Bachelor’s thesis

Information and Communications Technology

2021

Tuomas Maanpää

CONTINUOUS INTEGRATION WITH MICROSERVICE ARCHITECTURE

TURKU AMK

TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR’S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology

2021 | 33 pages

Tuomas Maanpää

CONTINUOUS INTEGRATION WITH MICROSERVICE ARCHITECTURE

Testing an application should be obvious part of an application development for developers but it might be left out of the application for one reason or another. This may lead to regression and dissatisfaction between developers and customers. In short regression means that the project would have same problems it had long time ago and developers might have to make same features multiple time. Sometimes even regression might give more work for developers because the application does not work as intended and it needs to be fixed. Continuous Integration is used to slow the regression, identify mistakes, and notify if something does not work as intended.

There are many different applications focusing on Continuous Integration (CI) and every one of them has similar features. So, deciding which CI application is best for a project is left on the developers. Most of the times it is a clever idea to use CI application developers are familiar with or something that works well with the project. It is also a good practice to check if the CI application supports the computer languages in the project because most the test is usually implemented within the project.

This thesis aims to demonstrate the importance of testing in general and how to mitigate the regression by creating a small project with tests. The project was created with Ruby on Rails and it was implemented in a GitLab monorepo. Monorepo is a repository that has multiple repositories under one repository. The Continuous integration will be created with GitLab CI/CD mostly because the repositories are in GitLab. This thesis is literature research between technologies but contains created examples to support understanding of the tests and how the tests should work on a project.

KEYWORDS:

Continuous integration, microservice architecture, monolith architecture, ruby on rails, docker, GitLab CI/CD

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintätekniika

2021 | 33 sivua

Tuomas Maanpää

JATKUVAA INTEGRAATIOTA MIKROPALVELU ARKKITEHTUURISSA

Sovellusten testaaminen pitäisi olla itsestään selvyys sovellusten kehitysvaiheessa. Usein kyseinen vaihe jää tekemättä syystä tai toisesta, mikä johtaa sovelluksen taantumiseen. Tämä voi myös johtaa kehittäjien ja asiakkaiden tyytymättömyyteen sekä tuottaa kehittäjille lisää työtä virheiden korjaamiseen takia. Jatkuvaa integraatiota eli sovellusten uusiutuvaa testaamista, käytetään virheiden torjumiseen sekä estäessä projektin vanhojen ongelmien palaamista. Jatkuva integraatio lähettää sovelluksen avulla kehittäjille viestiä huomatessaan ongelman.

Jatkuvaan integraatioon sovelluksia on monia, mutta useimmat sovellukset tarjoavat lähes samanlaista palvelua. Joten pitää miettiä, minkälainen jatkuvan integraation sovellus olisi parempi kehitettävään projektiin. Valinnassa kannattaa ottaa huomioon onko kehittäjillä aiempaa kokemusta jatkuvan integraation sovelluksesta tai miten hyvin jatkuvan integraation sovellus tukee projektin ohjelmointikieltä.

Tämä opinnäytetyö on pääsääntöisesti kirjallisuustutkimus eri teknologioista, joita voitaan käyttää jatkuvassa integraatiossa, joka sisältää luotuja esimerkkejä. Tässä opinnäytetyössä tarkasteltiin testauksen tärkeyttä sekä luotiin projekti, jonka ohella hyödynnetään jatkuvaa integraatiota. Projekti luotiin Ruby on Rails-kielellä joka sijaitsee GitLabin monorepositoriossa. Monorepositorio tarkoittaa yhtä arkistoa, joka sisältää monta eri arkistoa. Jatkuva integraatio kehitettiin GitLab CI/CD:n avulla, koska sitä pystytään hyödyntämään paremmin GitLab-repositorien kanssa.

ASIASANAT:

Jatkuva integraatio, mikropalvelu arkkitehtuuri, monoliittinen arkkitehtuuri, ruby on rails, docker, gitlab CI/CD

CONTENTS

FIGURES

LIST OF ABBREVIATIONS

API	Application Programming Interface
DB	Data Base
DSL	Domain-Specific Language
CD	Continuous Delivery / Continuous Deployment
CI	Continuous Integration
CLI	Command Line Interface
CSS	Cascading Style Sheet
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
REST	Representational State Transfer
SSH	Secure Shell
TDD	Test Driven Development
UI	User Interface
VCS	Version Control System

1. INTRODUCTION

People all over the world are gaining easier access to the internet and surfs through the internet to work, to gain information or simply to entertain themselves. That means users are more likely to go into different webpages to use their services. Depending on the customer base, one server and database is not a valid option because it would slow the page considerably. That means that there should be multiple databases for one website and one database should store different data depending on the information needed for the site.

Microservice architecture is a method of splitting tasks into small services inside a web application project. For example, the project could have developers working for web shop tasks and another group that working on customer service. This means one service specializes into one service and should work separately from other services. The opposite of microservice architecture is a monolith architecture where everything is centralized into a large web service project and is thought of as a traditional method of building applications.

With multiple developers working on a different service potentially increasing the development rate of the project might lead to regression. In short, the project might lag behind more easily without anybody noticing bugs at first. Bugs are features that makes the service work in a way that is not intended. Test driven development (TDD) and continuous integration (CI) slows the regression. In both, developers carry out tests that checks if everything is functioning as intended. If some of the tests fail the continuous integration will inform the developers which test fails and they can fix the mistake or edit the tests.

There are already many tutorials and useful information about CI with microservices devised by large companies and developers. For example, Microsoft has a document entitled "*CI/CD for microservices architectures*" [1] available online, and Alibaba Cloud has "*Continuous Deployment with Microservices*" [2] that goes a little bit further from the topic by including continuous deployment.

The purpose of this thesis is to find a method to implement continuous integration into a microservice architecture. Ruby on Rails is used to create a small project which demonstrates two methods of how to create some simple tests which will be integrated into a monorepo. A monorepo is a repository which contains multiple repositories. In this project, monorepo is used to facilitate the creation of a Docker environment.

2. IMPORTANT CONCEPTS IN MICROSERVICE ARCHITECTURE

This chapter presents the most important concepts of the thesis with examples. It also discusses the advantages and disadvantages of microservice architecture and what benefits testing offer to the project and the developers.

2.1 Microservice architecture

Microservice architecture is a method of building an application from smaller parts called services. Each service specializes in one purpose and should work independently from other services. An example of a service would be a website which has a store that could be implemented as a separate feature from the rest of the website. The website would also have other services such as customer service system or some fora that could also be implemented into different services. Because the application is split into distinct parts, developers can work on different services increasing flexibility. It is also possible to write different services in different languages, but it is not recommended because it can make the code more confusing and require developers with different skills. Services are connected by Application Programming Interfaces (APIs). In web development, this connection would be with Representational State Transfers (REST) that connects all the services in a specific pipeline. In a linear execution, a pipeline gives instructions to the process. Services also communicates with each other with HTTP/REST protocols. [3]

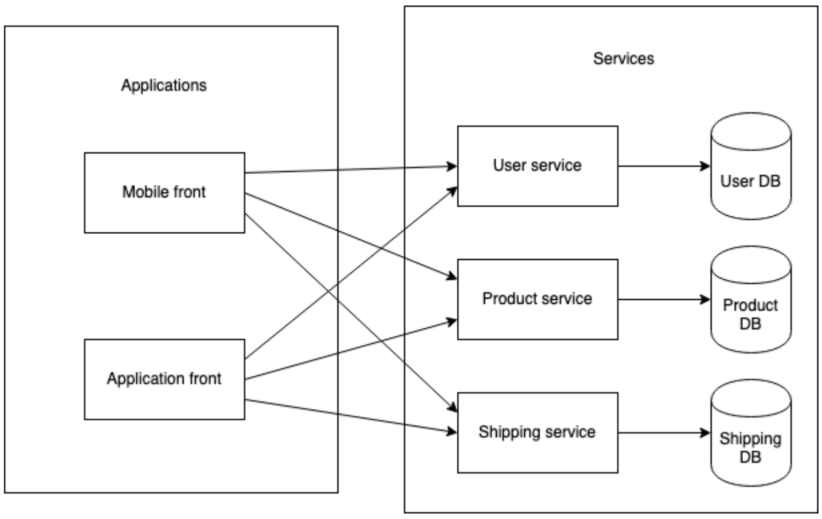


Figure 1: Example architecture of a microservice.

Figure 1 presents a simple application that uses microservice architecture. In Figure 1 the mobile front and application's front are applications that uses all of the services. In this example the User service connects to User database where it stores all user's credentials such as email, name, address and password. This applies to all services because all services have one purpose, they specialize in. [4]

To state the whole microservice architecture into a layman's terms microservice architecture is like a meal a customer recieves by giving a ticket that contains a few courses. The ticket would have for example a vegetarian course. That would be one service. The ticket could also be a lamb or chicken course that would add two more services. The pipeline is everything that happens in the middle of a specific order. For example, the first course is first course if it comes before the main course but after the main course it would be a dessert. This could also mean that different chefs could only focus on desserts or main courses.

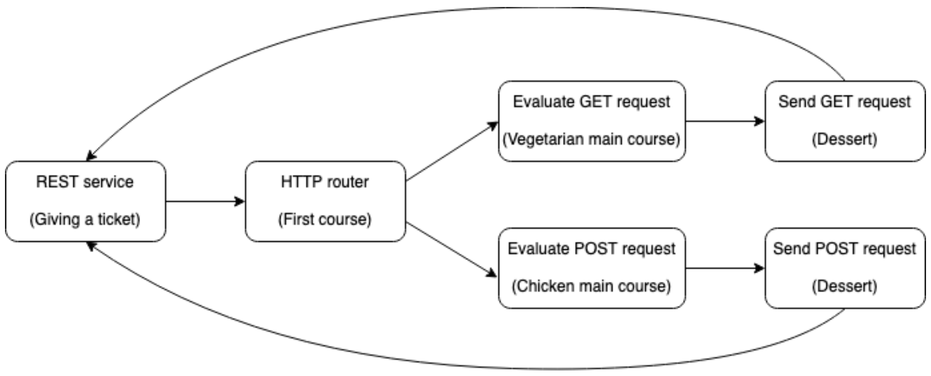


Figure 2: Microservice architecture pipeline with example.

Figure 2 has an example pipelines of the microservice architecture with the earlier example. All courses could change but if there is only one first course and a specific main course with a related dessert. After eating the dessert, it is possible to use another ticket to get a new course. [5]

There are both advantages and disadvantages when developing websites with microservice architecture. The advantages would be that it enables continuous delivery, automatic release after successful continuous integration, in a large and complex applications. Developers could also focus on a different service at a time improving its maintainability and flexibility. This increases the productivity of the new developers because they could start working on a part, they know something of. This eases the deployment of the services because the services are anindependent part of the project. It also improves fault checking. For example, if one service does not work as intended the developers could see what service causes the problem and would be able to fix the service while the others work normally. [32]

Even though microservice architecture might sound good for every upcoming project it has some drawbacks. Those drawbacks occur when developers must deal with additional complexity that comes from creating the services. The developers would have to implement a system that would connect all services together and if one of the services needs something from another service it would need good cooperation with the team. That would also increase complexity in the production because developers would have to manage a system that contains small parts, in other word, services. These services would also increase memory consumption in production and information barriers between developers if their co-operation is not adequately good.

2.2 Testing and Continuous integration

Testing web pages is an important part of web development. It helps developers to spot bugs and slows regression, in short it slows loss of progress. Also, because developers can see if there is something wrong in their code faster it will give them more confidence in their work and increases their productivity. [6]

Tests are a specific code that focuses on testing the code that the developers have worked on. Especially in the backend where developers focus what happens behind the visuals. For example, the application could have a test that would check if the user has given a username. The test would check if the username parameter had a value and returns true while passing the test. There could also be a tests that checks if the parameter is empty and depending on the wanted value it is possible to change the expected value.

Continuous integration (CI) means a development practice when developers integrate code to the repository and their code is verified by automated tests. The tests are not always necessary in CI, but they are typically implemented to ease the developers. [7] CI helps to detect errors in the code by checking the automated tests. If the code has tests that does not pass the test, it will inform the developer. [8]

A practical example of CI would be that when there are multiple developers working on a project and they edit the code, merge it together and after many different versions and merges, it would in the end create so called ‘merge hell’ that is when the project has too many merge conflicts. The CI tries to take those merge hells out by running tests and merging it so often that it would not hurt as much as one large merge after a long time.

Continuous Integration is also a mostly misunderstood concept. Some understand it means only the building and testing parts of the application when it should be merging files automatically many times. Especially after trying to understand how other developers have made CI to their applications, they have created only tests for CI part and continue straight to deployment. Of course, some developers use Continuous Deployment(CD) too so there is at least one place where the application is built.

3. CREATING THE PROJECT AND THE TESTS

Continuous integration is important in application development because it decreases regress and gives more confidence for the developers. There are multiple ways to make a different CI platform for the app but let’s focus on different CI tools. In this chapter there is also information about Rails and small tests made with its two most used testing suites.

3.1 Continuous integration tools

There are many different CI tools that focuses onto different things in CI like speed or accessibility. Most CI tools also offers clouds where it works on the builds. Testing is not always implemented with continuous integration, but it is recommended and mostly done.

3.1.1 Jenkins

Jenkins is CI server that offers automated testing for applications. It offers uncomplicated way to make the automated testing environment for projects written in almost any programming languages. Jenkins does not create all the scripts for the individual steps, but it will help when developers who are creating the whole project. [20]

Jenkins is an open-source project, so anyone can edit and develop it. It is among the best CI tools that there is with as good results as the others that are for example CircleCI and GitLab-Ci.

3.1.2 CircleCI

Circleci is a CI tool that focuses on running test and builds fast. They are also made to be optimized for speed. CircleCI can run pipelines efficiently because of caching, docker and resourcing classes. CircleCI also allows Secure Shell (SSH) scripting to debug build issues, maintaining the pipelines and configuring caches in the testing environment. CircleCI starts running the build tests in a Docker container or in a virtual machine after making a minor change in the repository. Each job CircleCI runs are run on a different containers or virtual machines. [21]

3.1.3 TeamCity

TeamCity is a commercial CI tool provided by JetBrains. TeamCity allows multiple product builds and run builds straight from the Integrated Development Environment (IDE), in short straight from a computer where the developers are editing the application. TeamCity can also take code from different Version Control Systems (VCS) and detects hung builds. Hung build means that a build has gotten stuck and cannot continue on its own. TeamCity has a distributed grid architecture that means it compromises the server and build agents that runs the build. Because TeamCity is supposed to watch builds it allows developers to use different agents to increase system performance. [22]

3.1.4 GitLab CI/CD

GitLab CI/CD is a continuous integration service made for GitLab. It builds and tests the repository after a developer pushes code to a repository. GitLab CI/CD is part of GitLab which company may use as a repository manager, that means it should be integratable for the different repositories the company has. GitLab CI/CD has an open-source version and enterprise version which are easy to learn. The builds are run in separate machines which developers can add as many as they want and they can run parallel increasing the speed of the builds. [23]

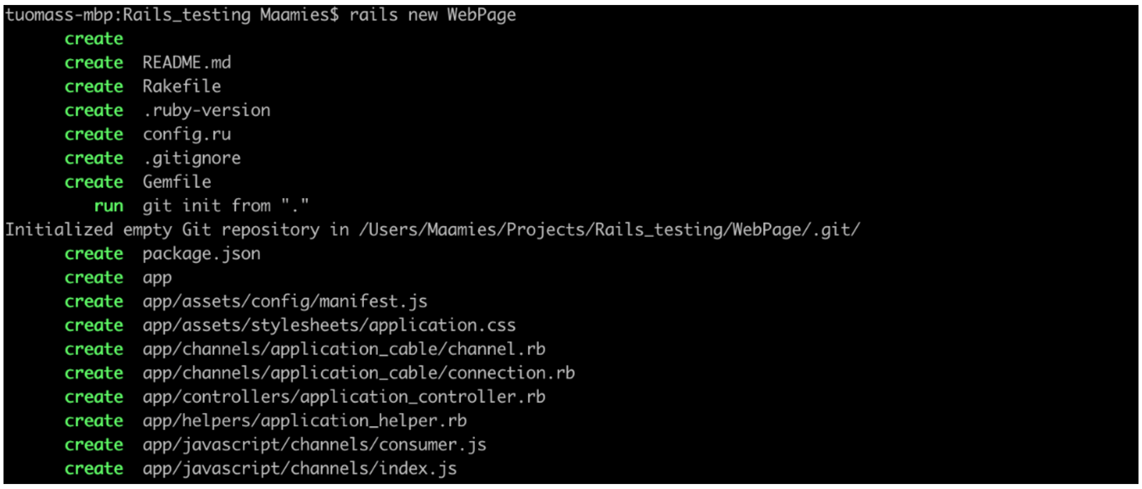
3.2 Ruby on Rails

Ruby on Rails, most of the time shortened as Rails, is a web-application framework made with Ruby. [10] A framework is like a library in other computer languages whitch contains code, tools & utilities to work with.[11] It is said that Rails is nowadays one of the best web-application frameworks alongside Angular, Express and Django which are also particularly good frameworks. [12] But let's focus on Rails and two most used testing suites in Rails. There is also a small example tests from both testing suites and small comparison why one is better than the other.

3.2.1 Rails

Ruby on Rails is a web-application framework made with Ruby. It is an open-source framework developed for database-based web-applications. Rails is made to be easy for everyone and is advertised to have “magic.” The “magic” is in fact a command line code that can make for example a simple view for the application. A Rails’ view is the front of the application in short everything visual in the application. It is possible to fine tune it with Cascading Style Sheet (CSS) so it does not need to look as rough as it might look at first.

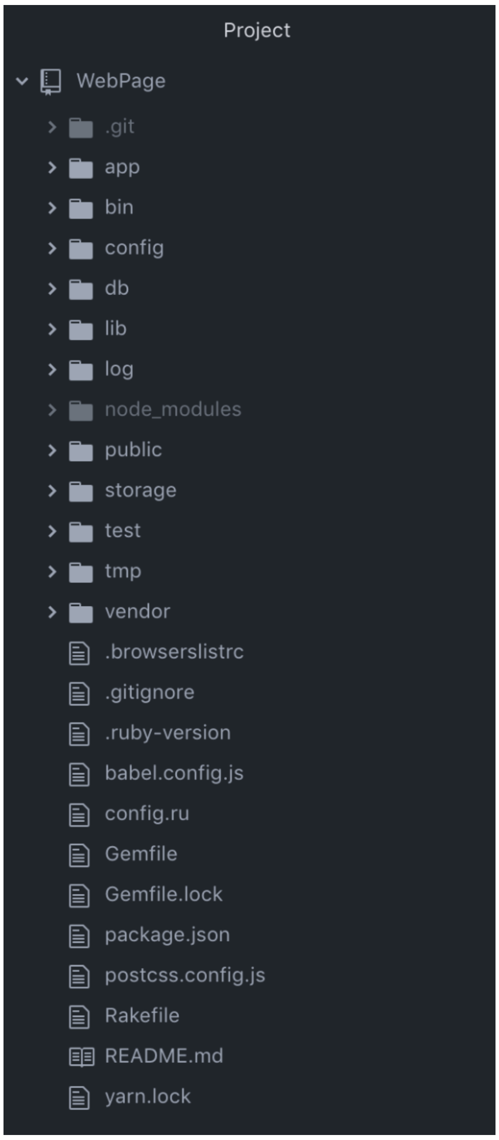
In Rails it is possible to create a quite simple static page, a web page that stays always the same unless someone changes them manually [13].



```
tuomass-mbp:Rails_testing Maamies$ rails new WebPage
      create
      create  README.md
      create  Rakefile
      create  .ruby-version
      create  config.ru
      create  .gitignore
      create  Gemfile
         run  git init from "."
Initialized empty Git repository in /Users/Maamies/Projects/Rails_testing/WebPage/.git/
      create  package.json
      create  app
      create  app/assets/config/manifest.js
      create  app/assets/stylesheets/application.css
      create  app/channels/application_cable/channel.rb
      create  app/channels/application_cable/connection.rb
      create  app/controllers/application_controller.rb
      create  app/helpers/application_helper.rb
      create  app/javascript/channels/consumer.js
      create  app/javascript/channels/index.js
```

Figure 3: Creating a simple web application.

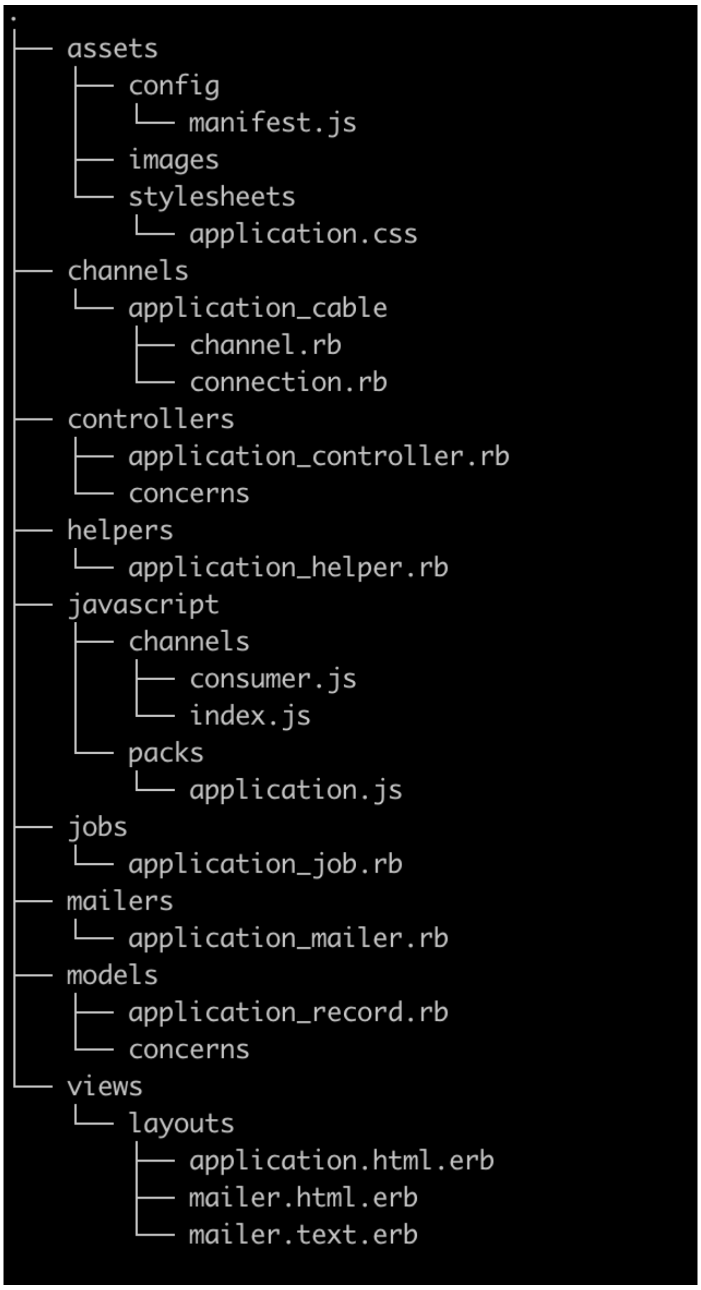
In figure 3 a simple Rails application was created with one line of code ´rails new WebPage´ that creates a basic Rails application named ´WebPage´, that is like a “Hello world” now, which is a simple application to test that everything is working as intended. The other ´create´ and ´run´ in the code are Rails’ way to show developers what it creates. Afterwards it might run automatically ´bundle install´ that installs all gems required to run the application. Gems are libraries that that eases the development. For example, Minitest and Rspec are gems that focuses on testing. They can be easily added and removed depending on if they are needed. One bad side about it is that in bigger projects people do not tend to know what gem is used where and the project can have hundreds of gems where each gem has vulnerabilities.



```
Project
WebPage
  .git
  app
  bin
  config
  db
  lib
  log
  node_modules
  public
  storage
  test
  tmp
  vendor
  .browserslistrc
  .gitignore
  .ruby-version
  babel.config.js
  config.ru
  Gemfile
  Gemfile.lock
  package.json
  postcss.config.js
  Rakefile
  README.md
  yarn.lock
```

Figure 4: Content Rails generates when creating a new project.

Running the code in figure 3 creates a base for the project and arranges folders like in figure 4. Each folder name describes what the folder contains. The most important folders are app, config, and test folders. App-folder contains most of the application. For example, all its front-end and some parts of configurations and models. Config-folder contains bigger configurations about the project. For example, how things are routed in REST API and database configurations. The test-folder contains the test files and testing configurations. There is also Gemfile-file in the root that contains a list of the gems that work like libraries for the application.



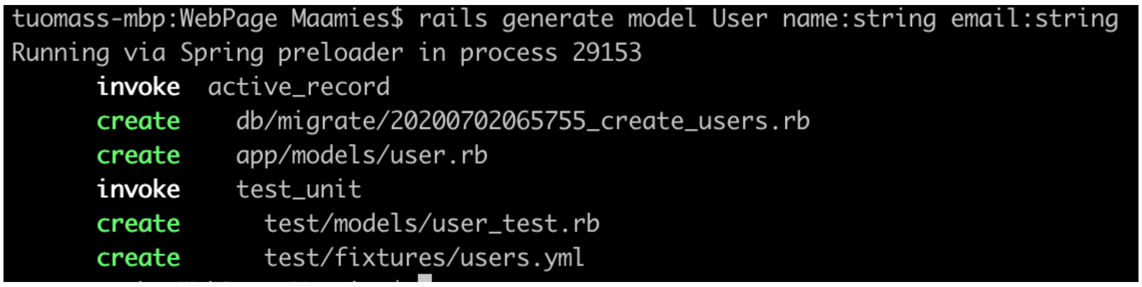
```
.
├── assets
│   ├── config
│   │   └── manifest.js
│   ├── images
│   └── stylesheets
│       └── application.css
├── channels
│   └── application_cable
│       ├── channel.rb
│       └── connection.rb
├── controllers
│   ├── application_controller.rb
│   └── concerns
├── helpers
│   └── application_helper.rb
├── javascript
│   ├── channels
│   │   ├── consumer.js
│   │   └── index.js
│   └── packs
│       └── application.js
├── jobs
│   └── application_job.rb
├── mailers
│   └── application_mailer.rb
├── models
│   ├── application_record.rb
│   └── concerns
└── views
    └── layouts
        ├── application.html.erb
        ├── mailer.html.erb
        └── mailer.text.erb
```

Figure 5: Contents of app-folder.

Figure 5 shows everything app-folder contains now. There are a few folders that needs to be understood what they do. Assets-folder contains most of the visuals. For example, most of the visuals in the front-page have been styled with files in assets-folder. In short it contains all CSS-files, images, and such if the application needs. Controllers-folder contains configurations how applications views and models communicate between each other. For example, when user sends an HTTP request controller will determinate which class and method it will use. Models-folder contains models whose purpose is to handle data between database and controller. For instance, it can get data from the database and developers could make it check if given value is in right format before saving it in the database. And in views-folder there is only the front-end of the application. So, the views-folder contains all buttons, input boxes and texts. [14]

3.2.2 Minitest

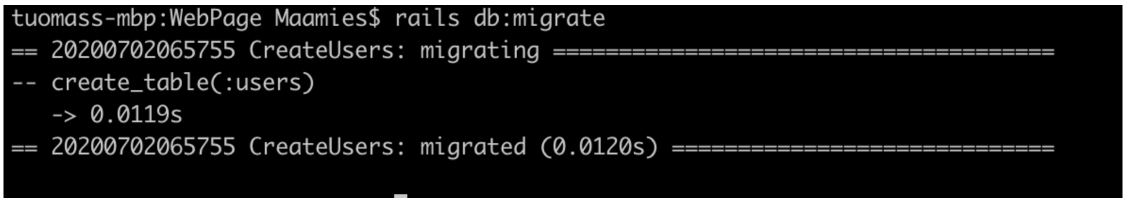
Minitest is a Rails gem that tests the application. It is also said that Minitest is Rails' default testing suite. [14] But let's see how it works in practice.



```
tuomass-mbp:WebPage Maamies$ rails generate model User name:string email:string
Running via Spring preloader in process 29153
      invoke  active_record
      create    db/migrate/20200702065755_create_users.rb
      create    app/models/user.rb
      invoke    test_unit
      create      test/models/user_test.rb
      create      test/fixtures/users.yml
```

Figure 6: Creating a simple User model.

Figure 6 is generating a simple user model for the application that was created in last chapter. For now, there is only name and email that is enough to show a little how things work. Now that there is a model, the application needs to be migrated. In short, the application needs to move its data to database. [16] The data now is just the models' information created in a Figure 6 that tells the database what kind of data it should have.

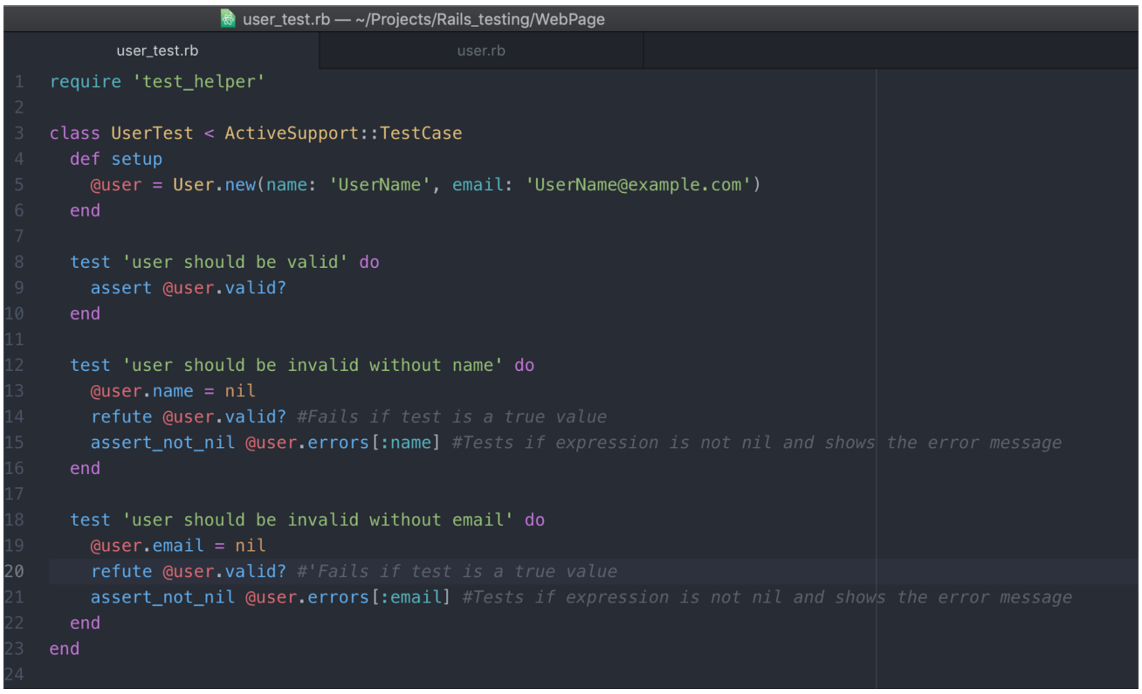


```
tuomass-mbp:WebPage Maamies$ rails db:migrate
== 20200702065755 CreateUsers: migrating ======================================
-- create_table(:users)
   -> 0.0119s
== 20200702065755 CreateUsers: migrated (0.0120s) =============================
```

Figure 7: Migrating a database.

In figure 7 the application creates small database that contains just the base for this application. For now, the tests could be still if true equals true or false consequently the application needs still more to truly show what testing is.

Making tests before developing the application is the idea of Test-Driven Development (TDD) and a recommended way to develop applications so let's make the small example in that order.

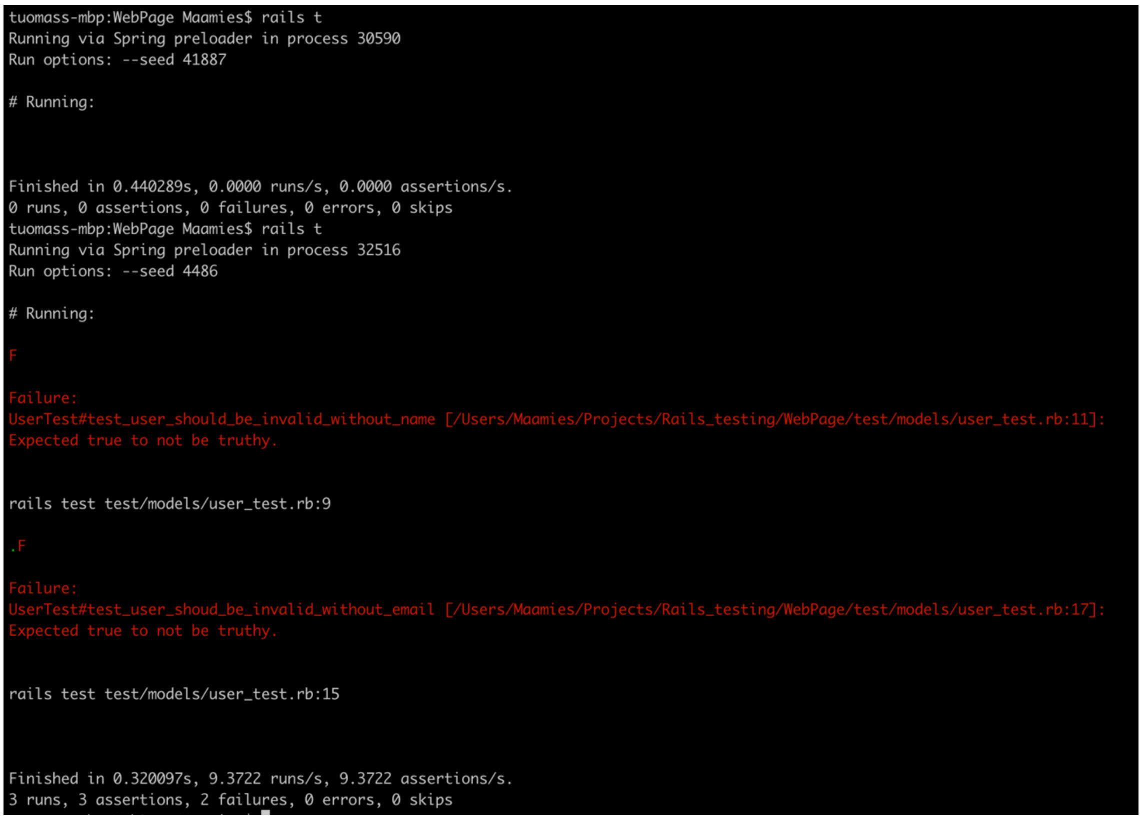

user_test.rb — ~/Projects/Rails_testing/WebPage

user_test.rb | user.rb

```ruby
require 'test_helper'

class UserTest < ActiveSupport::TestCase
  def setup
    @user = User.new(name: 'UserName', email: 'UserName@example.com')
  end

  test 'user should be valid' do
    assert @user.valid?
  end

  test 'user should be invalid without name' do
    @user.name = nil
    refute @user.valid? #Fails if test is a true value
    assert_not_nil @user.errors[:name] #Tests if expression is not nil and shows the error message
  end

  test 'user should be invalid without email' do
    @user.email = nil
    refute @user.valid? #'Fails if test is a true value
    assert_not_nil @user.errors[:email] #Tests if expression is not nil and shows the error message
  end
end
```

Figure 8: A few simple tests in './WebPage/test/models/user_test.rb'.

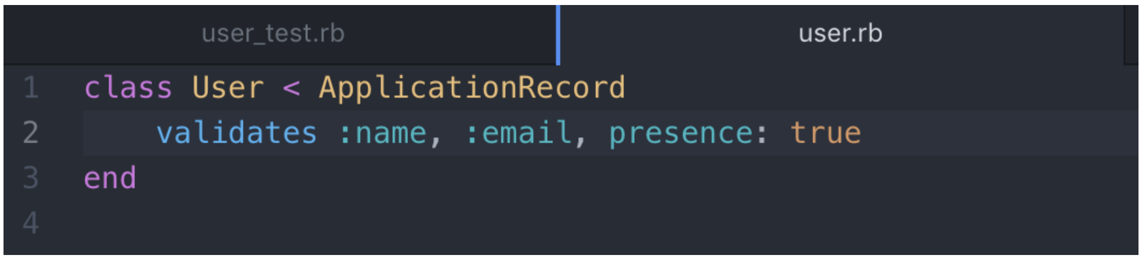
In figure 8 there is three simple tests that was created and a setup that initializes user. The tests 'user should be valid,' 'user should be invalid without name' and 'user should be invalid without email' are almost the same but tests different things. The test's name should give information what it is testing so it is easier for everyone to understand what is happening. For example, in these three, the first test tests if the user has valid name and password and rest are tests if either name or email is missing. For now, both name and email would be valid even though they are missing because there is not anything in the models to validate them. That would mean the last two test fails because the outcome is different than expected.



```
tuomass-mbp:WebPage Maamies$ rails t
Running via Spring preloader in process 30590
Run options: --seed 41887

# Running:



Finished in 0.440289s, 0.0000 runs/s, 0.0000 assertions/s.
0 runs, 0 assertions, 0 failures, 0 errors, 0 skips
tuomass-mbp:WebPage Maamies$ rails t
Running via Spring preloader in process 32516
Run options: --seed 4486

# Running:

F

Failure:
UserTest#test_user_should_be_invalid_without_name [/Users/Maamies/Projects/Rails_testing/WebPage/test/models/user_test.rb:11]:
Expected true to not be truthy.


rails test test/models/user_test.rb:9

.F

Failure:
UserTest#test_user_shoud_be_invalid_without_email [/Users/Maamies/Projects/Rails_testing/WebPage/test/models/user_test.rb:17]:
Expected true to not be truthy.


rails test test/models/user_test.rb:15



Finished in 0.320097s, 9.3722 runs/s, 9.3722 assertions/s.
3 runs, 3 assertions, 2 failures, 0 errors, 0 skips
```

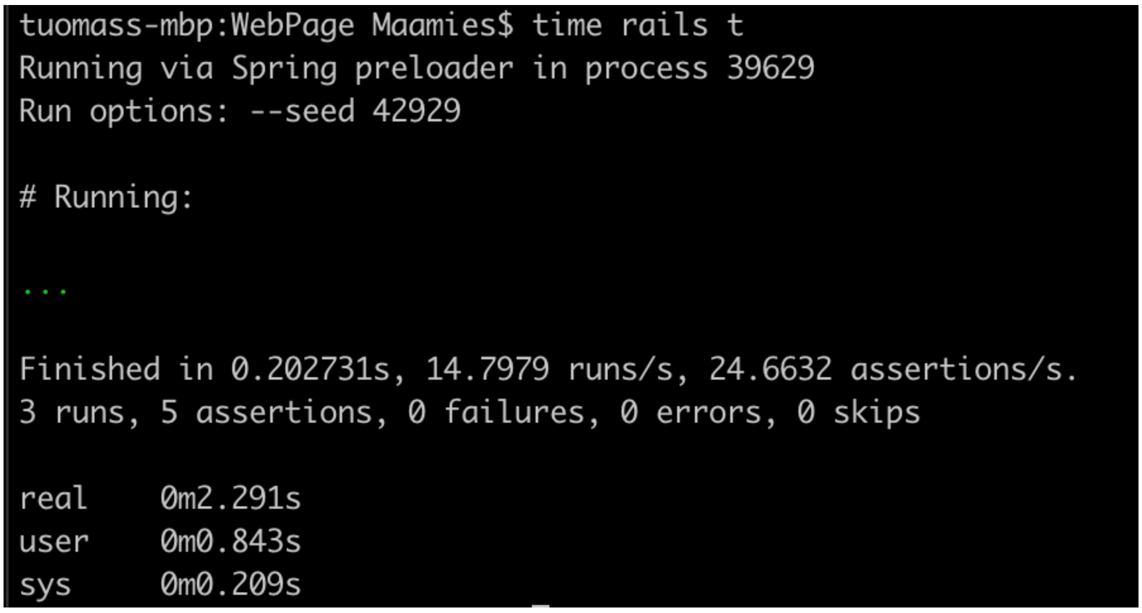
Figure 9: The test results.

In figure 9, the test was run with ‘rails t’-command which run rails’ Minitests, shows the expected result. The model validations that needs to be edited for the test can be found in ‘./WebApp/app/models/user.rb’. The validations should be simple for these tests because they should check if the users have given a name and email.



```ruby
class User < ApplicationRecord
    validates :name, :email, presence: true
end

```

Figure 10: Added validations in '.WebPage/app/models/user.rb'.



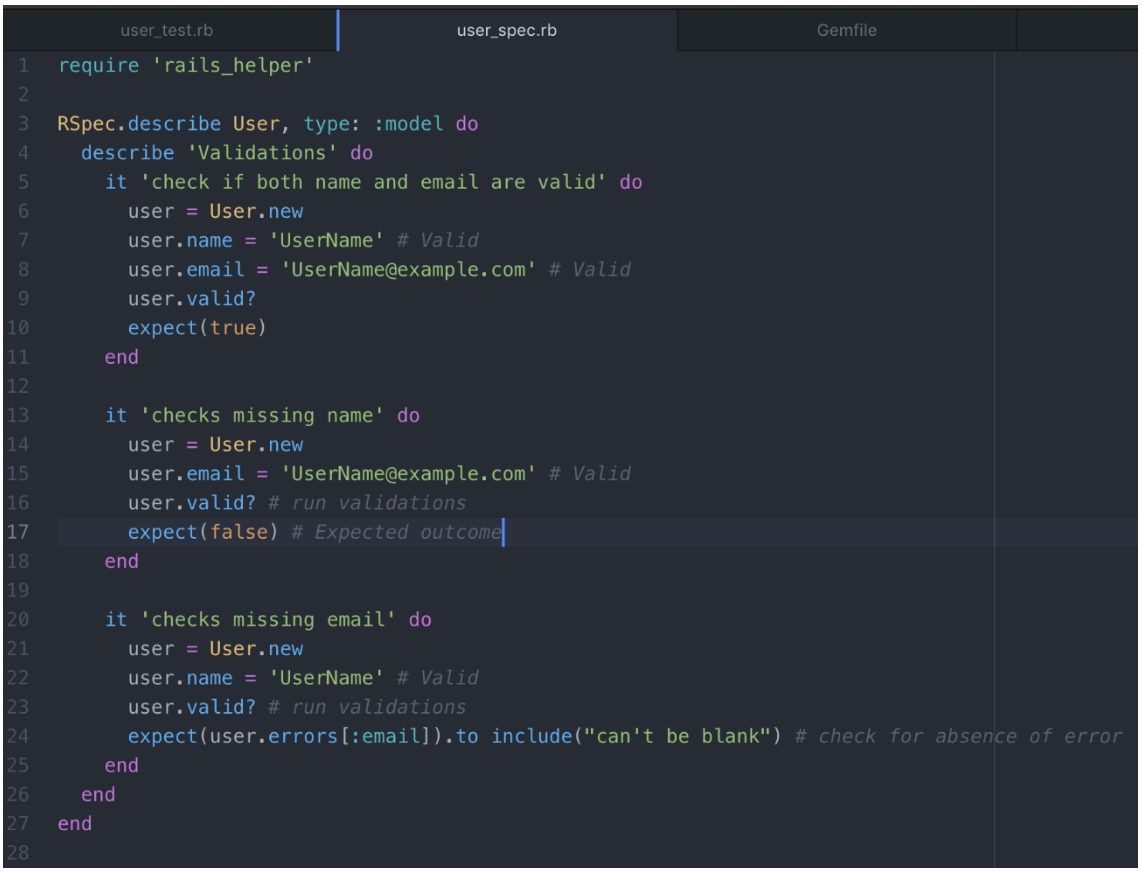
```
tuomass-mbp:WebPage Maamies$ time rails t
Running via Spring preloader in process 39629
Run options: --seed 42929

# Running:

...

Finished in 0.202731s, 14.7979 runs/s, 24.6632 assertions/s.
3 runs, 5 assertions, 0 failures, 0 errors, 0 skips

real    0m2.291s
user    0m0.843s
sys     0m0.209s
```

Figure 11: Example when all the tests have passed.

The line two of figure 10 is the added validations that are enough to make all the tests pass as figure 11 shows. These validations in figure 10 checks if name and email has a value and informs if they are missing it with an error message. After removing the name and email in the last two tests, the expected outcome is that they should not be valid. Figure 11 also shows the full time it takes to run all the tests.

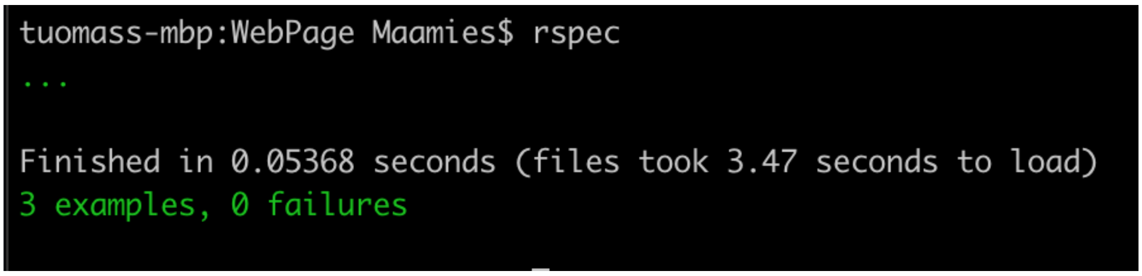
3.2.3 Rspec

Rspec is one of many alternative testing frameworks to Minitest on Rails. In Rspec the test are not only scripts but specifications. In other words, the tests are made to seem like it is in plain English. [17] Let’s try to make same tests that were made with Minitest.



```ruby
require 'rails_helper'

RSpec.describe User, type: :model do
  describe 'Validations' do
    it 'check if both name and email are valid' do
      user = User.new
      user.name = 'UserName' # Valid
      user.email = 'UserName@example.com' # Valid
      user.valid?
      expect(true)
    end

    it 'checks missing name' do
      user = User.new
      user.email = 'UserName@example.com' # Valid
      user.valid? # run validations
      expect(false) # Expected outcome
    end

    it 'checks missing email' do
      user = User.new
      user.name = 'UserName' # Valid
      user.valid? # run validations
      expect(user.errors[:email]).to include("can't be blank") # check for absence of error
    end
  end
end
```

Figure 12: Same kind of examples with Rspec as with Minitest.



```
tuomass-mbp:WebPage Maamies$ rspec
...

Finished in 0.05368 seconds (files took 3.47 seconds to load)
3 examples, 0 failures
```

Figure 13: Running the tests with Rspec.

Figure 12 shows same kind of tests that was made with Minitest but in Rspec. Every ‘it’ is a test the Rspec will run and tell if it passes or not. Figure 12 also shows two different outcome expectations where it is either true or false and error expectation. True and false are easy to guess how they work but error might be a little harder. And as figure 13 shows, all the tests have passed.

3.2.4 Differences

Both Minitest and Rspec are good testing frameworks that are nice to understand and know a little how they work. But it is also valuable to know the differences between them when deciding what test frameworks to use in a project.

Minitest, as the default testing framework runs faster than Rspec. For example, Figure 13 shows it took over three seconds for Rspec to load everything but in Minitest it took only a little over two seconds as the figure 11 shows. Also, when comparing figure 11 and figure 13 the Rspec Command Line Interface (CLI) looks more simpler compared to Minitest's CLI. It is also easier to run a specific test in Rspec than in Minitest. [18]

Minitest is also pure Ruby compared to Rspec which is Domain-Specific Language (DSL). DSL is a specialized programming language which is optimized for specific kind of problems. DSL is also usually less complex than general programming languages. [19]

3.3 Technologies in implementing microservice architecture.

Microservice architecture may sound as a good solution for everything but one of the challenges in microservice architecture is connecting every service together. There are multiple separate ways to connet every service to each other and with different methods how to connect. Here is information about two most used ways of implementing microservice architectures.

3.3.1 Broker

Broker is like a middle hand in microservice architecture. It connects all the services together and allows the services to communicate amongst themselves. Broker is only routing the messages it gets from services and sending them forward to other services if needed. This way the services do not need to know what other services are doing and can be isolated because the broker handles all the communication between them. This may lead to some drawbacks because the broker handles every connection between the services. However, that can be mitigated by having multiple brokers running parallel or having multiple different architectures in the application. [26]

3.3.2 Docker

Docker is a tool that is designed to handle applications with more flexibility. For instance, developers can create, deploy, and run applications as a container. Container is like one package that contains everything the application needs. [9]

One container is deployed after an image the developers have built or downloaded from Docker hub. Developers can run multiple containers at the same time. Container should have everything it needs to deploy properly. Practical example about Docker and containers would be that an application that is in a container which needs a database. Afterwards the application would need another container that contains the database. That means two different containers running at the same time. Other appeals to the docker are that each container is easily removeable and re-deployable.

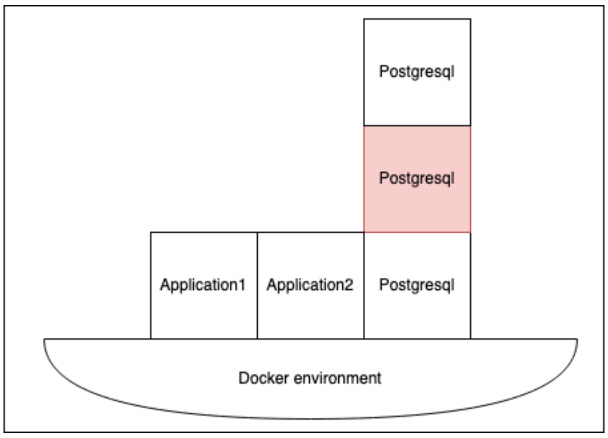


Figure 14: Docker environment with a few containers.

Figure 14 has a docker environment that has a few containers running on top of it. One square represents one container. The PostgreSQL is a database that could contain for example all product information. There are also two different applications that could use PostgreSQL or communicate between each other. Figure 14 also shows that the red PostgreSQL container is down and could be removed. There is also a PostgreSQL on top of the downed container that could have replaced the downed container.

Docker is a useful tool for microservice architecture because it can make one service as one container. It is also possible to deploy different containers depending on what services are needed to be running. There is also docker-compose that specifies larger environment a little bit better than normal docker. For example, it is possible to run all the services with only one line, and it would check what containers start and in what order they start.

Docker uses so called containers to hold the entire system. It creates a small virtual box-like environment for the containers where they can run according to the configurations. It could be said that Docker is made for microservice architecture because each service can run independently in a container and the system can run multiple containers at the same time. [27]

Docker can run over 10 000 containers at the same time. It means there is no need for other applications to connect the services because Docker would potentially run 10 000 different services. The problems will show up if some connections have faults and the machine is not powerful enough. [28]

3.4 Technical comparison

None of the continuous integration tools are so much better from each other that it becomes necessary but there are limitations like if company's security policies or if it fits the company's Version Control System (VCS). Since the VCS is GitLab in this case the GitLab CI would be recommended because it would be more compatible with the repositories and gives the possibility to develop further.

Rails' Minitest and Rspec are both valid for making tests but the biggest decisions are either speed or readability. The speed is generally good because the team could start developing faster without much loupe time but in this case let's use Rspec because it is more readable.

4. CONTINUOUS INTEGRATION IN MICROSERVICE ARCHITECTURE

This chapter presents creating docker environment for microservice architecture and creating GitLab CI for one service. The CI part is dedicated for one service, but it could be used for other similar services inside a project. The dockerization or monorepo is not necessary in continuous integration but it might help developers in the future when they can get the whole project in one package and can get the project working as intended faster.

4.1 Creating a monorepo

Monorepo is a repository that has multiple different repositories. It would ease developers to create docker-compose environment in the future and depending how they want to make the Continuous Integration.

Making monorepo is surprisingly easy with the git. The only problems will come later when developers want to keep every service updated while other developers are pushing code to different branches. But to make the environment, initialize the project folder as a monorepo. The code to create a repository is ‘git init’. Git init will create a new git directory which will be the base of the monorepo. And to add a new service for that repository run ‘git submodules add “project service URL”’. The git will get the service’s master branch commit id which it will know what to get.

After creating the base for the monorepo push the project into a repository. This will be the base for the future development. For example, making docker-compose environment in the later that would help to run the tests.

4.2 Tests in microservice architecture and database

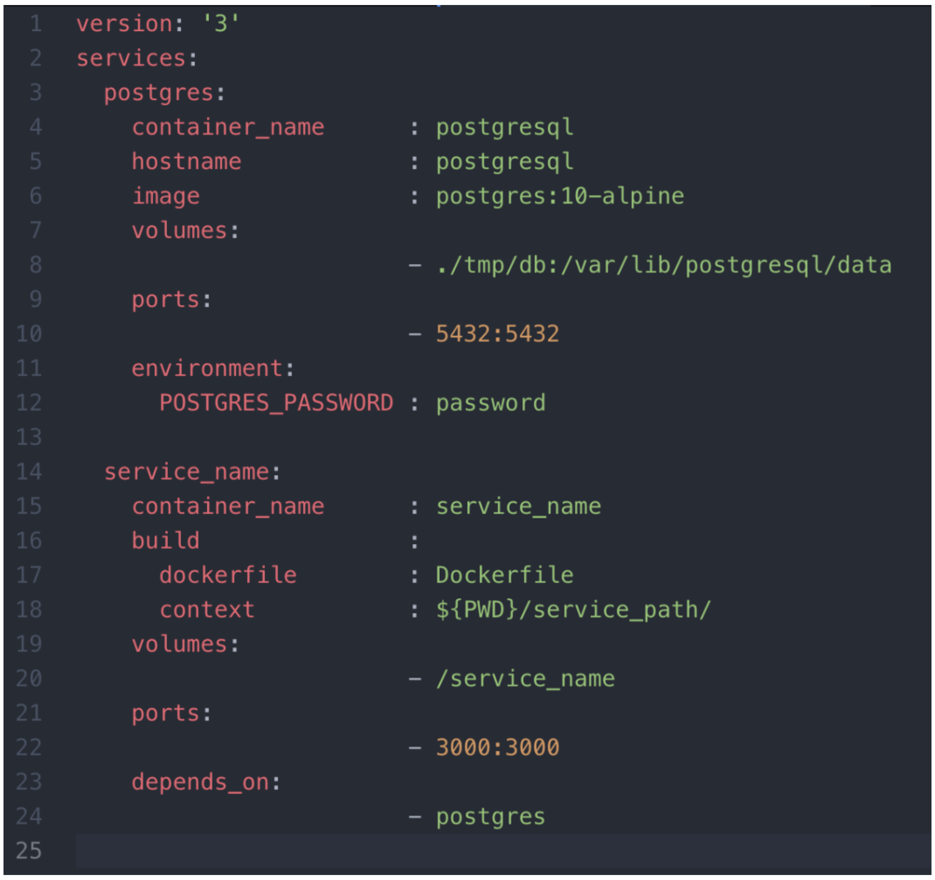
Testing a single service in microservice architecture works similarly than in monolith architecture projects. A project build with monolith method is a single unified project. The biggest challenge in microservices is how to integrate the main service to a database. Otherwise, it works on similar commands.

After running ‘rspec’ or ‘bundle exec rspec’ in one service, there will only be errors. For example, the error could be “PG::Connection Bad:” and “An error occurred while loading ‘file-path’”. Especially the first error means that the project could not get any connection to the database that is in this case PostgreSQL. The second error comes because it could not get any connections to the database that the file in the specific path needed.

At the moment the database cannot connect to the service so that will need to be fixed. There are multiple ways of connecting database to the service. One would be creating a local PostgreSQL server or to make a docker-compose environment which can start the whole application with one line. Docker-compose environments would be good for the projects because developers could add a new service easily and it eases for new developers to come into the project when they do not need to configure every service manually every time.

4.3 Making a docker-compose environment.

To make a docker-compose environment from microservices architecture there needs to be a ‘docker-compose.yml’ or ‘docker-compose.yaml’ file at the base of the project, in the folder that contains all the services. It should contain docker configurations. For example, what project image it is going to use as a base for the service or the database and what port it is going to use.

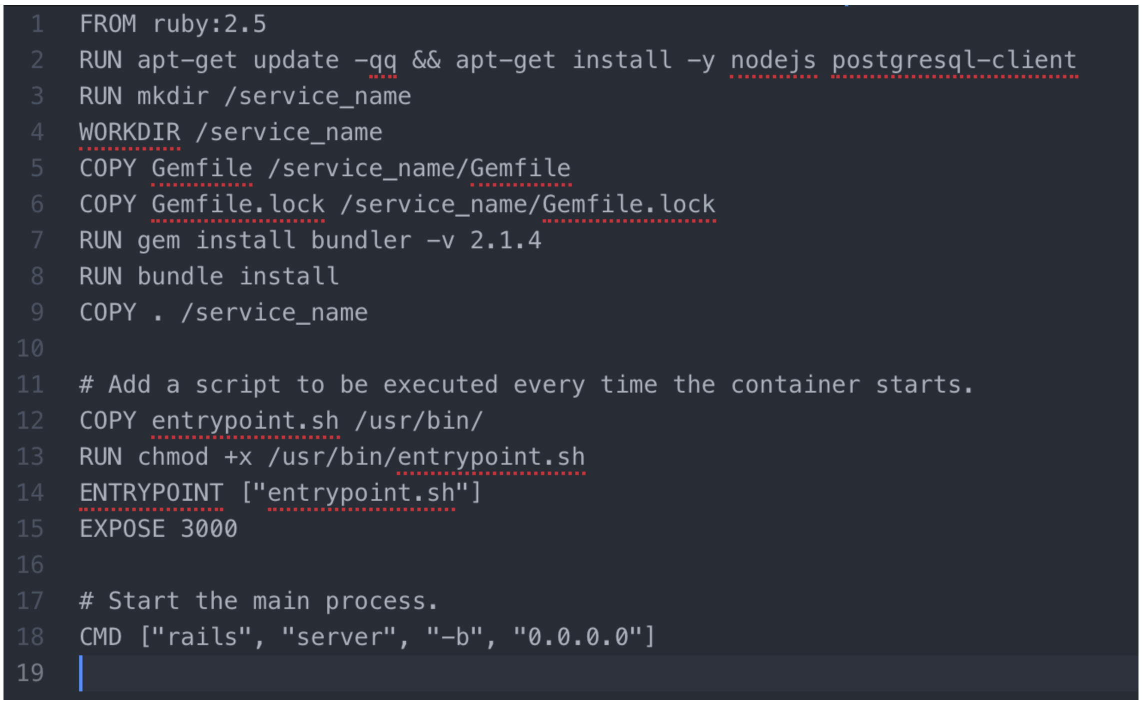


```yaml
version: '3'
services:
  postgres:
    container_name      : postgresql
    hostname            : postgresql
    image               : postgres:10-alpine
    volumes:
                        - ./tmp/db:/var/lib/postgresql/data
    ports:
                        - 5432:5432
    environment:
      POSTGRES_PASSWORD : password

  service_name:
    container_name      : service_name
    build               :
      dockerfile        : Dockerfile
      context           : ${PWD}/service_path/
    volumes:
                        - /service_name
    ports:
                        - 3000:3000
    depends_on:
                        - postgres

```

Figure 15: Example of a docker-compose.yaml.

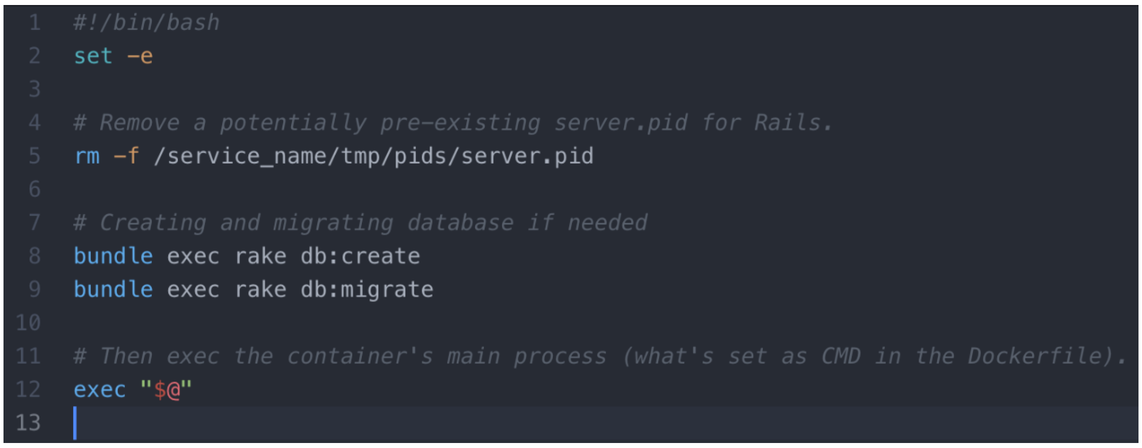
Figure 15 has an example of docker-compose.yaml. It specifies the file version first and then starts the service configurations. First there is the Postgres because it is the database that the service requires its capability of storing data. After that is the service and its configurations. The “service_name” and “service_path” should be different depending on the service and its name. It would be also recommended to create a network between services so they can communicate better between each other.

Docker-compose file is not enough to make everything work but after adding Dockerfile and entrypoint.sh to the service should work. Dockerfile itself could be enough but the entrypoint.sh gives the docker container better specifications like how to migrate the database.



```
FROM ruby:2.5
RUN apt-get update -qq && apt-get install -y nodejs postgresql-client
RUN mkdir /service_name
WORKDIR /service_name
COPY Gemfile /service_name/Gemfile
COPY Gemfile.lock /service_name/Gemfile.lock
RUN gem install bundler -v 2.1.4
RUN bundle install
COPY . /service_name

# Add a script to be executed every time the container starts.
COPY entrypoint.sh /usr/bin/
RUN chmod +x /usr/bin/entrypoint.sh
ENTRYPOINT ["entrypoint.sh"]
EXPOSE 3000

# Start the main process.
CMD ["rails", "server", "-b", "0.0.0.0"]
```

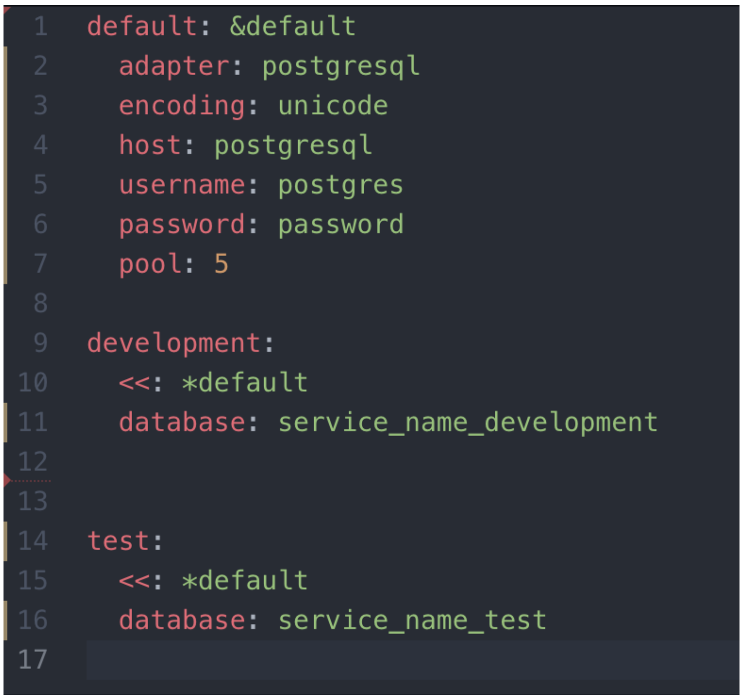
Figure 16: Example content for Dockerfile.



```
#!/bin/bash
set -e

# Remove a potentially pre-existing server.pid for Rails.
rm -f /service_name/tmp/pids/server.pid

# Creating and migrating database if needed
bundle exec rake db:create
bundle exec rake db:migrate

# Then exec the container's main process (what's set as CMD in the Dockerfile).
exec "$@"
```

Figure 17: Example content for entrypoint.sh.

Figure 16 is an example of Dockerfile and Figure 17 is an example of entrypoint.sh. They both contain “service_name” that should be the name of the service to be tested. Both of those files should also be in the root folder of the service and each service should have them both. The Dockerfile only tells how the docker image of the service is built and the image will be built along the instructions in the Dockerfile. For this example, the Dockerfile creates and updates the software, gems & directories needed for the image. Entrypoint.sh is for finetuning and additional commands to run before starting the service’s server.



```yaml
default: &default
  adapter: postgresql
  encoding: unicode
  host: postgresql
  username: postgres
  password: password
  pool: 5

development:
  <<: *default
  database: service_name_development


test:
  <<: *default
  database: service_name_test

```

Figure 18: Linking service to database.

And to fully link the service to the PostgreSQL change the ~service_name/config/database.yml contents to link to the database. Figure 18 has a small configurations example how a service is linked.

After everything is connected the tests can still be created with rspec or minitests except they must be run with docker container. So instead of running tests with only ‘rspec’ run them with ‘docker-compose run service_name bundle exec rspec’. It will create everything necessary for the test environment and start running the tests according to the instructions.

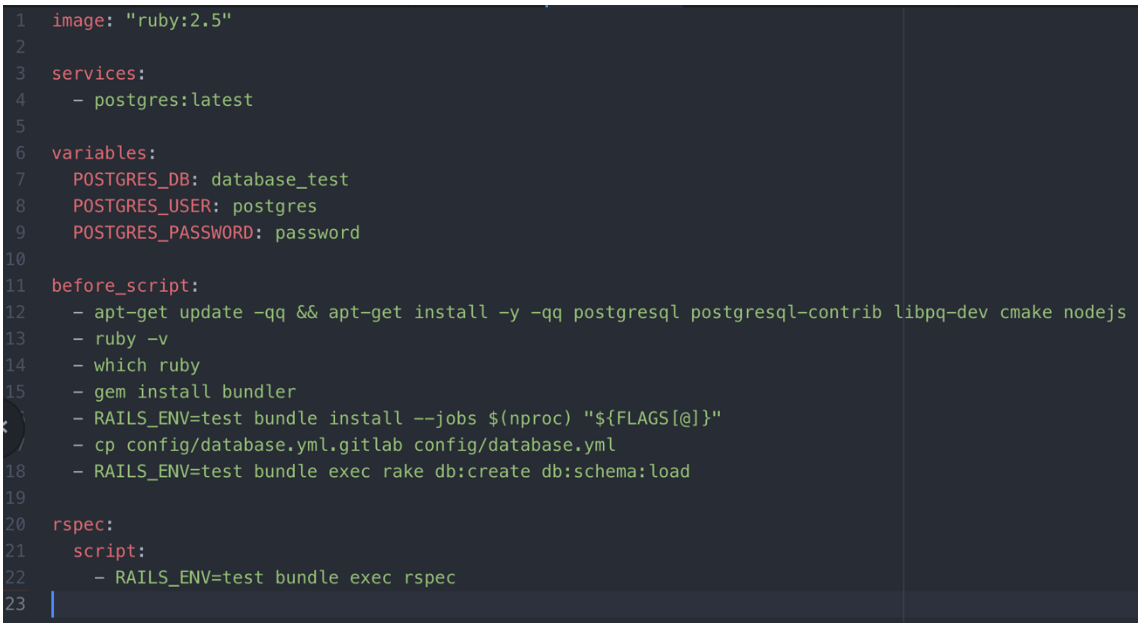
4.4 Setting up GitLab CI.

To set up a GitLab CI the GitLab Runner is needed. GitLab Runner is an open-source project that runs the test jobs and returns them back to GitLab. Those jobs could be for example that the runner builds the application and runs the tests in in the background. The Runner is used in GitLab CI/CD to coordinate the jobs. GitLab Runner is designed to run in most used operating systems like Windows, GNU/Linux and macOS. The Runner also requires Docker to run. [24] Gitlab has some runners that can be used automatically but it is good to have own runner because developers can change its settings more easily. Runner is a program that will run the tests on the background after a developer has pushed code to the repository.

There are multiple ways to install Gitlab Runner. One is to install it from GitLab's own repository. Secondly developers could use GitLab Runner as Docker service and tell it what to do. And there is a binary file that compiles the code itself.

To make a docker container run 'docker run -d --name gitlab-runner --restart always \ -v /srv/gitlab-runner/config:/etc/gitlab-runner \ -v /var/run/docker.sock:/var/run/docker.sock \ gitlab/gitlab-runner:latest' This will install and create docker container for runner that will run all the tests. After installing the gitlab-runner it will need to be registered to the repository. Run 'docker run --rm -it -v gitlab-runner-config:/etc/gitlab-runner gitlab/gitlab-runner:latest register'. Then it will ask about gitlab-ci coordinator URL, in short, repository's URL. Then it will ask about the token it has given. Afterwards it will ask a name the runner and tags to specify the runner. These will determinate what the runner is and the runner executioner, that is docker in this case. And to finish the registration add default image for the runner. That can be for example alpine: latest. Developers can also define different default image in .gitlab-ci.yml file that determinate what the CI is going to do. [29]

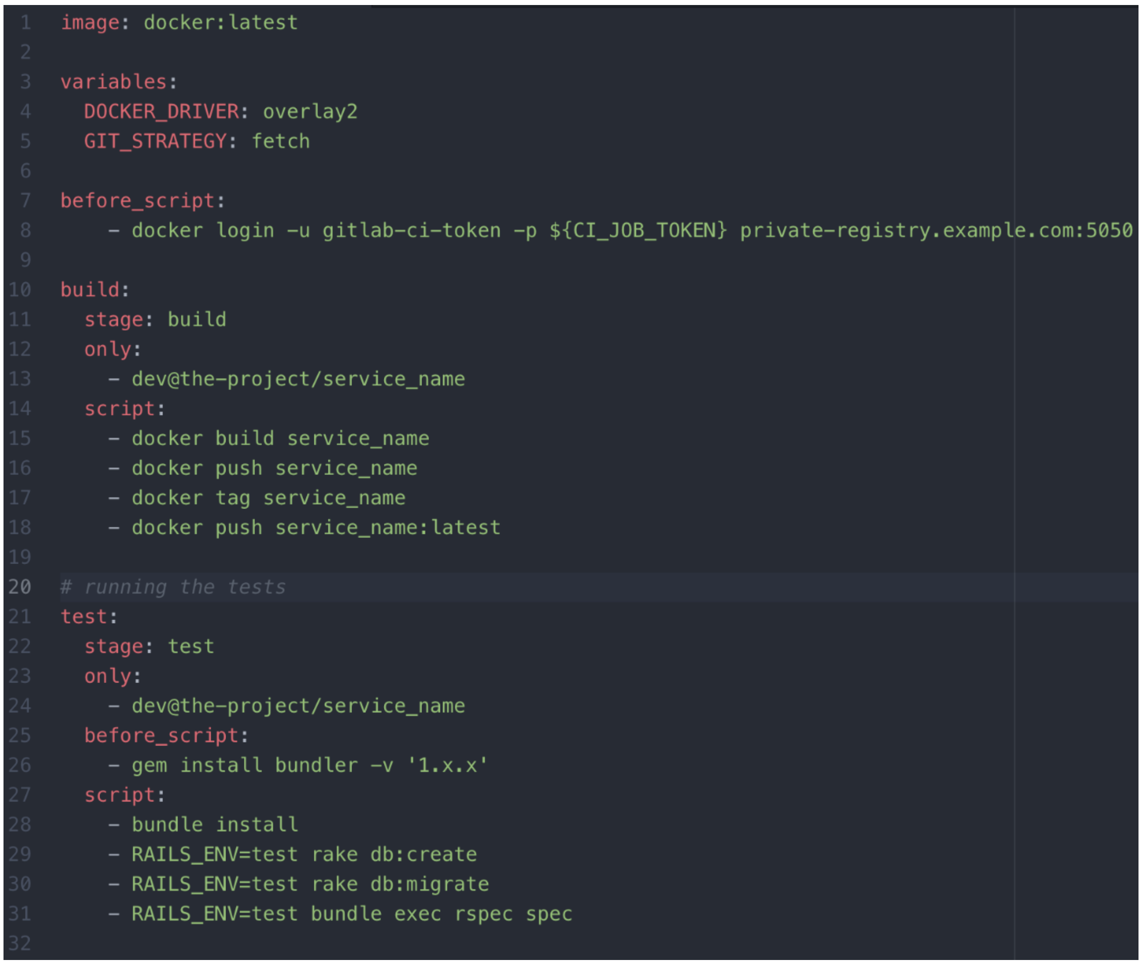
The first thing to do when adding GitLab CI is to add .gitlab-ci.yml file in the project's root folder. When a developer pushes code to the repository GitLab will look at the file and start running the GitLab Runner according the .gitlab-ci.yml file. And because the .gitlab-ci.yml is in the repository and the repository is version controlled the old builds are still running well even in the old pipelines. [25]



```yaml
image: "ruby:2.5"

services:
  - postgres:latest

variables:
  POSTGRES_DB: database_test
  POSTGRES_USER: postgres
  POSTGRES_PASSWORD: password

before_script:
  - apt-get update -qq && apt-get install -y -qq postgresql postgresql-contrib libpq-dev cmake nodejs
  - ruby -v
  - which ruby
  - gem install bundler
  - RAILS_ENV=test bundle install --jobs $(nproc) "${FLAGS[@]}"
  - cp config/database.yml.gitlab config/database.yml
  - RAILS_ENV=test bundle exec rake db:create db:schema:load

rspec:
  script:
    - RAILS_ENV=test bundle exec rspec

```

Figure 19: Example of '.gitlab-ci.yml'.

In figure 19's example is a simple CI example for an application which database is PostgreSQL. First it creates a base for the applications, connects to the database and migrates the database with pre-constructed data. It will run the 'before_script' part for each task it is going to run. For example, it will run the ‘before_script’ before rspec in this case but there could be other parts like ‘build’ or other code quality check parts where it would be running also. Of course, the gitlab-ci would return error any point if it fails to build the app or the tests do not pass. Figure 19 shows the minimum amount of CI for one application. There could be also Rubocop or Pronto that would check code style and quality. Both rubocop and Pronto are libraries for Rails.



```yaml
image: docker:latest

variables:
  DOCKER_DRIVER: overlay2
  GIT_STRATEGY: fetch

before_script:
    - docker login -u gitlab-ci-token -p ${CI_JOB_TOKEN} private-registry.example.com:5050

build:
  stage: build
  only:
    - dev@the-project/service_name
  script:
    - docker build service_name
    - docker push service_name
    - docker tag service_name
    - docker push service_name:latest

# running the tests
test:
  stage: test
  only:
    - dev@the-project/service_name
  before_script:
    - gem install bundler -v '1.x.x'
  script:
    - bundle install
    - RAILS_ENV=test rake db:create
    - RAILS_ENV=test rake db:migrate
    - RAILS_ENV=test bundle exec rspec spec

```

Figure 20: Building orders for docker.

It is also possible to create a docker container before running the tests. This would mean creating the container in the build-phase and test the said container in the test-phase and lastly it could be possible to deploy staging or production later if wanted. Figure 20 shows the order how it could be built for docker and further development. First log in to the docker hub or private docker registry. After that specify the project to be edited. Then build the said project push it to the docker registry and tag its version. In this case it would be the latest and push it with the said tag. After that migrate the test environment database and run the test and check if they pass. [30]

Adding the docker to the tests would be kind of a “true” CI because in the project there would be different versions of the application in gitlab and in docker repositories expect docker creates a latest version every time someone pushes code in gitlab. Or it would depend how they are configured because there can have different version .gitlab-ci.yml in master than in development branch.

5. CONCLUSION

Tests should be an important part of development. Most of the time they might be forgotten because developers, team leaders and customers want to get something out fast as possible not always caring how it looks like or how it is implemented. That may lead to regression and bad outcomes overall. This is especially true in larger projects that have multiple developers working on various parts of the project.

There are multiple methods of to implementing continuous integration and even more methods in microservice architecture. For example, developers could use different testing methods and CI tools for each service but that might not be the wisest decision because it might give some complications between the services and the developers would not be able to use the same files for each service. Each project is different and not everything may work as other projects that are developed similarly.

The purpose of this thesis was to give more consideration to testing and give an idea of how to implement continuous integration for future projects. It also demonstrates how to make docker-compose environment to combine services that might help some developers that are familiar with Docker. GitLab repositories are overall good, and many developers use git repositories, so it is useful to have a basic understanding of how GitLab CI/CD functions and how it is implemented.

The greatest restrictions in this thesis is that all is integrated with Ruby on Rail and for example tests will not work on another languages. The advantages are that creating GitLab and Docker environments should work on most of other projects and it does not depend on any specific computer language.

Improvements in this thesis could have been that there could be implemented a continuous deployment at the same time with the continuous integration. It would be an improvement if the continuous integration would run on all of the tests on the whole application instead of one service at a time.

REFERENCES

[1] Microsoft 2019, *CI/CD for microservice architecture*, [Online]
Available at: https://docs.microsoft.com/en-us/azure/architecture/microservices/ci-cd
[Accessed 4th June 2020]

[2] Alibaba Cloud 2017, *Continuous Deployment with Microservices*, [Online]
Available at: https://medium.com/@Alibaba_Cloud/continuous-deployment-with-microservices-f259dcc60618
[Accessed 4th June 2020]

[3] Tom Huston 2020, *What is Microservices*, [Online]
Available at: https://smartbear.com/solutions/microservices/
[Accessed 11th June 2020]

[4] Chris Richardson 2018, *Pattern: Microservice Architecture*, [Online]
Available at: https://microservices.io/patterns/microservices.html
[Accessed 4th June 2020]

[5] StreamSets 2020, *Microservice Pipelines*, [Online]
Available at:
https://streamsets.com/documentation/datacollector/latest/help/datacollector/UserGuide/Microservice/Microservice_Title.html
[Accessed 11th June 2020]

[6] Sukoreno Mukti 2018, *Why TDD (Test Driven Development)?*, [Online]
Available at: https://medium.com/@sukorenomw/why-tdd-test-driven-development-a1bc983a2cc0
[Accessed 11th June 2020]

[7] Martin Fowler 2006, *Continuous Integration*, [Online]
Available at: https://martinfowler.com/articles/continuousIntegration.html
[Accessed 11th June 2020]

[8] Max Rehkopf 2020, *What is Continuous Integration*, [Online]
Available at: https://www.atlassian.com/continuous-delivery/continuous-integration
[Accessed 11th June 2020]

[9] Red Hat 2019, *What is Docker?,* [Online]
Available at: https://opensource.com/resources/what-docker
[Accessed 18th June 2020]

[10] Tutorials point 2020, *Ruby on Rails – Introduction*, [Online]
Available at: https://www.tutorialspoint.com/ruby-on-rails/rails-introduction.htm
[Accessed 25th June 2020]

[11] Jesus Castello 2018, *What is Ruby on Rails & Why Is It Useful?*, [Online]
Available at: https://www.rubyguides.com/2018/10/what-is-ruby-on-rails/
[Accessed 25th June 2020]

[12] Aman Goel 2020, *10 Best Web Development Frameworks*, [Online]
Available at: https://hackr.io/blog/top-10-web-development-frameworks-in-2020
[Accessed 2nd July 2020]

[13] Maneesh Kumar Singh 2019, *Difference between Static and Dynamic Web Pages*, [Online]
Available at: https://www.geeksforgeeks.org/difference-between-static-and-dynamic-web-pages/
[Accessed 25th June 2020]

[14] Aweys Ahmed 2020, *What is Model-View-Controller (MVC)? (Ruby on Rails perspective)*, [Online]

Available at: https://dev.to/aweysahmed/what-is-model-view-controller-mvc-ruby-on-rails-perspective-3d85
[Accessed 25th June 2020]

[15] Heidar Bernhardsson 2016, *How to Test Rails Models with Minitest*, [Online]
Available at: https://semaphoreci.com/community/tutorials/how-to-test-rails-models-with-minitest
[Accessed 2nd July 2020]

[16] Garret Alley 2019, *What is Database Migration?*, [Online]
Available at: https://www.alooma.com/blog/what-is-database-migration
[Accessed 2nd July 2020]

[17] Phil Pirozhkov & 62 Others 2020, *rspec-rails*, [Online]
Available at: https://github.com/rspec/rspec-rails
[Accessed 2nd July 2020]

[18] Vinicius Stock 2019, *Rspec or Minitest for testing Rails app*, [Online]
Available at: https://dev.to/truggeri/rspec-or-minitest-for-testing-rails-apps-42fi/comments
[Accessed 9th July 2020]

[19] Jet Brains 2020, *Domain-Specific Languages*, [Online]
Available at: https://www.jetbrains.com/mps/concepts/domain-specific-languages/
[Accessed 9th July 2020]

[20] Martin Heller 2020, *What is Jenkins? The CI server explained*, [Online]
Available at: https://www.infoworld.com/article/3239666/what-is-jenkins-the-ci-server-explained.html
[Accessed 10th July 2020]

[21] Circleci 2020, *About Circleci*, [Online]
Available at: https://circleci.com/docs/2.0/about-circleci/
[Accessed 10th July 2020]

[22] Ankesh K 2018, *What is Teamcity and what are its benefits?*,[Online]
Available at: https://www.linuxnix.com/what-is-teamcity-and-what-ar e-its-benefits/
[Accessed 10th July 2020]

[23] GitLab 2020, *GitLab Continuous Integration (CI) & Continuous Delivery (CD)*, [Online]
Available at: https://about.gitlab.com/stages-devops-lifecycle/continuous-integration/
[Accessed 16th July 2020]

[24] GitLab 2019, *GitLab Runner Docs*, [Online]
Available at: https://docs.gitlab.com/runner/
[Accessed 16th July 2020]

[25] GitLab 2020, *Getting started with GitLab CI/CD*, [Online]
Available at: https://docs.gitlab.com/ee/ci/quick_start/
[Accessed 16th July 2020]

[26] Francisco Temudo 2016, *How to Set up a Microservices Architecture in Ruby: A Step by Step Guide*, [Online]
Available at: https://www.toptal.com/ruby/how-to-set-up-a-microservices-architecture
[Accessed 17th July 2020]

[27] Amanda Fawcett 2020, *Microservices Architecture Tutorial: all you need to get started*, [Online]
Available at: https://www.educative.io/blog/microservices-architecture-tutorial-all-you-need-to-get-started
[Accessed 23rd July 2020]

[28] Seetharami Seelam 2015, *Docker at insane scale on IBM Power Systems*, [Online]
Available at: https://www.ibm.com/cloud/blog/docker-insane-scale-on-ibm-power-systems

[Accessed 23rd July 2020]

[29] GitLab Docs, Registering Runners, [Online]
Available at: https://docs.gitlab.com/runner/register/index.html
[Accessed 30th July 2020]

[30] Aviad Levy 2019, CI/CD of microservices architecture with GitLab, [Online]
Available at: https://medium.com/@aviadlevy/ci-cd-of-microservices-architecture-with-gitlab-fa9330bd32d0
[Accessed 6ht August 2020]

[31] GitLab Docs, GitLab CI/CD include examples, [Online]
Available at: https://docs.gitlab.com/ee/ci/yaml/includes.html
[Accessed 13th August 2020]

[32] Joe Nemer 2019, *Advantages and Disadvantages of Microservice Architecture, [*Online]
Available at: https://cloudacademy.com/blog/microservices-architecture-challenge-advantage-drawback/
[Accessed 28th January 2021]