# Ruby on Rails

Rails app consists of:

* Model
* Route
* Controller
* view

Steps to create a ruby ‘application’

* Generate new rails app
* Generate controller and add a new *action*
* Create a route that maps a URL to the controller action
* Create a view with html and CSS
* Run the local web server and preview in the browser

To save user data, the controller interacts with the model.

The model in turn interacts with the Database to get data.

This data is then sent to the view.

Create new app:

**$ Rails new myMessengerApp**

Install gems from gemfile

**$ Bundle install**

Generate new model

**$ rails generate model Message (name:string email:string)**

* This makes a model file app/models/message.rb, which represents a table in a database.
* Also a migrate file db/migrate/ which is used to update the database.

Db/migrate

Uses the def function to show what to change in the database.

e.g. Create table.

We can add to the table with:

t.text :content

t.timestamps

rake db:migrate updates the databases with the new messages data model.

Rake db:seed seeds the database with sample data from db/seeds.rb

**$ Rails generate controller messages**

Show all routes in the application

**$rake routes**

Model holds all data; Rails uses SQLite by default. You can use another DB manager but that requires some configuration.

Generate a controller with methods

**$ rails generate controller say hello**

So, to start a new file:

**$Rails new myApp**

**$Bundle install**

**$Rails generate model myModel**

**$Rails generate controller say hello**

Or you can use scaffolding(After bundle install):

**$Rails generate scaffold Recipe name: string content: text**

You can also destroy the controller

**$Rails destroy controller say hello**

## HTTP Requests

**PUT:**

* Put a file or resource at a specific URI (Uniform Resource Indicator)

**POST:**

* Send data to specific URI and expect the resource at the URI to process the request

**GET:**

* Request data from a specific resource

**DELETE:**

* Request the origin server delete the resource identified by the request URI

## Routes

**Resourceful and non-resourceful**

**Resourceful** – when you use rails g photo, routes and pages are generated automatically.

**Non-resourceful** - Does not create all the shit for you, if you don’t want all the functionality of rails g.

With non-resourceful routing you can map all the routes yourself.

In the routes file, the root should be on the top of the file.

# Ruby Syntax

**$(0..9).to\_a**

**$ a == (0..9).to\_a**

**$ a = [0,5,6,1,6]**

**$a.sort**

**$a.last**

**$a.reverse**

**$a.empty?**

**$a[2..(a.length-1)]**

**Blocks**

Curly braces to do cool stuff

# Ruby on Rails Databases

* Relational database
* Active record

## Database

* Rows, columns
* Primary key
* Database schema (collection of tables and their relationships)
* DBMS (Database management system)

In rails we don’t use SQL directly since it is not object-oriented and it repeats itself.

## Ruby DB Model

**Object-relational mapping**

ORM is written in an object oriented language and wrapped around a relational database.

* An object class -> A table in DB
* An object instance -> A record in the tables

## Object-Relational Mapping

Old-school way:

String sql = “SELECT all FROM persons WHERE first\_name = ‘David’ “;

DbCommand cmd = new DbCommand(connection, sql);

Result res = cmd.Execute();

In ruby we use the ORM API

* No need to write code for specific DB
* Reduce code and increase efficiency
* Rich query interface
* Concurrency

Disadvantages

* Overhead: a layer in between two interfaces, higher CPU and RAM usage
* SQL ignorance

## Active Record

* Data is represented as models, which can create associations
* Inheritance can be achieved through related models
* Validation
* OOP way of dealing with databases

## Creating Active record models

Creates a subclass of the ApplicationRecord class.

You can query the model with $rails console.

Active Record: CRUD

* Create new e.g. article.new, article.title, article.save or article.destroy etc.

## DB Migration

* Rails way of creating and updating databases
* Doesn’t use SQL. Wrong operations can be rolled back.

Create model

Create a scaffolding

Create a standalone migration

Rails generate model Student name:string address:string

Rake db:migrate - update DB in schema

Rake db:rollback -

# 27.9 Database Design

Database design- E.g. to keep track of students sports activities.

To keep data organised, it is better to have many tables with a Primary key (ID) than have a giant table with lots of information.

Students table: ID is the primary key

Activities table: ID is a foreign key

Rails uses foreign key reference to relate tables to primary keys of another table for example.

There are declaring associations like belongs\_to, has\_one, has\_many etc.

## One-to-one associations

Class User < ActiveRecord::Base

Has\_one :profile

End

Class Profile < ActiveRecord::Base

Belongs\_to :user

End

The first one checks the users id with the profiles ID.

## Methods with the has\_one association

E.g.

User.profile

User.profile.nil?

User.build\_profile(attributes={})

>> user.create\_profile :name => ‘Jane Doe’, :color => ‘pink’

## One-to-many associants

Has\_many :articles

A row in one table is related to one or more rows in another table.

User = User.first

User.articles << Article.first

<< = Append operator.

## Has\_many associations

User.articles

User.articles << article

User.article.ids

User.articles.clear

User.articles.find

User.articles.create(attributes={})

## Many-to-many Associations

* Two tables are connected to multiple rows on both sides.

Class Article < ActiveRecord::Base

Belongs\_to :user

Has\_and\_belongs\_to\_many :categories

End

## Seeding Data

Db/seeds.rb contains All the record creation needed to seed the DB with it’s default values

Rake db:seed

## Rich Many-to-Many Associations

* Join multiple tables
* Retrieve all comments added to users’ articles
* You can say “users have many comments through articles”

## Advanced finding

>> Article.where(:title => ‘Advanced Active Record’)

>> Article.where(“created\_at > ’23-04-2013’ OR body NOT LIKE ‘%model%’”)

## Using an Array Condition Syntax

* Using this syntax can specify conditions to DB calls safer than SQL syntax

E.g. Find all the articles that were published before today:

>> Article.where(“published\_at < ?”, Time.now)

Find all articles created at the same time as the last article.

>> Article.where(“created\_at =?”,Article.last.created\_at)

Association Proxies

* Chain multiple calls to Active record
* >> User.first.articles.all
* >> current\_user.articles.create(:title => ‘Private’, :body => ‘Body here.’)

Joining Tables Together

* Inner and outer Join
* Inner join searches tables for matching or overlapping data, returns info to a new table.
* Product.joins(:quantities)

## Left Outer Join (or Left Join)

* Select a set of records whether or not they have associated records.
* I.e. just take all the records from one table

## Joining multiple associations

* Article.joins(:category, :comments)
* “Return all articles that have a category and atleast one comment”)

## Specify Conditions on the joined tables

* time\_range = (Time.now.midnight -1.day)..Time.now.midnight

# GIT 30 SEP

Git is a distributed Version Control System. Tracks changes in files. Does not save the whole thing, just the differences.

Working directory -> Staging Area -> Git Repo

# 25 Oct