Module 5: SDLC; Structuring your Project; Databases and Data Modeling; ORM's and Sequelize.js; Views and Templates

Edgardo Molina, PhD | Head Instructor

CUNY Tech Prep 2019-2020

Outline

- SDLC
- Structuring your Project
- Databases and Data Modeling
- ORM's and Sequelize.js
- Views and Templates

SDLC

Software Development Life Cycle

How do we manage the development of a project?

How much do we plan?

What requirements do we need upfront?

When do we start coding?

The Waterfall Development Approach

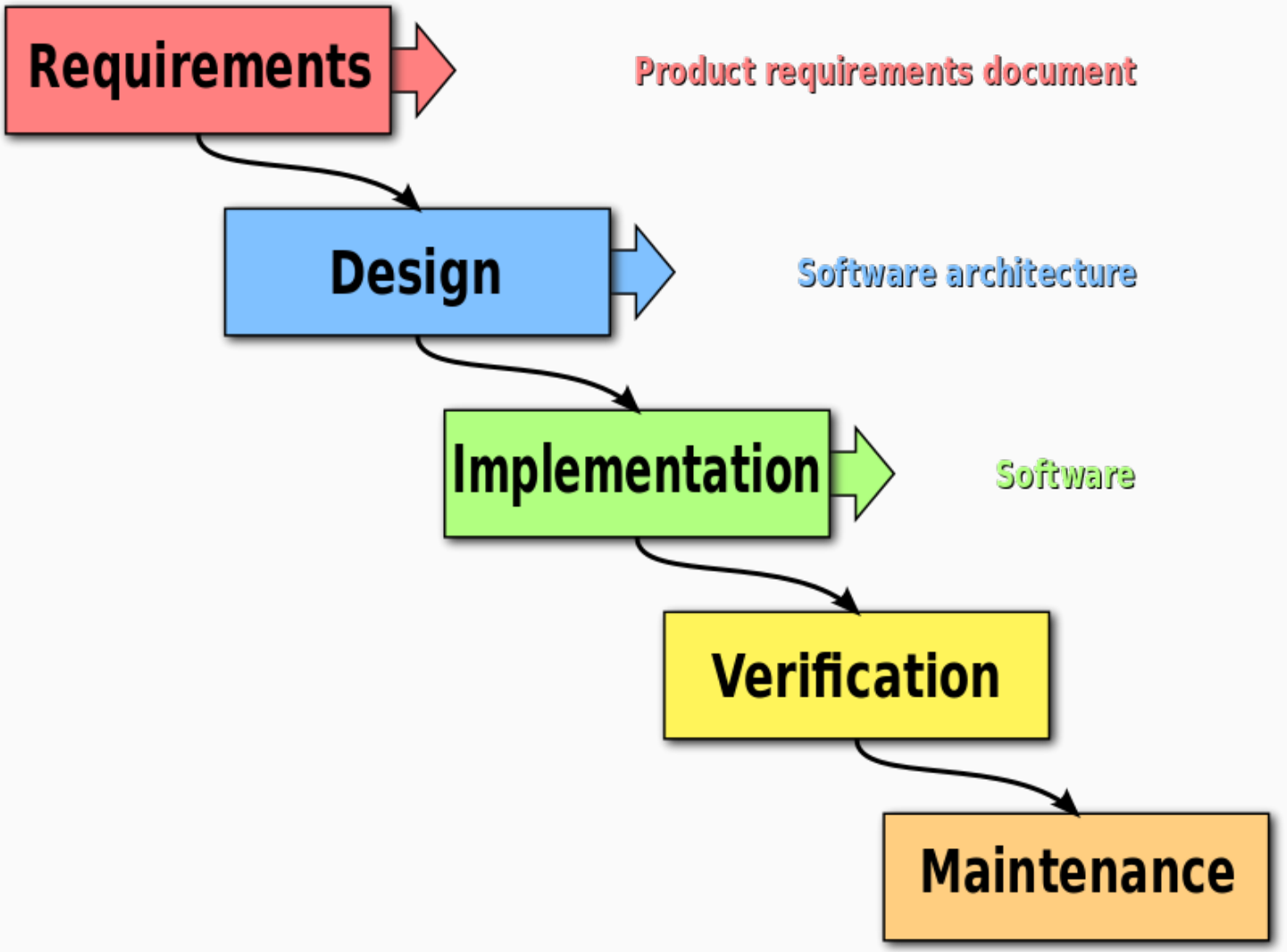

Figure 1:

https://en.wikipedia.org/wiki/Waterfall_model#/media/File:Waterfall_model.svg

The Agile Development Approach

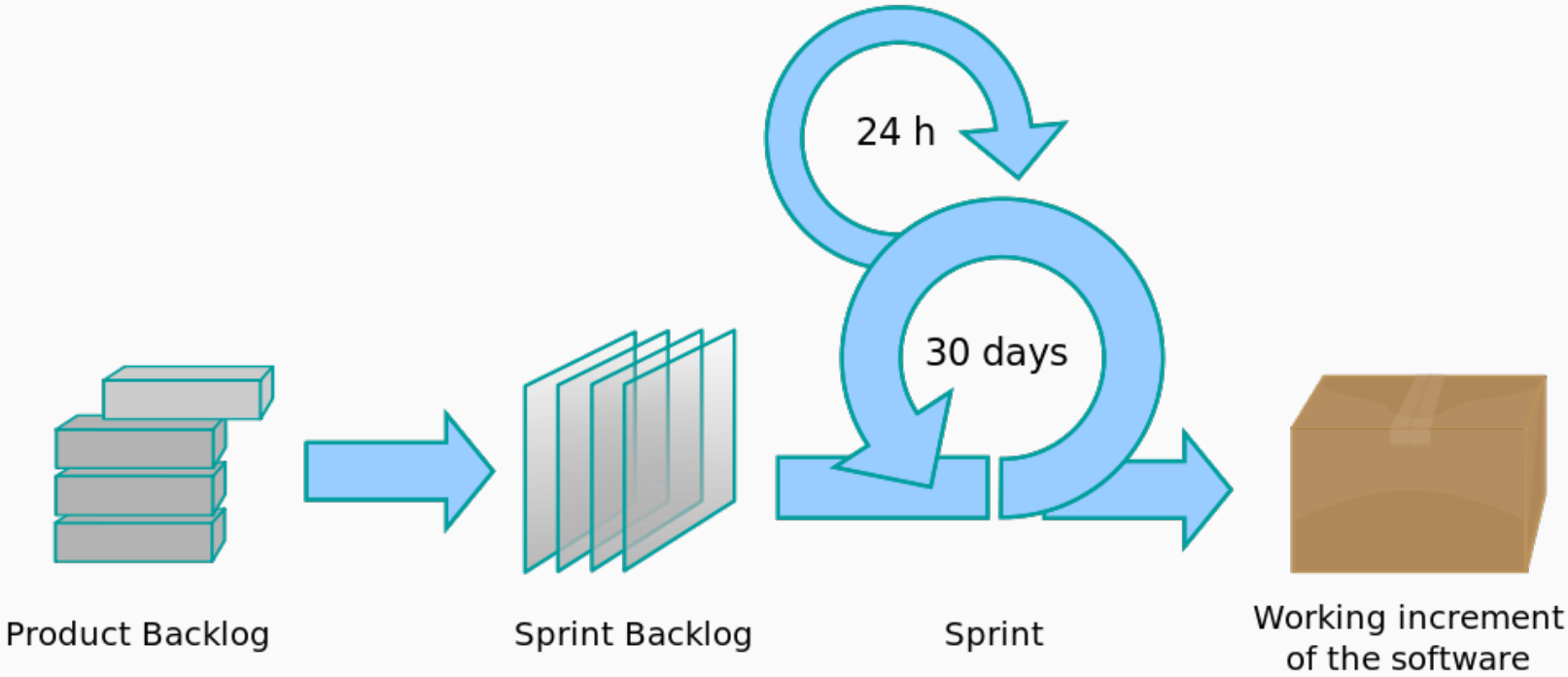

Figure 2:

https://en.wikipedia.org/wiki/Scrum_(software_development)#/media/File:Scrum_process.

The Agile Development Approach

Scrum is a framework for agile development. A sprint typically lasts 2 weeks or 30 days.

Agile *does not* have to use scrum. It is based on the following principles:

Iterative, Incremental, and Evolutionary

- Start small, and deliver early.
- Adapt to changes.
- Continuous feedback

Learn more: https://en.wikipedia.org/wiki/Agile_software_development

Twelve Agile Principles

http://www.agilemanifesto.org/

1. Customer satisfaction by early and continuous delivery of valuable software
2. Welcome changing requirements, even in late development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)

Twelve Agile Principles

7. Working software is the principal measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Best architectures, requirements, and designs emerge from self-organizing teams
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly

Structuring your Project

The Model-View-Controller (MVC) Architecture

What is MVC?

(M)odels, (V)iews, and (C)ontrollers

MVC is a software pattern that defines how we should structure and layout our application code.

Many modern web frameworks have adopted the use of the MVC pattern.

Models

Models ***represent our data entities****, and provide a mechanism to* ***store the data*** *to persistent storage (databases, filesystem, services, etc), and to* ***retrieve the data****.*

Models

> *Models* ***represent our data entities****, and provide a mechanism to* ***store the data*** *to persistent storage (databases, filesystem, services, etc), and to* ***retrieve the data****.*

This is where we model the data that our web app is working with, such as: *Articles, Movies, Songs, Books, Cars, Users, Admins,* etc.

It is an interface to our database/persistence layer, (although there does not have to be one).

Views

Views ***generate our applications output****. Primarily concerned with the* ***presentation*** *and* ***display*** *of our data.*

Views

> *Views* ***generate our applications output****. Primarily concerned with the* ***presentation*** *and* ***display*** *of our data.*

This is where the output presentation is handled. Minimal to no logic should exist here.

We should treat it as if it were a “fill in the blanks” template.

Controllers

> *Controllers contain* ***action methods (route handlers)*** *that* ***receive HTTP requests****, the action method processes any input (route, query, and body parameters), it calls models and services as needed, and finally uses a view to* ***provide output for the HTTP response****.*

Controllers

> *Controllers contain* ***action methods (route handlers)*** *that* ***receive HTTP requests****, the action method processes any input (route, query, and body parameters), it calls models and services as needed, and finally uses a view to* ***provide output for the HTTP response****.*

This is where we map our desired URL route space to specific *action* functions (route handler callbacks) in Express.

The actions manage the lifecycle of http request and response.

Models, Views, and Controllers

MVC is only a pattern. It is up to the programmer to use the pattern effectively to layer their application.

Models, Views, and Controllers

MVC is only a pattern. It is up to the programmer to use the pattern effectively to layer their application.

For example:

- Views should not include db queries or business logic code
- Controllers should not directly talk to the database and it should not directly generate HTML.
- Models should not be concerned with html output or business logic rules*

Models, Views, and Controllers

MVC is only a pattern. It is up to the programmer to use the pattern effectively to layer their application.

For example:

- Views should not include db queries or business logic code
- Controllers should not directly talk to the database and it should not directly generate HTML.
- Models should not be concerned with html output or business logic rules*

** These are the “minimum” application layers. Your business needs may require that you add additional layers between these 3 as your apps become more complex.

What is business logic?

- Authorization and Access rules
    - Who is allowed to access the data?
    - Who is allowed to modify the data?
- What input is needed?
- Which actions are allowed?

Databases

What are databases?

Databases are collections of information/data.

Types of Databases

- Relational Databases (SQL)
- Key-Value Stores (NoSQL)
- Document Stores (NoSQL)
- Graph Databases
- Many other types...

Relational Databases

Relational databases – are structured collections of data with associations between collections.

Relational Databases

Relational databases – are structured collections of data with associations between collections.

Collections of data are typically modeled as database tables.

Relational Databases

Relational databases – are structured collections of data with associations between collections.

Collections of data are typically modeled as database tables.

Rows in a table represent a record (or entry) of a data point.

Relational Databases

Relational databases – are structured collections of data with associations between collections.

Collections of data are typically modeled as database tables.

Rows in a table represent a record (or entry) of a data point.

Columns in a table represent the fields/attributes of each record. Each column in the table is typically constrained to a single data type.

Database Tables

Here is an example database table for a **blogs** resource. Each stored record contains 5 columns, each with a datatype.

table name:		**blogs**				
column datatype:		*int*	*string*	*text*	*datetime*	*datetime*
column name:		**id**	**title**	**body**	**created_at**	**modified_at**
4 Records		1	How to build a website	Lorem ipsum ...	10/1/15	10/1/15
		2	How to deploy with Heroku	Lorem ipsum ...	10/2/15	10/2/15
		3	Github or Bitbucket, which is best?	Lorem ipsum ...	10/3/15	10/3/15
		4	Express.js Tutorial	Lorem ipsum ...	10/4/15	10/4/15

Figure 3: Database Table

Database Tables

Here is an example database table for a **blogs** resource. Each stored record contains 5 columns, each with a datatype.

table name:		**blogs**				
column datatype:		*int*	*string*	*text*	*datetime*	*datetime*
column name:		**id**	**title**	**body**	**created_at**	**modified_at**
4 Records		1	How to build a website	Lorem ipsum ...	10/1/15	10/1/15
		2	How to deploy with Heroku	Lorem ipsum ...	10/2/15	10/2/15
		3	Github or Bitbucket, which is best?	Lorem ipsum ...	10/3/15	10/3/15
		4	Express.js Tutorial	Lorem ipsum ...	10/4/15	10/4/15

Figure 3: Database Table

- **id**: convention for every table to have an *id* column. Each record should have a unique id value.
- **created_at** and **modified_at**: convention for every table to have these columns to track creation time and last edit time. User information can additionally be stored for auditing purposes.

Movies Example

Let's take a look at another example:

id	movie_name	movie_synopsis	genre	year	actor_name	actor_dob	actor_bio	actor_salary
1	Independence Day	Blah blah blah…	sci-fi	1996	Will Smith	9/25/68	In west Philadelphia born and raised	$5M
2	Men in Black	Bleh bleh bleh…	comedy	1997	Will Smith	9/26/68	In west Philadelphia born and raised	$5M
3	I, Robot	Lorem ipsum…	sci-fi	2004	Will Smith	9/27/68	In west Philadelphia born and raised	$28M
4	I Am Legend	Hmm, blah blah…	sci-fi	2007	Will Smith	9/28/68	In west Philadelphia born and raised	$25M

Figure 4: Movie Data

Does this look OK? Are there any problems?

Movies Example

id	movie_name	movie_synopsis	genre	year	actor_name	actor_dob	actor_bio	actor_salary
1	Independence Day	Blah blah blah...	sci-fi	1996	Will Smith	9/25/68	In west Philadelphia born and raised	$5M
2	Men in Black	Bleh bleh bleh...	comedy	1997	Will Smith	9/26/68	In west Philadelphia born and raised	$5M
3	I, Robot	Lorem ipsum...	sci-fi	2004	Will Smith	9/27/68	In west Philadelphia born and raised	$28M
4	I Am Legend	Hmm, blah blah...	sci-fi	2007	Will Smith	9/28/68	In west Philadelphia born and raised	$25M

Figure 5: Movie Data

Concerns with this table:

Data Redundancy

a lot of duplicate data taking up storage

Data Integrity

What if “Will Smyth” is mispelled in 1+ entries?

What if the Bio has to be updated?

Data Modeling

Database Normalization

Normalization is the design of database tables and columns to reduce redundancy and maintain data integrity.

Let's look at a better way to model the Movie data:

movies				
id	name	synopsis	year	genre_id
1	Independence Day	Blah blah blah...	1996	2
2	Men in Black	Bleh bleh bleh...	1997	4
3	I, Robot	Lorem ipsum...	2004	2
4	I Am Legend	Hmm, blah blah...	2007	2

movie_actors			
id	movie_id	actor_id	salary
1	1	1	$5M
2	2	1	$5M
3	3	1	$28M
4	4	1	$25M

genres	
id	name
1	drama
2	sci-fi
3	horror
4	comedy

actors			
id	name	dob	bio
1	Will Smith	9/25/68	In west Philadelphia born and raised

Figure 6: Normalized Movie Data

Database Normalization

Why is this better:

movies				
id	name	synopsis	year	genre_id
1	Independence Day	Blah blah blah...	1996	2
2	Men in Black	Bleh bleh bleh...	1997	4
3	I, Robot	Lorem ipsum...	2004	2
4	I Am Legend	Hmm, blah blah...	2007	2

movie_actors			
id	movie_id	actor_id	salary
1	1	1	$5M
2	2	1	$5M
3	3	1	$28M
4	4	1	$25M

genres	
id	name
1	drama
2	sci-fi
3	horror
4	comedy

actors			
id	name	dob	bio
1	Will Smith	9/25/68	In west Philadelphia born and raised

Figure 7: Normalized Movie Data

Database Normalization

How is this better:

- I can add actors to a movie, without duplicating movie information.
- I can update an actors Bio once, without having to update multiple movies.
- When I add a movie, I don't have to enter data for actors that are already in my database.

How is this more complicated:

- How do I determine the genre name for a given movie?
- How do I lookup all of the movies for a given actor?
- How do I find all of the actors for a given movie?
- How do I determine all the genres an actor has worked in?

SQL – Structured Query Language

Finding answers to my data questions will require looking at multiple tables.

SQL – Structured Query Language

Finding answers to my data questions will require looking at multiple tables.

SQL is a declarative programming language designed to facilitate querying normalized data. It helps us follow the relations in our database tables.

We will take a look at SQL later in the program.

SQL – Structured Query Language

Finding answers to my data questions will require looking at multiple tables.

SQL is a declarative programming language designed to facilitate querying normalized data. It helps us follow the relations in our database tables.

> *We will take a look at SQL later in the program.*

Columns such as `movies.genre_id`, `movie_actors.movie_id`, and `movie_actors.actor_id` produce relations among our records.

Relations and Associations

There are three types of record associations:

- 1 to 1 (1:1)
- 1 to many (1:n)
- many to many (n:m)

One to One Associations

If we look at our movie dataset, we could split the actor table into two tables, an `actor` and `actor_bio` table.

actors			
id	name	dob	actor_bios_id
1	Will Smith	9/25/68	6

actor_bios	
id	bio
6	In west Philadelphia born and raised

Figure 8: One to One example

In our modeling we say:

Each actor has one actor_bio.

> *This is useful when one table has many fields, but many are less frequently used then others. The split is done for query performance.*

One to Many Associations

genres	
id	name
1	drama
2	sci-fi
3	horror
4	comedy

movies				
id	name	synopsis	year	genre_id
1	Independence Day	Blah blah blah...	1996	2
2	Men in Black	Bleh bleh bleh...	1997	4
3	I, Robot	Lorem ipsum...	2004	2
4	I Am Legend	Hmm, blah blah...	2007	2

Figure 9: One to Many Example

In our modeling we say:

Each genre has many movies.

Each movie belongs to a single genre.

Many to Many Associations

movies				
id	name	synopsis	year	genre_id
1	Independence Day	Blah blah blah...	1996	2
2	Men in Black	Bleh bleh bleh...	1997	4
3	I, Robot	Lorem ipsum...	2004	2
4	I Am Legend	Hmm, blah blah...	2007	2

movie_actors			
id	movie_id	actor_id	salary
1	1	1	$5M
2	2	1	$5M
3	3	1	$28M
4	1	2	$4M
5	4	3	$15M
6	4	1	$25M

actors			
id	name	dob	bio
1	Will Smith	9/25/68	In west Philadelphia born and raised
2	Jane Doe	11/26/57	Lorem ip...
3	John Ramon	9/27/89	Lorem ipsum...

Figure 10: Many to Many Example

In our modeling we say:

Each movie has many actors through movie_actors.

Each actor has many movies through movie_actors.

Data Modeling

A visual way to model tables and relations is using ER-Diagrams (entity-relations). Each table is modeled as an object, and relations are arrows.

Entity-Relationship Diagrams

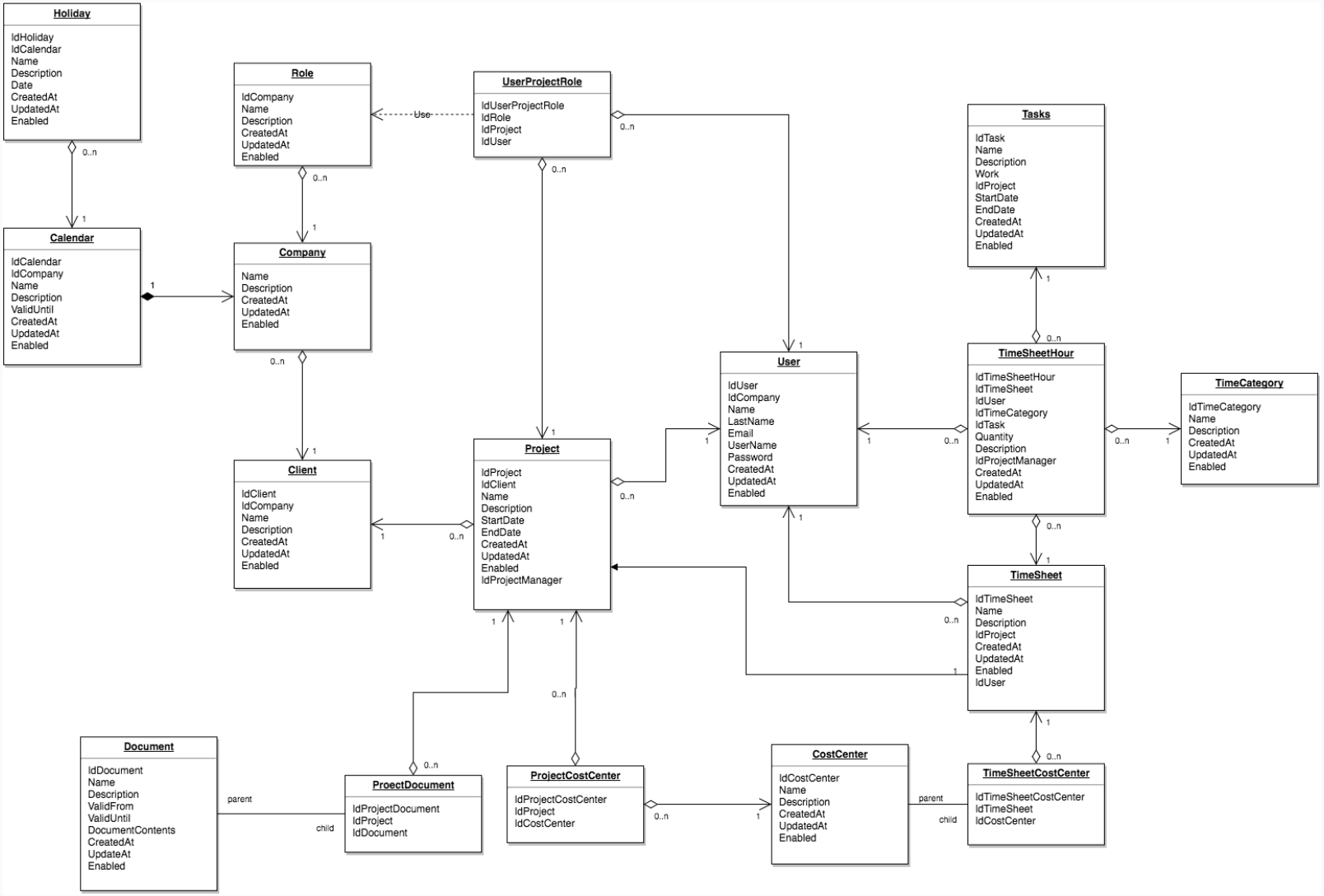

Figure 11: ER-Diagram example https://www.draw.io/

Further Reading

Terms to know

- RDBMS - Relational Database Management System
- Primary Keys
- Foreign Keys

About Databases

Introduction to SQL: http://cs.lmu.edu/~ray/notes/introsql/

[Extra] Codd's 12 Rules

https://www.tutorialspoint.com/dbms/dbms_codds_rules.htm

[Advanced] Database Normalization:

https://en.wikipedia.org/wiki/Database_normalization

[Advanced] Database Normalization:

https://www.tutorialspoint.com/dbms/database_normalization.htm

ORM’s / Sequelize.js

ORM - Object Relational Mapping

The majority of our database tables and associations fit into an object and association abstraction.

ORM's are software tools that help us create objects for each table, and provide us CRUD and association methods for each object.

Sequelize.js
http://docs.sequelizejs.com/

Sequelize.js is an ORM for Relational databases (Postgres, MySQL, Oracle, etc).

ORM - Object Relational Mapping

ORM's generate SQL code for interacting with our databases.

ORM's are not a magic tool for every case. Given very complicated relations and chainings, it may be for performant to write your own SQL command, then to rely on the ORM's generated SQL.

Adding a database

Using a Database

To use a databse in our Express.js apps:

- Install the DB (Postgres)
- Create a DB User for your project
- Install Sequelize.js in your project
- Configure your app to find postgres

Using Sequelize.js

Add sequelize and some tools to our project:

```
npm install --save sequelize pg
npm install -g sequelize-cli
```

sequelize-cli provides us the command line tool for managing the database and generating models and migrations. Add this globally using `-g` option.

Configuring the Sequelize to use Database

```
sequelize init
```

This creates a `config`, `models`, `seeders`, and `migrations` structure in your project.

- **config**: contains db information for your project under different environments
- **models**: will contain your DB models. The `index.js` contains code to auto load the models (careful when you modify it).
- **migrations**: will contain scripts to add/delete/modify the database tables and overall schema.
- **seeders**: will contain scripts to *seed* tables with some basic data records. This is used for testing and development.

Using Sequelize.js

Edit `config/config.json` with your corresponding DB information

```
"development": {
  "username": "pg_user",
  "password": "pg_pass",
  "database": "myproject_development",
  "host": "127.0.0.1",
  "dialect": "postgres"
},
```

This tells sequelize how to find and connect to your database

Using Sequelize.js

Generate models using the sequelize command:

```
sequelize model:create --name Article --attributes
    title:string,slug:string,body:text

sequelize model:create --name Author --attributes
    first_name:string,last_name:string,bio:text
```

These commands will create **both** a model file and a migration file.

Using Sequelize.js

Your Database is not modified/updated unless you **run** the migration scripts.

This is done using the following command:

```
sequelize db:migrate
```

Undoing a migration is done with:

```
sequelize db:migrate:undo
```

You don’t have to use migrations with sequelize

Using Sequelize.js

Both Sequelize and the sequelize-cli tool are very powerful. They have separate documentation. The best way to get information about what sequelize-cli does is to run the command:

```
sequelize help
```

and visiting: https://github.com/sequelize/cli

and you can learn about **migrations** here:
http://docs.sequelizejs.com/en/latest/docs/migrations/

Views and Templates

NOTE

When building simple JSON API's the view layer is lightweight and you often don't need a view library. Although some view libraries are available.

The next slides cover how to use views and templates if you wish to have server-side rendered pages, instead of client-side rendered pages.

Views and Templates

Views are composed of a collection of content templates. Generally the templates produce html.

Templates can produce any text based output (XML, JSON, JS code, etc).

Templates allow for inserting dynamic content within static content.

Handlebars

Handlebars is a templating engine for JavaScript.

It performs variable and expression substitution within the output for any code wrapped in double curly-braces, *i.e.* `{{ myVarible }}`

Simple conditionals and loops can be written within templates. See documentation for if-else and for loop syntax.

Documentation: https://github.com/ericf/express-handlebars

How to structure views (Best Practice)

Mirror your controller and actions structure

Create a `views` folder

Create a sub-folder to *match* each controller name.

Name each template according to your actions and models as necessary.

See `ctp-microblog` example

Layouts

Most templating engines have the concept of layouts. These allow us to maintain common components (headers and footers) in one reusable file.

These are changed in cases where *global differences* are needed, such as *regular vs admin users* or *supported vs unsupported browsers*.

Partials

Many html components are rendered on multiple pages (think sidebars, ads, etc). We can create partial templates for these components that can be rendered from any view.

Let’s add handlebars engine

In `app.js`:

```
const exphbs = require('express-handlebars');

...

app.engine('handlebars', exphbs({
  layoutsDir: './views/layouts',
  defaultLayout: 'main',
}));
app.set('view engine', 'handlebars');
app.set('views', `${__dirname}/views/`);
```

This tells express where to find templates, layouts, and partials.

Rendering

The `res.render('controller/action_name')` function will know to use handlebars to find and execute the views and send the output as a response to the requesting client.

Checkout the `render()` documentation for information on overriding layouts and passing variables.

Also checkout the views in `ctp-microblog`.

Alternatives

Express supports many other templating engines. Check them out, compare, and decide on what you like.

https://github.com/expressjs/express/wiki?_ga=1.152066174.611013023.1462811509#template-engines

Notables:

- EJS
- Jade
- Haml.js