Q 1.You’ll need to create an API server with CRUD based routes using NodeJS/Python

const UserModel = require('../model/user')

// Create and Save a new user

exports.create = async (req, res) => {

if (!req.body.email && !req.body.firstName && !req.body.lastName && !req.body.phone) {

res.status(400).send({ message: "Content can not be empty!" });

}

const user = new UserModel({

email: req.body.email,

firstName: req.body.firstName,

lastName: req.body.lastName,

phone: req.body.phone

});

await user.save().then(data => {

res.send({

message:"User created successfully!!",

user:data

});

}).catch(err => {

res.status(500).send({

message: err.message || "Some error occurred while creating user"

});

});

};

// Retrieve all users from the database.

exports.findAll = async (req, res) => {

try {

const user = await UserModel.find();

res.status(200).json(user);

} catch(error) {

res.status(404).json({message: error.message});

}

};

// Find a single User with an id

exports.findOne = async (req, res) => {

try {

const user = await UserModel.findById(req.params.id);

res.status(200).json(user);

} catch(error) {

res.status(404).json({ message: error.message});

}

};

// Update a user by the id in the request

exports.update = async (req, res) => {

if(!req.body) {

res.status(400).send({

message: "Data to update can not be empty!"

});

}

const id = req.params.id;

await UserModel.findByIdAndUpdate(id, req.body, { useFindAndModify: false }).then(data => {

if (!data) {

res.status(404).send({

message: `User not found.`

});

}else{

res.send({ message: "User updated successfully." })

}

}).catch(err => {

res.status(500).send({

message: err.message

});

});

};

// Delete a user with the specified id in the request

exports.destroy = async (req, res) => {

await UserModel.findByIdAndRemove(req.params.id).then(data => {

if (!data) {

res.status(404).send({

message: `User not found.`

});

} else {

res.send({

message: "User deleted successfully!"

});

}

}).catch(err => {

res.status(500).send({

message

Q.2Using your API, users can Create, Read, Update or Delete data from the database

NAME

Dancer::Plugin::SimpleCRUD - very simple CRUD

(create/read/update/delete)

DESCRIPTION

A plugin for Dancer web applications, to use a few lines of code to

create appropriate routes to support creating/editing/deleting/viewing

records within a database table. Uses CGI::FormBuilder to generate,

process and validate forms, Dancer::Plugin::Database for database

interaction and HTML::Table::FromDatabase to display lists of records.

Setting up forms and code to display and edit database records is a very

common requirement in web apps; this plugin tries to make something

basic trivially easy to set up and use.

SYNOPSIS

The following assumes that you already have a working Dancer app and

have put your database connection details in your "config.yml" to be

read by Dancer::Plugin::Database, which this plugin uses in order to

obtain a database connection.

# In your Dancer app,

use Dancer::Plugin::SimpleCRUD;

# Simple example:

simple\_crud(

record\_title => 'Widget',

prefix => '/widgets',

db\_table => 'widgets',

editable => 1,

);

simple\_crud(

record\_title => 'Team',

prefix => '/teams',

db\_table => 'team',

labels => { # More human-friendly labels for some columns

venue\_id => 'Home Venue',

name => 'Team Name',

},

validation => { # validate values entered for some columns

division => qr/\d+/,

},

input\_types => { # overriding form input type for some columns

supersecret => 'password',

lotsoftext' => 'textarea',

},

key\_column => 'id', # id is default anyway

editable\_columns => [ qw( venue\_id name division ) ],

display\_columns => [ qw( id venue\_id name division ) ],

deleteable => 1,

editable => 1,

addable => 0, # does not allow adding rows

sortable => 1,

paginate => 300,

template => 'simple\_crud.tt',

query\_auto\_focus => 1,

downloadable => 1,

foreign\_keys => {

columnname => {

table => 'venues',

key\_column => 'id',

label\_column => 'name',

},

},

table\_class => 'table table-bordered',

paginate\_table\_class => 'table table-borderless',

custom\_columns => [

{

name => "division\_news",

raw\_column => "division",

transform => sub {

my $division\_name = shift;

my $label = "News about $division\_name";

$division\_name =~ s/([^-\_.~A-Za-z0-9])/sprintf("%%%02X", ord($1))/seg;

my $search =

return "<a href='$search'>$label</a>";

},

column\_class => "column-class",

},

],

auth => {

view => {

require\_login => 1,

},

edit => {

require\_role => 'Admin',

},

},

);