‘mongoose');

1. Connect to the MongoDB instance

```
mongoose.connect('mongodb://localhost/myproject');
```

2. Wait for connection to complete: Mongoose exports an EventEmitter

```
mongoose.connection.on('open', function ()  {
    // Can start processing model fetch requests
});
```

```
mongoose.connection.on('error', function (err) { });
```

Can also listen for connecting, connected, disconnecting, disconnected, etc.

Schema defines collections

String, Number, Date, Buffer, Boolean

Array - e.g. comments: [ObjectId]

ObjectId - Reference to another object

Mixed - Anything

```
var userSchema = new mongoose.Schema({
        first_name: String,
        last_name: String,
        emailAddresses: [String],
        location: String
});
```

Schema allows secondary indexes and defaults

- Simple index

```
first_name: {type: 'String', index: true}
```

- Index with unique enforcement

```
user_name: {type: 'String', index:
                {unique: true} }
```

- Defaults

```
date: {type: Date, default: Date.now }
```

Secondary Indexes

- Performance and space trade-off
  - Faster queries: Eliminate scans - database just returns the matches from the index
  - Slower mutating operations: Add, delete, update must update indexes
  - Uses more space: Need to store indexes and indexes can get bigger than the data itself
- When to use
  - Common queries spending a lot of time scanning
  - Need to enforce uniqueness

Make model from schema

- A Model in Mongoose is a constructor of objects
- A collection may or may not correspond to a model of the MVC

```
var User = mongoose.model('User', userSchema);
```

- Create objects from Model

```
User.create({ first_name: 'Amy',
              last_name: 'Pond'}, doneCallback);
function doneCallback(err, newUser) {
    assert (!err);
    console.log('Created object with ID', newUser._id);
}
```

Query collection

- Returning the entire User collection

```
User.find(function (err, users) {
     /*users is an array of objects*/ });
```

- Returning a single user object for user_id

```
User.findOne({_id: user_id}, function (err, user) { /* ... */ });
```

- Updating a user object for user_id

```
User.findOne({_id: user_id}, function (err, user) {
     // Update user object
     user.save();
});
```

Other query operations

```
var query = User.find({});
```

- Projections

```
query.select(“first_name last_name”)
     .exec(doneCallback);
```

- Sorting

```
query.sort("first_name").exec(doneCallback);
```

- Limits

```
query.limit(50).exec(doneCallback);
query.sort("-location")
```

Deleting objects from collection

- Deleting a single user with id user_id

```
User.remove({_id: user_id}, function (err) { } );
```

- Deleting all the User objects

```
User.remove({}, function (err) { } )
```

Web Architectures

- Higher level structures underlying a web ap
- Made up of elements, their properties and the relationships among elements

n-tier

- Separate the components of a web app into different tiers or layers
  - Components of a web application are presentation, processing logic, and data management
  - Benefit of separating into tiers is all the good software engineering properties

1-tier

- All three layers are on the same machine and presentation, logic and data are tightly connected.
  - Scalability: Single processor, no separation, hard to scale
  - Portability: moving to a new machine, or changing anything may mean major re-writes
  - Maintenance: Changing one layer requires changing other layers

2-tier

- Database runs on a server separate from application
- Presentation and logic layers are tightly coupled
- Can easily switch to a different database
- Coupling of presentation and logic can lead to heavy server load and network congestion.

3-tier

- Each tier is separate with a clear interface between them.
- Each tier can potentially run on a different machine

- Client-server
- Each tier is independent
- Unconnected tiers should not communicate
- Change in platform affects only that tier

Presentation layer

- Provides user interface and interactions
- Client-view or front-end
- Shouldn’t contain business logic or data access code

Logic Layer

- Set of rules for processing the data and accommodates many individual users
- No presentation or access code

Data Layer

- Physical storage for data persistence
- Manages access to the DB or file system
- Also called the back-end

- No presentation or business logic

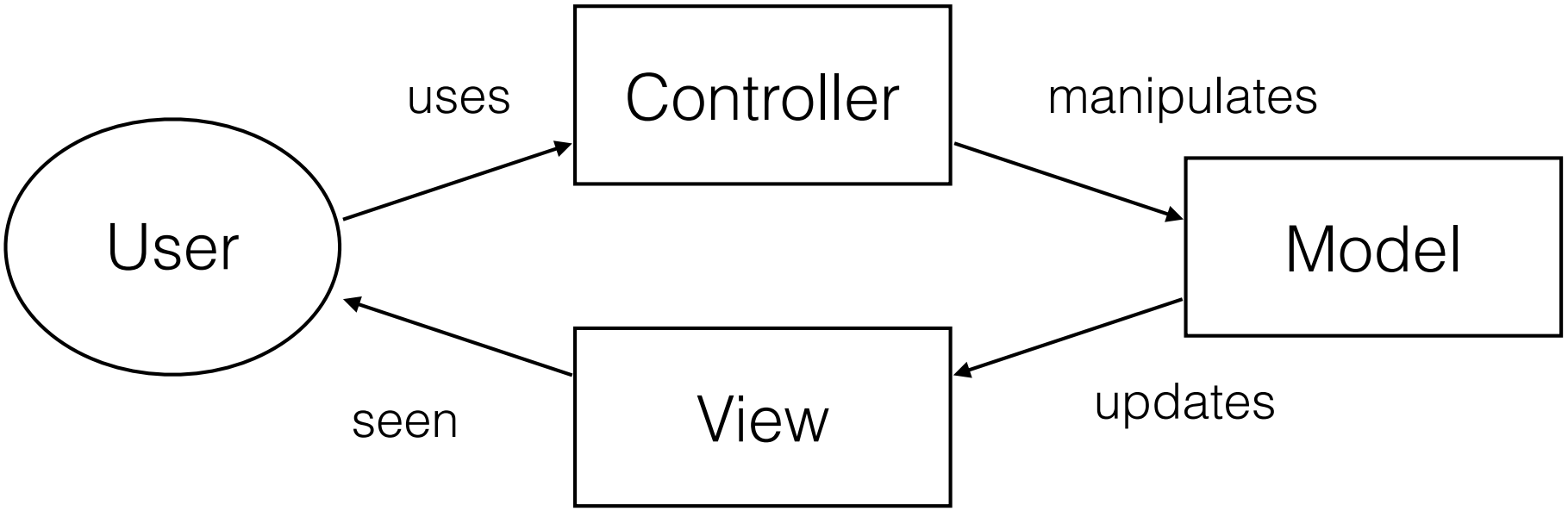
- The **presentation layer** is the static or dynamically generated content rendered by the browser (front- end).

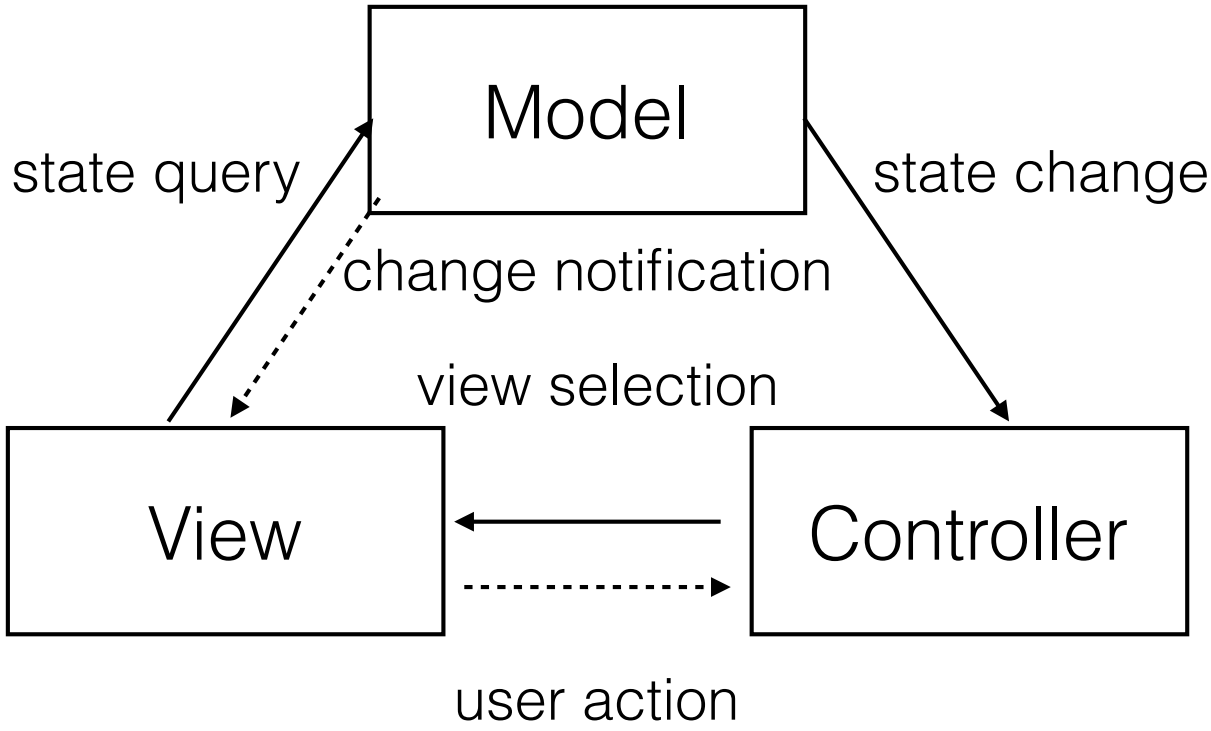
- The **logic layer** is responsible for dynamic content processing and generation, e.g. using Node.js, PHP, Java EE, ASP.NET (middleware).

- The **data layer** is the database, comprising the data sets and data management system (back-end).

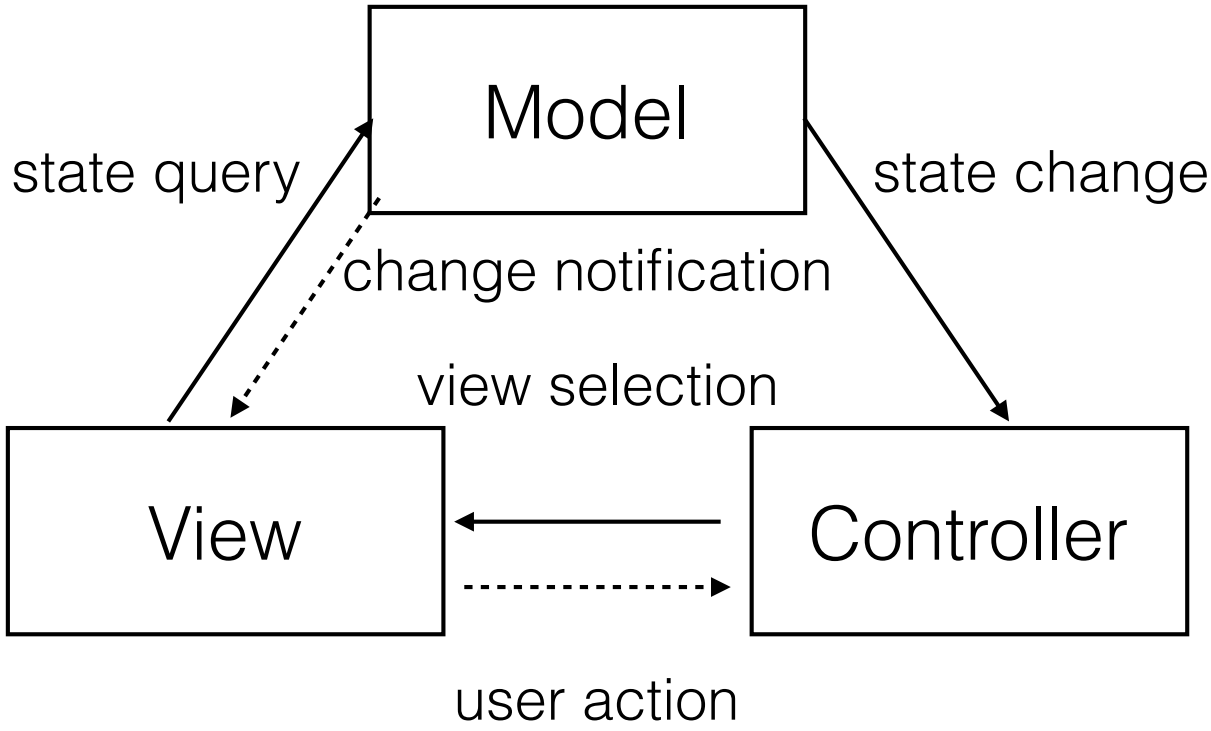
- The main benefit of 3-tier is the independence of layers, which means:
  - Easier maintenance
  - Reusable components
  - Faster development (division of work)

Model View Controller

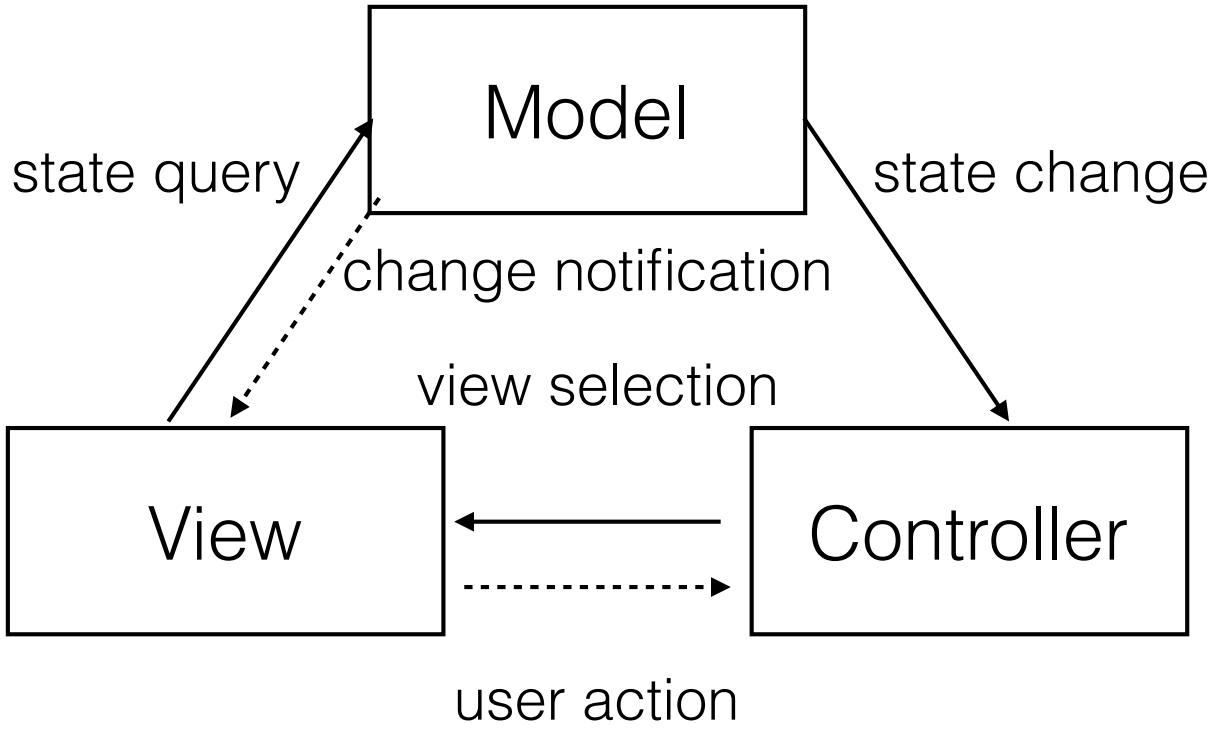




- The model manages application state by responding to requests for information about it **state** (from the view) or instructions to **change state** (from the controller)



- The view renders the model into a form suitable for **interaction** (user interface). Multiple views can exist for a single model given different contexts.



- The controller receives user input and initiates a response by making updates to the **state** of objects in the model and **selecting** a view for the response.

In practice…

- The model is made up of the **database** (tables or documents), **session** information (current state), and **rules** governing transactions.

- The view is the **front-end** website, made up of HTML, CSS, and server-side templates.

- The controller is any client-side scripting that manages **user interactions**, HTTP **request** processing, and business **logic**/preprocessing.

Benefits

- Clarity of design, e.g., model methods providing an API for data and state management.
- Separation of concerns, where any component can be easily used, replaced, or updated individually.
- Parallel development, with construction and maintenance based around adding parts in a modular fashion and separation of concerns.
- Distributable, ie., re-use of parts in different apps.

3-tier vs MVC

- **Communication**: While in 3-tier the presentation layer never communicates directly to the data layer (only through the logic layer), in MVC all layers communicate directly (triangle topology).

- **Usage**: In 3-tier, the client, middleware, and data tiers are on physically separate platforms, but in MVC the model, view, and controller are together.

Other Architectures

- SOA - Service Oriented Architecture
  - everything has a clear interface
  - service is a black box
  - https://apievangelist.com/2012/01/12/the-secret-to-amazons-success-internal-apis/
- Microservices - UNIX philosophy for the web
  - decoupled
  - https://martinfowler.com/articles/microservices.html

SPA vs MPA

- Single page architecture
  - load all html, css, and javascript on the first load
  - use javascript to modify page
  - request data from APIs and/or server
- Multipage architecture
  - changing views means requesting new html and new data from the server

Pros and Cons?

https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58

Bottom Line

- Applications need to persist data somewhere (model)
- Applications need to display information to users (view)
- The layer in between is where much of the complexity lies
- Designers and developers continue to explore different ways of thinking about how to describe the way in which the models and views are updated and managed.