**Cleaning Company Website**

**Feature:**

* Click to Call Action
* Contact Form
* Show Reviews
* GIVE THE OPTION FOR VISITORS TO WRITE THEIR REVIEW
* put a google map
* internalization
* call to action
* social media links
* Chabot
* Online booking
* Blog section

Users:

* The website will have two users.
* The Admin and the client
* Functions of the admin:
* Create blog post, read and approve reviews, update blog sections

Functions of the client:

* Write reviews
* Write comments
* Share post

## React, Node.js Express, MySQL Architecture

The data of the site will be stored in MySQL database and connected to the front end by a Node.js server. For the front end we will use react.js. We’re going to build the application with following architecture:

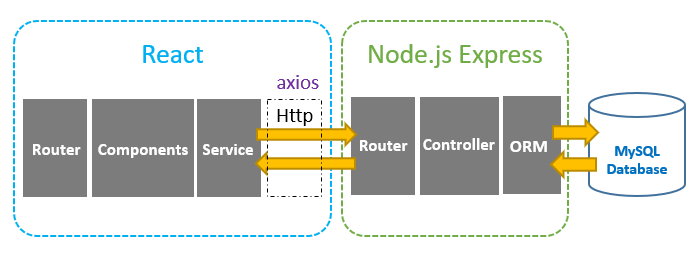


Photo credit: https://bezkoder.com/

From the Features of the website, the blog will have the following data tables, Post, Users, Comments, Emails and user reviews. The data that stored in the database are;

For the Post:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| id | title | featured-image | description | content | createdAt | slug | Users\_Id |

For the comments

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Id | post\_id | comments\_id | name | email | publishedAt | content |

For the Reviews:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| name | email | content | rating | id |

For the News Letter:

|  |  |  |
| --- | --- | --- |
| id | name | email |

For the Contact Form:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Id | name | email | subject | message | submitted |

Using a database tool, we design the database model showing their various relations

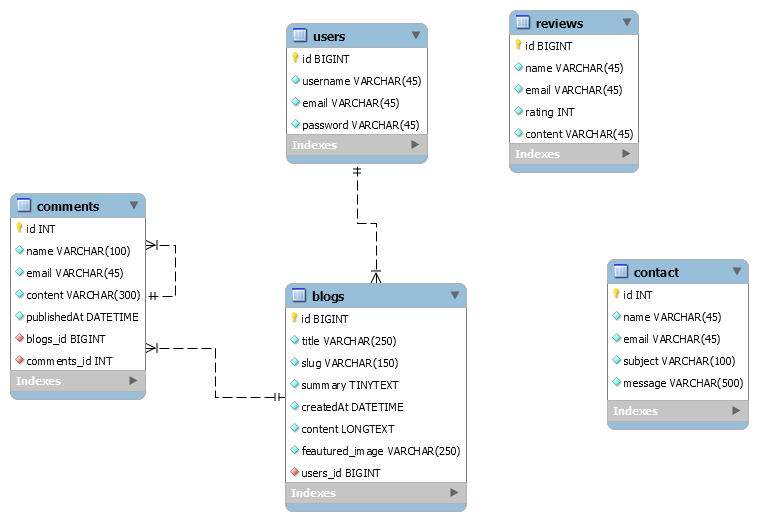


Figure 1: Physical Database Model

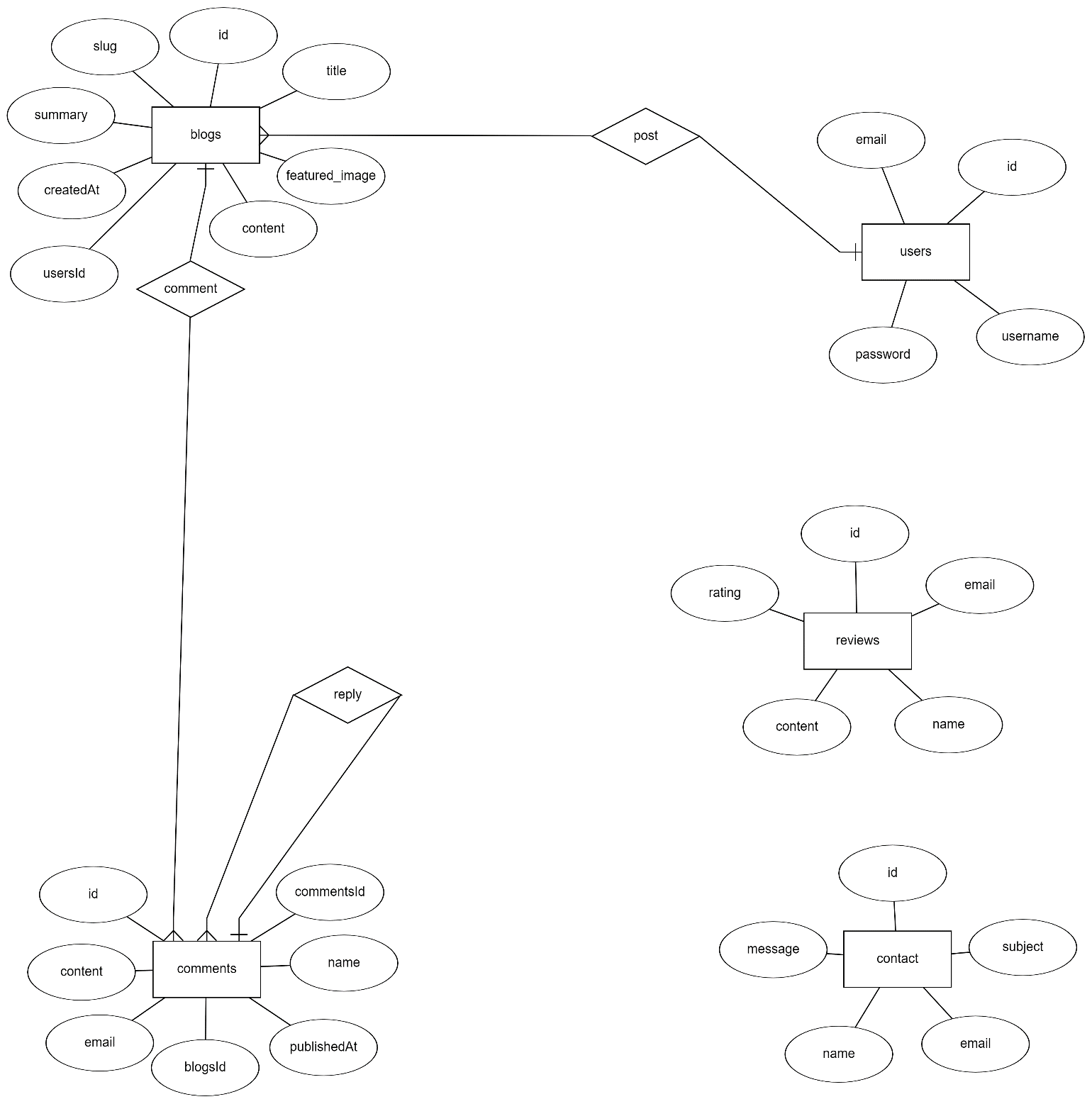


Figure 2: Conceptual Database Model

To manage the database, we use Sequelize. [Sequelize](https://sequelize.org/) is a promise-based ORM for Node.js. It supports the dialects PostgreSQL, MySQL, SQLite and MSSQL and features solid transaction support, relations, read replication and more.

Following the MVC Pattern of Development,

We create the models, controllers and config folders. The Sequelize cli handles the generation of the models.

Models

To create all the models, we need

sequelize model:generate --name Users --attributes userName:STRING,email:STRING,password:STRING

sequelize model:generate –name Reviews --attributes email:STRING,name:STRING,rating:INTEGER,content:STRING

sequelize model:generate --name Mails --attributes name:STRING,email:STRING,subject:STRING,message:STRING,number:STRING

sequelize model:generate --name Post --attributes title:STRING,slug:STRING,summary:STRING,content:String,featuredImage:String,user\_id:INTEGER

sequelize model:generate --name Comments --attributes name:STRING,email:STRING,content:STRING,post\_id:INTEGER,comments\_id:INTEGER

These Sequelize Models represents tables in MySQL database. These columns will be generated automatically: *id*, *title*, *description*, *published*, *createdAt*, *updatedAt*.

After initializing Sequelize, we do not need to write CRUD functions, Sequelize supports all of them, for example the Post model:

* create a new Post: [create](https://sequelize.org/master/class/lib/model.js~Model.html#static-method-create)(object)
* find a Post by id: [findByPk](https://sequelize.org/master/class/lib/model.js~Model.html" \l "static-method-findByPk)(id)
* get all Posts: [findAll](https://sequelize.org/master/class/lib/model.js~Model.html" \l "static-method-findAll)()
* update a Post by id: [update](https://sequelize.org/master/class/lib/model.js~Model.html#static-method-update)(data, where: { id: id })
* remove a Post: [destroy](https://sequelize.org/master/class/lib/model.js~Model.html#static-method-destroy)(where: { id: id })
* remove all Posts: destroy(where: {})
* Find all Posts by title: findAll ({where: {title: ... } })



Figure:3 Generated Post Model



Figure 4: Generated Mails Model



Figure 5: Generated Post Model



Figure 6: Generated Reviews Model



Figure 7: Generated Users Model

After generating the models with Sequelize, we need to define the relations in the models manually.

Models associations

1:1

The first relation is that a Post can be created by one user. To establish this we add this line of code to the Post model.



These are APIs that our Node.js Express server is expected to export:

|  |  |  |
| --- | --- | --- |
| Methods | Urls | Actions |
| GET | api/posts | Get all Post |
| GET | api/posts/:id | Get Post by id |
| POST | api/posts | Add new Post |
| PUT | api/posts/:id | Update Post by id |
| DELETE | api/posts/:id | Remove Post by id |
| DELETE | api/posts | Remove all Post |
| GET | api/posts?title=[kw] | find all Posts which title contains 'kw' |
|  |  |  |
| GET | api/comment/:post\_id | Get all Commets for a particular post |
| POST | api/comment | Add new comment |
| DELETE | api/comment/:user\_id | Remove comment by user\_id |
|  |  |  |
| GET | api/review | Get all review |
| POST | api/review | Add new review |
| DELETE | api/review/: comment\_id | Remove review by id |
|  |  |  |
| Get | api/user/:id | Get user by id |
| Post | api/user | Add new user |
| Delete | api/user/:user\_id | Remove user by id |
|  |  |  |
| Get | api/user/:id | Get user by id |
| Post | api/user | Add new user |
| Delete | api/user/:user\_id | Remove user by id |
|  |  |  |
| Post | api/mail | Add new mail |