National College of Ireland

Technical Report

John O'Grady | BSCHE | 14101718

X14101718@student.ncirl.ie
2017/2018

Contents

Declaration Cover Sheet for Project Submission

SECTION 1 *Student to complete*

Name: **John O'Grady**
Student ID: **14101718**
Supervisor: **Padraic DeBurca**

SECTION 2 Confirmation of Authorship

The acceptance of your work is subject to your signature on the following declaration:

I confirm that I have read the College statement on plagiarism (summarised overleaf and printed in full in the Student Handbook) and that the work I have submitted for assessment is entirely my own work.

Signature: John O'Grady. Date: 13/05/2018

NB. If it is suspected that your assignment contains the work of others falsely represented as your own, it will be referred to the College's Disciplinary Committee. Should the Committee be satisfied that plagiarism has occurred this is likely to lead to your failing the module and possibly to your being suspended or expelled from college.

Complete the sections above and attach it to the front of one of the copies of your assignment,

What constitutes plagiarism or cheating?

The following is extracted from the college's formal statement on plagiarism as quoted in the Student Handbooks. References to "assignments" should be taken to include any piece of work submitted for assessment.

Paraphrasing refers to taking the ideas, words or work of another, putting it into your own words and crediting the source. This is acceptable academic practice provided you ensure that credit is given to the author. Plagiarism refers to copying the ideas and work of another and misrepresenting it as your own. This is completely unacceptable and is prohibited in all academic institutions. It is a serious offence and may result in a fail grade and/or disciplinary action. All sources that you use in your writing must be acknowledged and included in the reference or bibliography section. If a particular piece of writing proves difficult to paraphrase, or you want to include it in its original form, it must be enclosed in quotation marks

and credit given to the author.

When referring to the work of another author within the text of your project you must give the author's surname and the date the work was published. Full details for each source must then be given in the bibliography at the end of the project

Penalties for Plagiarism

If it is suspected that your assignment contains the work of others falsely represented as your own, it will be referred to the college's Disciplinary Committee. Where the Disciplinary Committee makes a finding that there has been plagiarism, the Disciplinary Committee may recommend

- that a student's marks shall be reduced
- that the student be deemed not to have passed the assignment
- that other forms of assessment undertaken in that academic year by the same student be declared void
- that other examinations sat by the same student at the same sitting be declared void

Further penalties are also possible including

- suspending a student college for a specified time,
- expelling a student from college,
- prohibiting a student from sitting any examination or assessment.,
- the imposition of a fine and
- the requirement that a student to attend additional or other lectures or courses or undertake additional academic work.

1 Introduction

1.1 Background

The background to this project is set out to overcome the lack of online security tools available for small to medium sized web development companies. Currently, there are few applications online that would be affordable by small and medium sized web development/design companies. There are companies out there that will perform security scans of a company's website, but they are quite expensive, and are only geared towards larger enterprises, so smaller companies may not be able to afford their service. By developing this tool, companies will be able to use an online service where they can perform a web vulnerability scan on a web server. This will help with costs, and help companies keep on top any security issues.

1.2 Aims

The aim of this project is to develop an application, where a user can supply their public IP address or domain name, then using the system they can initiate or schedule scans on their website. The system will run a scan using easily available, and custom tools to perform scans like port scans, header inspection, XML sitemap, SQL injection vulnerabilities, and Nikto, which will build a list of all possible know vulnerabilities that may exist on the customer's web server. Using this information, customer will be able to implement patches and fixes to address the issues found during the scan.

1.3 Technologies

The technologies used in this project are python, MySQL, PHP, HTML/CSS, JavaScript, python XML sitemap builder, and Linux Command Line tools.

The main application is to be developed using the Python programming language. This application will run on a back-end server and will be completely seamless. This means that if another instance of the application is run on another server, it can start running and executing jobs without any additional configuration. This makes the application extremely scalable and using a service like AWS auto-scaling the application can really run at its greatest potential.

The front-end web application is to be designed using HTML, CSS, and JavaScript. The front-end application is what the user will interact with. They will never interact with the backend Python application. By using the front-end users can interact with the backend API, which will handle all requests from the front-end application, and the python backend application. Here, users can account, add a domain/IP address to their dashboard, and start immediate or scheduled scans.

The tools used for the actual scan are as follow:

- Nmap – Used for scanning a domain or IP address for open ports. This data is presented to the user, so they can close any unnecessary ports.
- Curl – Command line URL tool. This tool used to make HEAD requests to the intended target and retrieve the list of headers returned from the web-server. These headers are inspected, and the user is prompted to remove any headers that the application deems not useful or exposes potential security risks.
- Nikto – A tool used by penetration testers to find a list of known vulnerabilities with a web server. This output is captured and presented to the user, so they can patch/fix any possible issues.
- Sqlmap – Used for testing possible SQL injection vulnerabilities on a customer's application.
- Openssl – this is used to obtain SSL information on a customer's SSL certificate, and provide a grading on the information returned from this tool.

1.4 Structure

1. Introduction – An introduction to the reasons behind creating this application, and what the general idea of the application tends to solve.
2. Architecture – The architecture of the application, and how each element interacts with each other to deliver the service the application provides.
3. Technologies – The technologies used to make this application a success.
4. Demonstration – A quick demonstration of the application prototype.
5. Issues & Next Steps – Issues and obstacles faced during the application prototype, and what the next steps are to overcome the obstacles and complete the project design.

2 System

2.1 Requirements

2.1.1 [FREQ-01] User Registration

Description & Priority

The user registration feature in this application has quite a high priority because when you register an account you must also configure Multifactor Authentication with the Google Authenticator smart phone application. Multifactor Authentication is one of the functional requirements with the highest priority. As security is of great priority in this application, having multifactor authentication is a must.

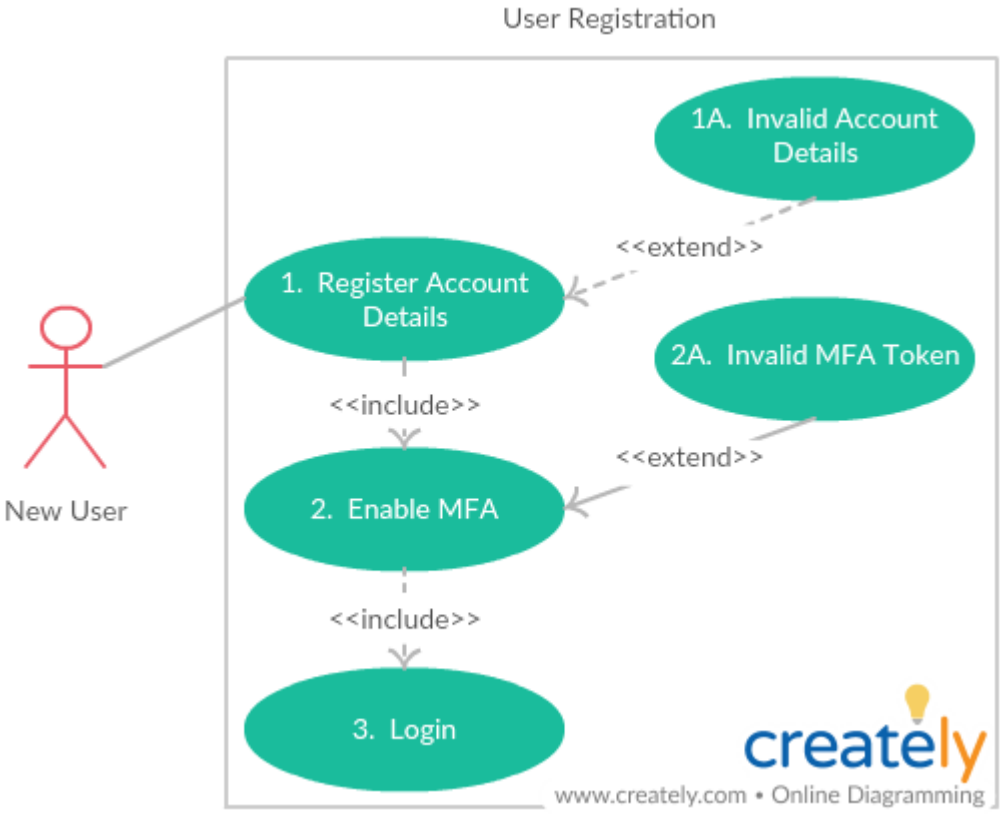
Scope

The scope of this use case is to create a feature for the web vulnerability scanner where when a user creates their account they must configure Multifactor Authentication on their account.

Description

This use case describes the action when a New Users visits the web application to create a new account, so they can avail of the services the application provides.

Use Case Diagram



Flow Description

Preconditions

There are no pre-conditions to creating an account.

Activation

This use case starts when a New User visits the website and goes to the registration page of the web application.

Main Flow

1. The New User submits their account details and the system verifies the account details
2. The User Registration prompts and accepts the OTP from the user to enable MFA.
3. The system stores the information and moves the user to the Enable MFA use case.
4. The User Registration system logs the new user in.

Alternate Flow 1A: Invalid Account Details

1. The account details submitted are incorrect. Use case restarts at 1 Register Account Details.

Alternate Flow 2A: Invalid MFA Token

1. Invalid MFA information submitted. Use case restarts at 2 Enable MFA.

Termination

The system has created an account, enabled MFA, and proceeded to log the user in.

Post Condition

The New User is now redirected to their dashboard where the main functional requirements of the application present.

2.1.2 [FREQ-02] Add Endpoint

Description & Priority

This use case describes where a user is adding an endpoint to their list of endpoints. This list of endpoints will be scan when a full scan is initiated. As these endpoints are the main data driving the web application, care must be taken to validate the input from the user. Also, as this is the main information to carry out the primary function when a scan is initiated on the endpoint this use case has a priority of critical.

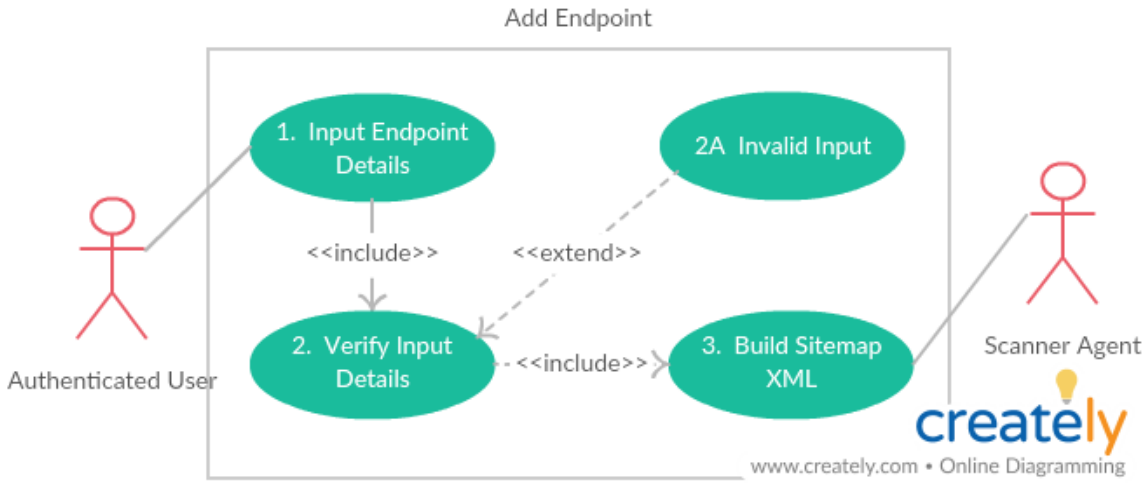
Scope

The scope of this use case is to supply a function where a user can add a new endpoint to their list of endpoints.

Description

This use case describes the action when a user wants to create a new endpoint, which in turn can be used to perform a scan against their website for vulnerabilities.

Use Case Diagram



Flow Description

Precondition

The user is successfully authenticated and logged in.

Activation

This use case is activated when the user visits their account settings page and inputs the domain name of the endpoint.

Main Flow

1. The Authenticated User inputs the details of their domain and/or IP address.
2. The Add Endpoints system verifies the users input
3. The Scanner Agent builds an XML sitemap of the user's endpoint.

Alternate Flow: 2A Invalid Input

1. The user submits invalid endpoint format. The use case returns to 1 Input Endpoint Details.

Termination

The endpoint is available for scanning for common vulnerabilities.

Post Condition

The system awaits the next command

2.1.3 [FREQ-03] Start Scan

Description & Priority

This use case describes when a user wants to initiate a scan on endpoints submitted using requirement 2. This is the use case with the highest priority as the main functional requirement of this application is to allow a user to start a scan on their website or webserver.

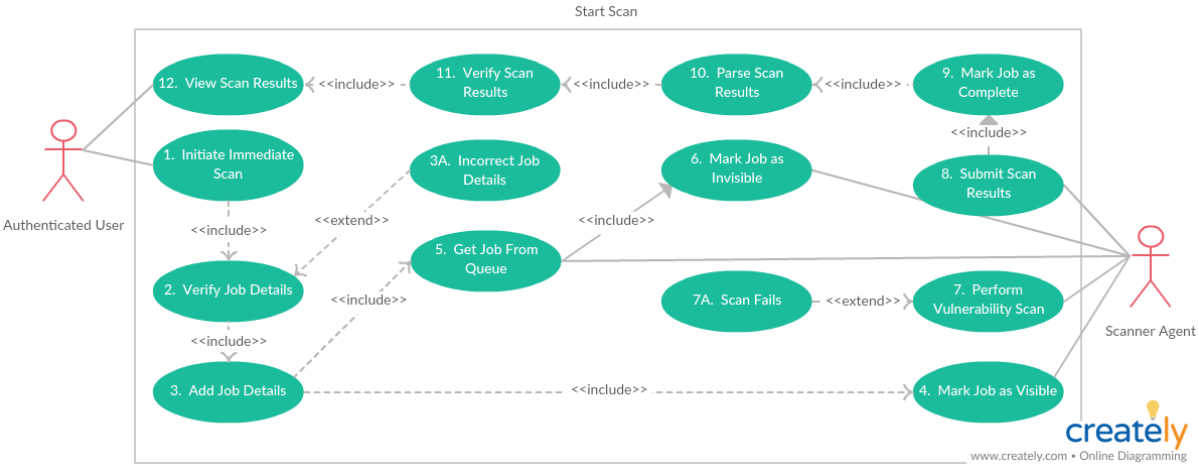
Scope

The scope of this use case is to create a feature that will allow a user to submit a request to start a scan on their domain name or IP address.

Description

This use case describes the action when a user wants to perform a scan against their website for vulnerabilities.

Use Case Diagram



Flow Description

Precondition

The Authenticated user has successfully logged in and has successfully submitted an endpoint to be scanned.

Activation

This use case is activated when the user visits the main dashboard and submits a request to start a scan on a specific endpoint.

Main Flow

1. The Authenticated User initiates a request to perform a scan on their website.
2. The details of the scan are verified by the Start Scan system.
3. The job details are added to a queue.
4. The Start Scan system marks the new job as visible, to be visible by the Scanner Agent Actor.
5. The Scanner Agent gets the job details from the job queue.
6. The Scanner Agent marks the job as invisible, so other agents are aware that someone is working on the job.
7. The Scanner Agent starts a vulnerability scan.
8. The Scanner Agent submits the details of the scan to the Start Scan system.

9. The Scanner Agent marks the job as complete.
10. The Start Scan system parses the scan results.
11. The Start Scan system verifies the results of the scan.
12. The Authenticated User can now view the results of the scan in a human readable format.

Alternate Flow: 3A Incorrect Job Details

1. The system recognizes that the details of the submitted job are incorrect. The use case restarts at 1 Initiate Scan.

Termination

The results of the scan are returned to the dashboard front end to be viewed by the user whenever they need to.

Post Condition

The system awaits the next command.

2.1.4 [FREQ-04] Edit User Settings

Description & Priority

This use case describes when a user wants to change any of the details pertaining to their account. For example, password, their name, etc.

Scope

The scope of this use case is to create a feature that will allow a user to change the details of their account.

Description

This use case describes the action when a user wants to change the details of their account.

Flow Description

Precondition

The Authenticated user has successfully logged in.

Activation

This use case is activated when the user visits the main dashboard and selects account from the drop-down list available after clicking their username.

Main Flow

1. The user adds a new value to the form input fields available.
2. The user clicks submit/update.
3. The backend verifies that request came from the correct authenticated user.
4. The database is updated with the new details.
5. A message is returned to the user advising on the status of the update (successful, failed, etc.).

Termination

The user is presented with a message outlining if the update was successful or not.

Post Condition

The system awaits the next command.

2.1.5 [FREQ-05] Delete Endpoint

Description & Priority

This use case describes when a user wants to delete an endpoint from their list of verified endpoints.

Scope

The scope of this use case is to create a feature that will allow a user to remove any of their verified endpoints.

Description

This use case describes the action when a user wants to delete an endpoint from their account

Flow Description

Precondition

The Authenticated user has successfully logged in.

Activation

This use case is activated when the user visits the main dashboard and selects account from the drop-down list available after clicking their username, or they click the view all button above the list of verified endpoints on the user dashboard.

Main Flow

1. The user clicks the delete button next to the listed endpoints.
2. The backend verifies that request came from the correct authenticated user.
3. The endpoint, along with the verification details are removed from the database.
4. A message is returned to the user advising on the status of the update (successful, failed, etc.).

Termination

The user is presented with a message outlining if the update was successful or not.

Post Condition

The system awaits the next command.

2.2 Non-Functional Requirements

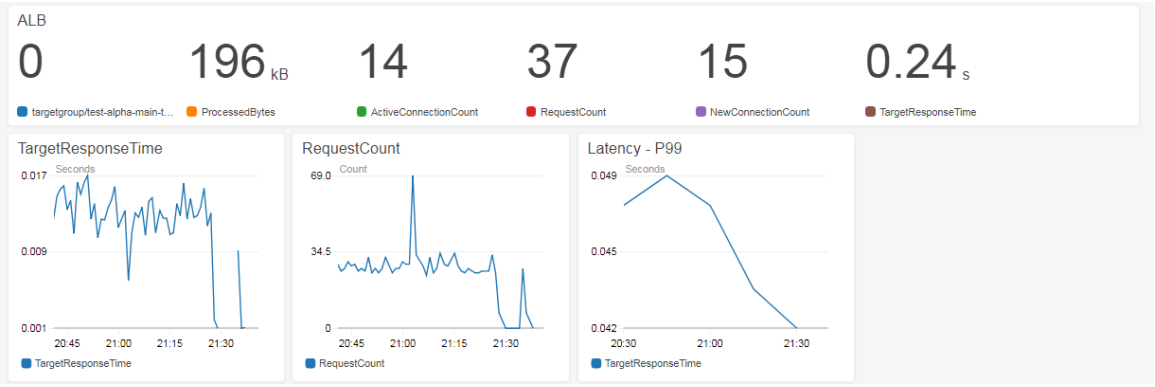
2.2.1 [NFR-01] Availability Requirement

Availability is a very important non-functional requirement. As users should be able to access their scan results at any time and have the system complete submitted jobs as quick as possible. To make sure that the application is available to complete user's job as quickly as possible, the scanner agent can easily be scaled to other servers and almost immediately start consuming user's submitted jobs.

Using AWS auto-scaling, based on the CPU utilization of the current servers processing jobs can be used to determine whether a more resources are required to keep up with the demand. Servers will be able to scale out and in depending on the demand of the application, ensuring the highest availability.

AWS CloudWatch metrics have been implemented to create a Dashboard that allows for immediate notifications in the event of web server failures. If a backend webserver fails, a notification is sent, and the instance is replaced using AWS auto-scaling.

Using CloudWatch dashboards we can constantly monitor the state of the web application through a very convenient dashboard viewer:



As you can see from the above dashboard widgets, we can constantly keep an eye on the latency between the AWS Application Load Balancer, and the registered backend web servers. We can keep an eye on the request count for possible DDoS attacks. The metrics that are updated through the CloudWatch dashboard are:

- TargetResponseTime – The latency between the application load balancer and the registered servers.
- RequestCount – The number of requests hitting our application.
- ActiveConnectionCount – The number of established and active TCP connections to our web servers.
- ProcessedBytes – The number of bytes that has been processed by the application load balancer.
- HealthyHostCount – The number of instances behind the load balancer that are considered healthy and running.
- UnhealthyHoustCount – The number of instances behind the load balancer that are still registered and have failed the load balancing health checks.

2.2.2 [NFR-02] Recovery Requirement

This section will outline the need for a recovery structure in the Web Vulnerability Scanner application. As customer data is of the upmost importance, the recovery requirement of the application is also a high priority. The database being used, and ORM frameworks being used have both been created with transactions with a massive concern. The application has been developed to roll back any changes that may fail while writing to the database. A single schema is used but is backend up and replicated to ensure that in the event of data loss, the data is easily recoverable. This will ensure that user data is safe in the event of a catastrophic event and is easily recoverable.

2.2.3 [NFR-03] Robustness Requirement

Robustness in the development of the web application and agent application are a high priority considering the stability of the applications to prevent users encountering errors or the scanner encountering system crashes. To prevent catastrophic failure several considerations must be taken into consideration for any scenario that may cause catastrophic failure to the system. It is essential that during the development cycles of the applications that all bugs and errors must be corrected. The elimination of such errors and bugs will ensure the successful and robust deployment and use of the Web Vulnerability Scanner application.

2.2.4 [NFR-04] Security Requirement

This section will outline the need for very strong security implementations in the Web Vulnerability Scanner application. There are many security concerns in a web application that interacts with a database, and a front end that a user uses to interact with the application. To ensure confidentiality of a user's account multifactor authentication has been implemented. To ensure integrity of user's data multiple implementations have been made to sanitize and verify user submitted data to ensure the data is the same data that the user submitted. TLS encryption has been implemented in all sections of the application to ensure users that their data is encrypted while in transit to the web application. To ensure availability, the application has been developed to be easily scalable while still ensuring security.

2.2.5 [NFR-05] Reliability Requirement

This section will outline the need for good reliability requirements to be implemented in the application. To determine the application demands and performance reliability testing will be performed before launching the application to production phase. Metrics like response time, CPU utilization, memory utilization, will be gathered when benchmarking the application before releasing it to a production environment. This will ensure that the application is always reliable and available to the users always.

2.2.6 [NFR-06] Maintainability Requirement

When it comes to maintainability, there are several concerns that are being addressed in this project specification document. The online front-end application can be updated on the fly and synchronized with the serve in near real time. Database maintainability will be ensured by routinely backing up the data of the application for easy of recovery and ensure maintainability.

2.2.7 [NFR-07] Portability Requirement

As this application is delivered as a web application and is supported in all major browsers the portability requirements of this application have low priority as all users will be able to access through both desktop and phone versions of browser to use the application.

2.2.8 [NFR-08] Extendibility Requirement

A key requirement for this application is the extendibility of the system. The potential of the application is promising, and with the user of extensive cloud computing and superior networking the application can easily grow in stability and functionality without compromising the performance of the existing software.

2.2.9 [NFR-09] Reusability Requirement

The ability to re-use code in the application will help to make sure that the application and its updates are delivered to its users in the most time efficient and stable manner. By re-using parts of the backend and front-end code the system can re-use code to help deploy the updates to users as soon as possible to easily and efficiently respond to feature requests, and bug fixes.

2.2.10 [NFR-10] Resource Utilization Requirement

This section outlines the need for a resource utilization requirement. By developing an application that is easily scalable, we ensure that the resource utilization requirements are dealt with a high priority. The application that performs the heavy load of the application has been designed in such a way that if more resources are required to keep up with the demand of the application, another instance of the application can be launched onto another server and start assisting current servers processing the load. This ensures that all resource required for the demand of the application are always available. When the demand of the application has decreased, the servers can be easily

scaled down to ensure that only enough resources that are required at any exact moment are being used.

2.2.11 [NFR-11] Usability Requirements

The usability requirements of this application are that it must be available on all operating systems, and smart phone devices. By developing a mobile friendly web application, we can deliver the application to users of all operating systems, on nearly all platforms.

2.2.12 [NFR-12] Error Exceptions & Handling

To not interrupt the flow of the web application if errors are encountered. A logical exception class structure has been created for catching errors, but not causing the application to crash.

```php
<?php
/** @class Exception ...*/

namespace Ecne\Exceptions;

use Throwable;

class Exception extends \Exception
{
    /**
     * Exception constructor.
     * @param string $message
     * @param int $code
     * @param Throwable|null $previous
     */
    public function __construct($message = "", $code = 0, Throwable $previous = null)
    {
        parent::__construct($message, $code, $previous);
    }

    /**
     * @return string
     */
    public function getErrorMessage()
    {
        return "Error on line: {$this->getLine()} in {$this->getFile()} <b>{$this->getMessage()}<b>";
    }
}
```

The main reason for throwing exceptions is when the user attempts to access a resource from the database that does not exist. A user can easily modify the URL for loading resources and cause the application to crash. So, an exception class was created for this and try and catch blocks are used when accessing the database directly.

```php
<?php
/** @class NoModelFoundException ...*/

namespace Ecne\Exceptions;

class NoModelFoundException extends Exception
{
    /**
     * NoModelFoundException constructor.
     * @param string $message
     * @param int $code
     * @param null $previous
     */
    public function __construct($message = "", $code = 0, $previous = null)
    {
        parent::__construct($message, $code, $previous);
    }

    /**
     * @return string
     */
    public function getErrorMessage()
    {
        $this->message="Empty result returned when querying database.  No modal found.";
        return parent::getErrorMessage();
    }
}
```

2.2.13 [NFR-13] SQL Injection Prevention

SQL injection is still a major concern for a lot of applications. Especially web applications which allow users to modify the URL with characters that may cause the DB driver to throw and error. To overcome this user input is not only sanitized but, the input is also added to prepare statements by using an ORM framework that uses PDO objects for accessing the database models.

2.2.14 [NFR-14] XSS Prevention

All user input through the form of GET & POST variables are sanitized before being stored in the database. All possible HTML characters are escaped so that users cannot input JavaScript script tags and run commands that are stored in the database, and then executed as DOM elements when displayed back to the user.

2.2.15 [NFR-15] CSRF Prevention

To stop Cross-Site Request Forgery every form has a secret token that is associated with the user's session details. This means if an attacker was to force a form submission for a user the backend application will reject this as the secret token passed with for was not associated wit the user's session. Once the page is refreshed, the token is re-generated, and the previous token will be unusable.

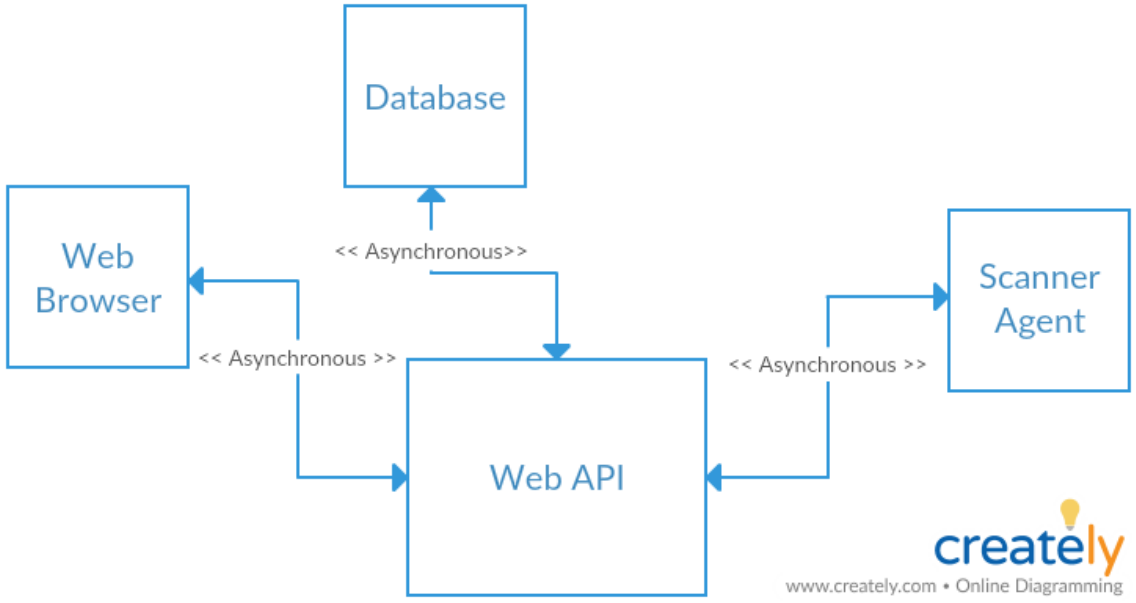
```
▼<form id="form-update-name" method="post" ajax-location="/user/update" style="width: 75%;">
    <input type="hidden" name="profile-name-update-token" value="8c2b1d2a84e31ee3833ff1d8698a3ca9">
  ▼<div class="form-group">
      <label for="account-name">Name</label>
      <input name="account-name" type="text" class="form-control" id="account-name" placeholder="John O'Grady" value="John O'Grady">
    </div>
    <input type="submit" class="btn btn-success">
  </form>
```

As you can see from this snippet. There is a hidden input field with the name profile-name-update-token, and its value is a randomly generated string. If we look at the code that is run when this form is submitted, you will see that we are making sure that the form submission actually came from the logged in user.

```
public function update()
{
    if (Session::exists( name: 'userid')) {
        try {
            # create model of logged in user
            $user=User::select()
                ->type('User')
                ->eq('Id', Session::get('userid'))
                ->one();
            if ($user !== null) {
                if (Input::get('account-name') && Input::secure( name: 'profile-name-update-token')) {
                    $user->Name=Input::clean( string: 'account-name');
                    $user->save();
                    echo json_encode(['status'=>'success','message'=>'Name updated successfully']);
                    exit();
                } elseif (Input::get('account-old-password') && Input::secure( name: 'password-change-token')) {
                    if ($user->Password === Hash::make(Input::get('account-old-password'), $user->Salt)) {
                        if (Input::get('account-new-password') === Input::get('account-confirm-password')) {
                            $user->Password=Hash::make(Input::get('account-new-password'), $user->Salt);
                            $user->save();
                            echo json_encode(['status'=>'success','message'=>'Password changed successfully']);
                            exit();
                        }
                    }
                }
            }
        } catch (NoModelFoundException $nmfe) {
            echo $nmfe->getErrorMessage();
        }
    }
}
```

2.3 Design & Architecture

2.3.1 System Architecture

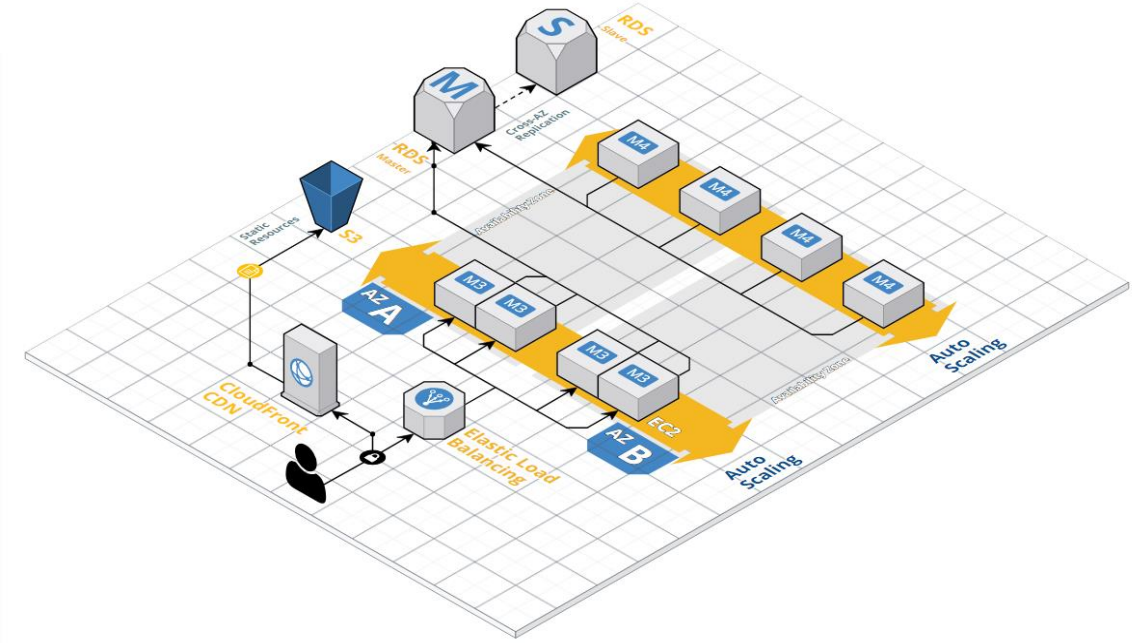


The Web Vulnerability Scanner application is separated into separate tiers for separation of tasks to ensure that the system is as highly available and reliable as possible.

The following modules are included in the system architecture of the Web Vulnerability Scanner application.

1. Web API – This is the module that asynchronously interacts with all other modules of the application. This is where users submit details that need to be stored in the database or sent to the scanner agent for processing. The Web API can both read and write to the database at the same time, while also displaying the information from the database to the web browsers to be viewed by users in the front-end GUI. The scanner agent also interacts with the Web API to submit and post details of the jobs submitted by users through the web browser. The scanner agent also submits details to the Web API to be processed and saved in the database.
2. Web Browser – This is how the user will interact with the system. Using the front-end GUI available via a web browser, the user can create and account, add endpoints to their dashboard, and submit and/or schedule scans on the endpoints in their dashboard. The details are forwarded to the Web API which will add them to the database. If the details submitted are regarding the details of a new job, the Scanner Agent will poll the Web API for those details and submit the job accordingly.
3. Scanner Agent – This is the scanner agent application developed using python3 and is set to run on Linux backend servers. This application is scalable, as this application uses a lot more resources that the Web API or Web Browser. This application is responsible for pulling newly submitted jobs and consuming them. Once the job is completed, the details are submitted to the Web API to be stored in the database and displayed to the user using the Web Browser.
4. Database – This is the module that is solely responsible for reliably storing all information about users, jobs, endpoints, and any activity that happens within all other modules of the application.

2.3.2 Amazon Web Services



CloudFront CDN

Amazons CloudFront Content Delivery Network will be used to load and cache static media for the web application, such as images, CSS files, and JavaScript files.

S3

Amazons Simple Storage Solution will be used to host images, and other assets for the web application.

Elastic Load Balancing

Elastic Load Balancing will be used for the web application. By using an Application Load Balancer, and multiple web servers, high availability is achieved to users. If a web server fails, it can simply be removed from, and no users will be routed to that server.

An Application Load Balancer allows us to perform SSL termination separate to the web servers, meaning that the computational resources needed for SSL termination is separated from the web servers.

Application Load Balancers also allows for writing access logs to an S3 bucket and can be used if we need to troubleshoot any issues with the web application.

Enabling session stickiness will allow for users to be returned to the same backend instance they originally connected to so that session information is not lost between requests. A shared medium like memcache could easily be introduced, but it out of scope for this application, and session stickiness will suffice.

Auto-Scaling

By using auto-scaling we can scale the web server backend instances in the event of increased load. Auto-scaling can also be used to scale out more workers for performing scans against client domains.

RDS

Amazon RDS is used as the main database for the web and python application.

Availability

2.4 Implementation

This application is designed with a queueing system. What this means is that when a user submits a job (i.e. scan a website) the job is added to a queue. The application is constantly checking this queue for a job. Once a job is inserted into the queue, the application will pull down the details of the job and start to execute the job based on the details. This means that a queue class has been created. Like below:

```
class Queue:

    items = []

    def __init__(self):
        self.items = []

    def is_empty(self):
        return self.items == []

    def enqueue(self, item):
        self.items.insert(0, item)

    def dequeue(self):
        return self.items.pop()

    def get_size(self):
        return len(self.items)
```

The queue class is quite simple, and contains an array of items, and functions to check if the queue is empty, to add a new item (enqueue), to obtain the oldest item in the list (dequeue) and return the size of the queue.

A polling mechanism has been added. This polling mechanism is responsible for connecting to the Web API using an access key that is only known to the python agents and checking for any jobs that are currently visible. The poll class uses a function called poll_queue to check for any new jobs that the application needs to run:

```
def poll_queue(self):
    """ poll_queue

    :param cursor:
    :param queue:
    :return:
    """

    # make api call to get jobs from queue
    response = requests.post('https://scanner.ogradyjohn.com/api/GetWorkItems', data = {'access-key': 'gxC3bGJhD9R0bGcaLhpt'}, verify = True)

    workitems_json = json.loads(response.text)
    for item in workitems_json:
        # add workitem to queue
        self.queue.enqueue(item)

        # udpate workitem status to invisible to hide from other agents
        workitem_obj = { "workitem_id": item['Id'], "status": "invisible", "access-key": "gxC3bGJhD9R0bGcaLhpt"}
        update = requests.post('https://scanner.ogradyjohn.com/api/UpdateWorkItemStatus', data = workitem_obj, verify = True)
```

As you can see from the above screenshot, once a job is loaded by the poll class, the jobs status is set to invisible. These means that other instances of the application running on separate servers will not also obtain the same job, but if the server that has downloaded the job details runs into any errors, or issues, the job can be marked as visible again, and another instance can download the job where the current instance fails. This is to try to eliminate the possibility of jobs not running due to errors or issues with the servers.

The main method in the main.py class is where the scans take place. This class uses multiple instances of a command class, in the command class we specify the command we want to run and all the parameters. This class will run the intended command and store the output to be used for later.

```
class Command:

    encoding="utf-8"

    subprocess=None
    command=None
    parameters=None

    output=None

    def __init__(self, command, parameters):
        self.command=command
        self.parameters=parameters

    def run(self):
        args=[]
        args.insert(0, self.command)

        for parameter in self.parameters:
            args.insert(len(args), parameter)

        self.subprocess=subprocess.Popen(args, stdout=subprocess.PIPE)
        self.output=(self.subprocess.communicate()[0].decode(self.encoding))

        return self.output
```

The command class returns the results to the main class to be written to a database to be parsed and displayed on the front end. The command or main class have no direct access to the database, and will push all changes to the Web API which in turn is responsible for writing and parsing the data.

The main class is responsible for running the job and updating the job entity with the details of the job that has been run.

```python
def update_queue():
    poller = Poll(queue)
    time.sleep(5)
    poller.poll_queue()
    while True:
        if (queue.get_size() > 0):
            workitem = queue.dequeue()
            workitem_details = json.loads(workitem['Details'])
            # used for parsing the parameters of the job template
            parsed_params=[]
            for param in workitem_details['parameters']:
                if (re.search(r'\$host', param)):
                    param = re.sub(r'\$host', workitem_details['endpoint'], param)
                # append the parsed parameters
                parsed_params.append(param)

            # run the command
            logger.info('running: ' + workitem_details['command'])
            command = Command(workitem_details['command'], parsed_params)

            # don't submit nikto command output - write to file and upload to S3
            if (workitem_details['command'] == '/usr/local/bin/nikto.pl'):
                command.run()
                # when command finished put object in S3 bucket and remove from server
                file_content = open('/nikto_results/' + workitem_details['endpoint'] + '.htm', 'rb')
                file_key = workitem_details['job_id'] + '.htm'
                response = s3.Bucket('scanner-nikto-output').put_object(Key=file_key, Body=file_content)
                url = client.generate_presigned_url(ClientMethod='get_object', Params = {'Bucket': 'scanner-nikto-output', 'Key': file_key})
                os.remove('/nikto_results/' + workitem_details['endpoint'] + '.htm')
                logger.info(response)
                logger.error(url)
                command_output = url
```

```python
                logger.error(url)
                command_output = url
            elif (workitem_details['command'] == 'python3'):
                command.run()
                file_content = open('/sitemaps/' + workitem_details['endpoint'] + '.xml', 'rb')
                file_key = workitem_details['endpoint'] + '.xml'
                response = s3.Bucket('scanner-sitemaps').put_object(Key=file_key, Body=file_content)
                os.remove('/sitemaps/' + workitem_details['endpoint'] + '.xml')
                logger.info(response)
                # get public url of sitemap.xml file
                bucket_location = client.get_bucket_location(Bucket='scanner-sitemaps')
                url = "https://s3-{0}.amazonaws.com/{1}/{2}".format(bucket_location['LocationConstraint'], 'scanner-sitemaps', file_key)
                logger.error(url)
                command_output = url
            else:
                command_output = command.run()
            logger.success('completed: ' + workitem_details['command'])

            # @todo udpate workitem details
            workitem_complete = {
                "workitem_id": workitem['Id'],
                "workitem_output": command_output,
                "access-key": "gxC3bGJhD9R0bGcaLhpt"
            }

            update = requests.post('https://scanner.ogradyjohn.com/api/AddWorkItemDetails', data = workitem_complete, verify = True)
            logger.success('updated: ' + workitem_details['command'])

        # sleep for <x> seconds between workitems submissions
        time.sleep(2)
        poller.poll_queue()
```

This class is always running, by using an infinite loop, and a delay between executions, the instance is always running, and when a new job is obtained (if queue.get_size() > 0) then the application proceeds to run all the necessary commands to complete the job.

First the job is dequeued from the queue, and the job details are obtained and parsed into a json object. For development purposes the details of when the job is started are outputted to the screen. Then the application builds an XML site of the user's website.

I have designed a template for running scans that the python agents are able to parse as it is stored in json format. The template splits the job into separate work items so that individual work items can be run in parallel, rather than running them in sequence (as was the case for the mid-point presentation). This allows for jobs to run quicker, and more agents to spawn to run more task simultaneously. This ensures that the application runs at its highest potential and finishes jobs as quick as possible.

See a sample job template below:

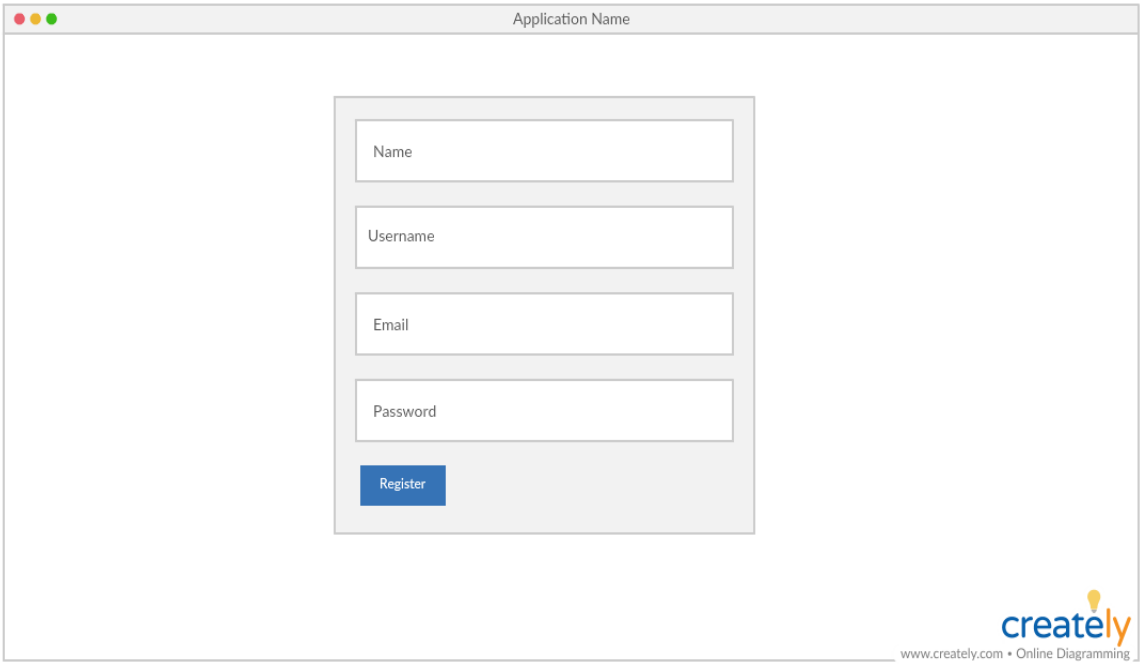
```
{
  "nmap":[
    "$host",
    "-p 1-1024"
    ],
  "/usr/local/bin/nikto.pl":[
    "-h",
    "http://$host",
    "-D",
    "DEPV",
    "-F",
    "htm",
    "-o",
    "/nikto_results/$host.htm"
  ],
  "curl":[
    "-sIL",
    "http://$host"
  ],
  "python3":[
    "/usr/local/bin/pythonsitemap/main.py",
    "--domain",
    "http://$host",
    "--output",
    "/sitemaps/$host.xml"
  ],
  "/usr/bin/nmap":[
    "--script",
    "ssl-enum-ciphers",
    "-p",
    "443",
    "$host"
  ]
}
```

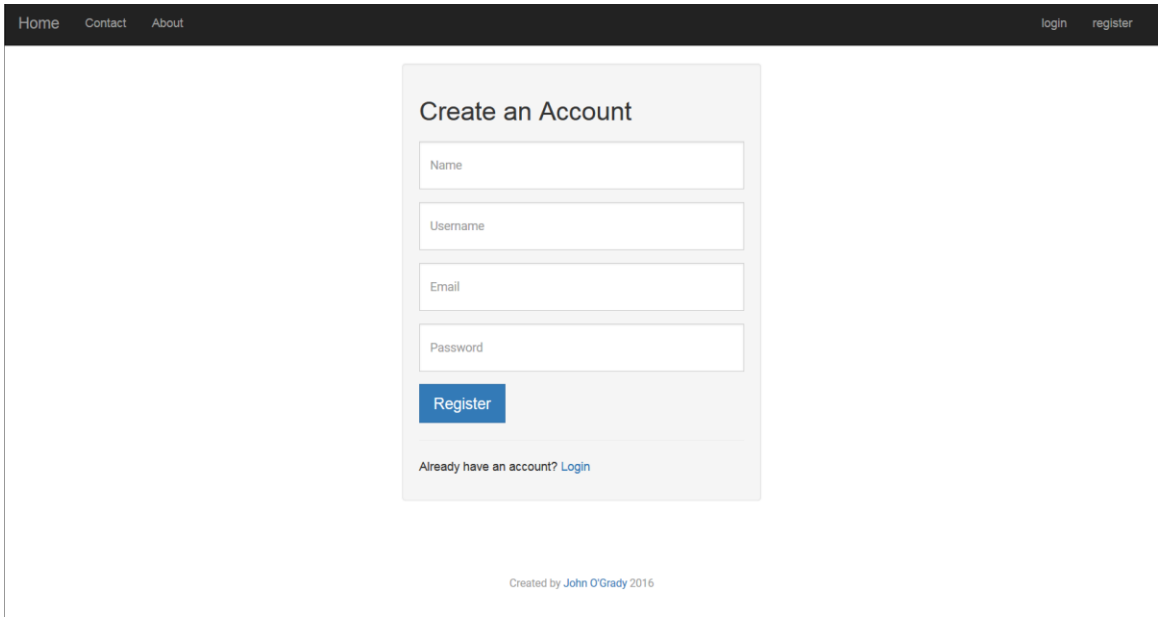
As you can see from the code as well that I am using a boto3 client, which is a python library for interacting with AWS services. Some command outputs are too large to submit to the API for storage, and exceed the maximum entity size for nginx servers, so I have written these details to a file, uploaded to S3, and then removed the file from the application servers.

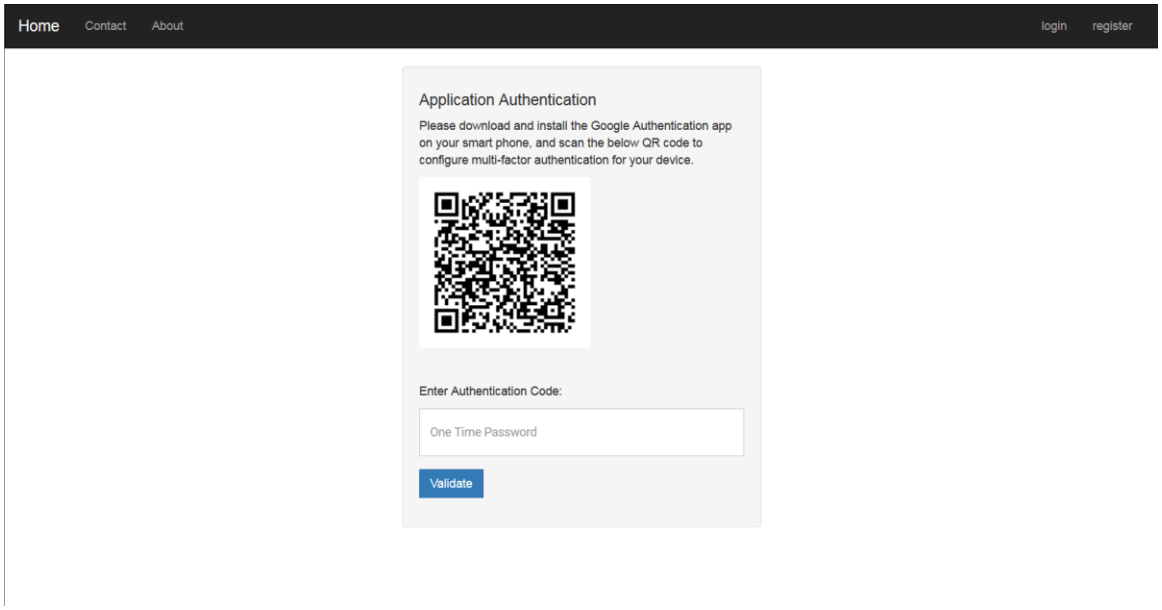
I have left the code of the front end and API, as the both just use standard MVC & ORM frameworks, and the main application is the python application performing most of the scanning features.

2.5 Graphical User Interface Layout

Account Registration

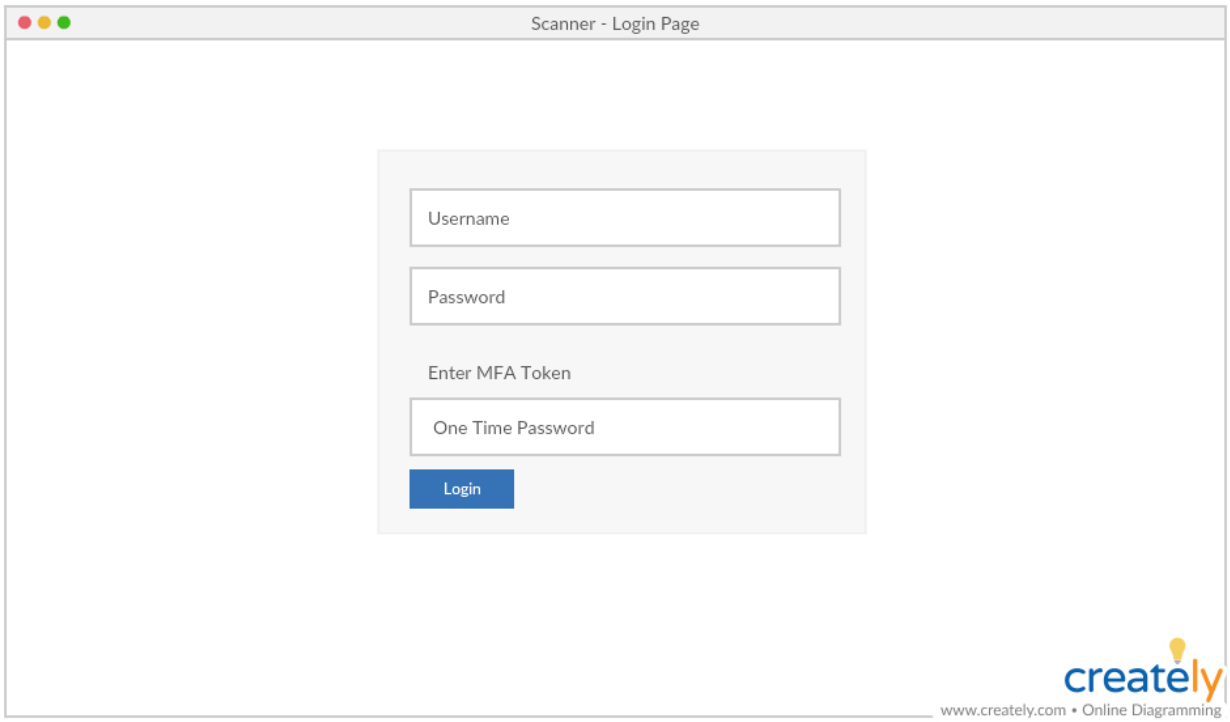






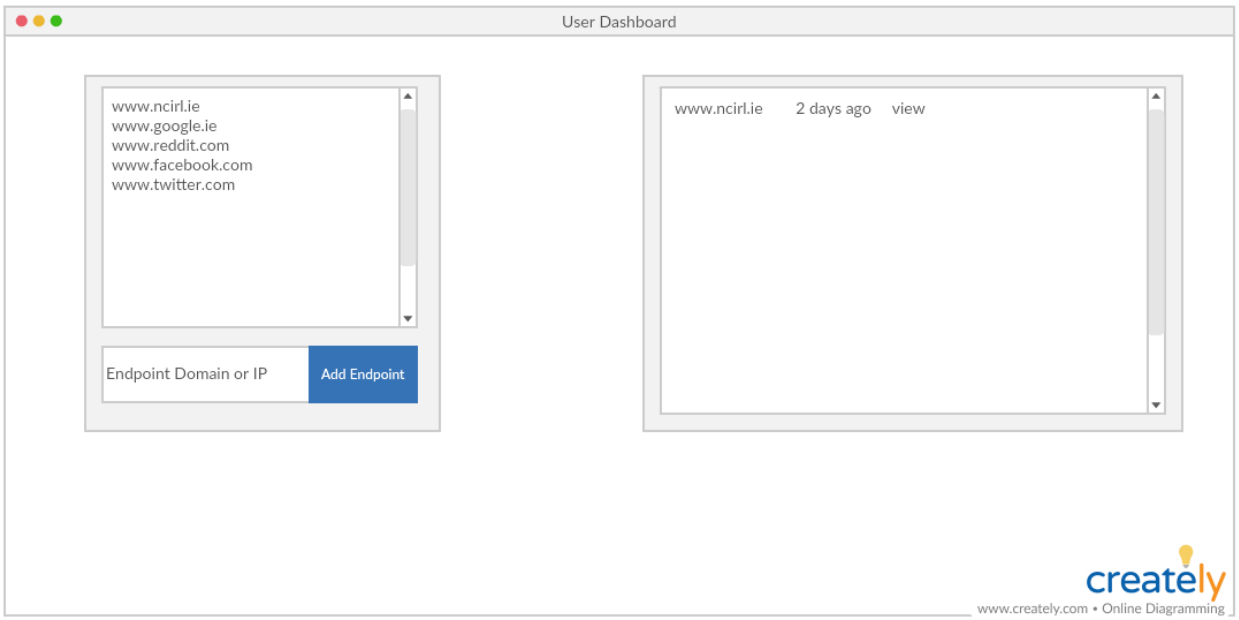
As you can see from the above screenshots, Multifactor Authentication will be necessary when signing up for an account. On the first page, the user will be prompted to add their name, username, email address, and password. Once they have registered these details, they will be brought to a screen where they need to scan a QR code using the google authenticator MFA application. Only then will their account be fully configured.

User Login

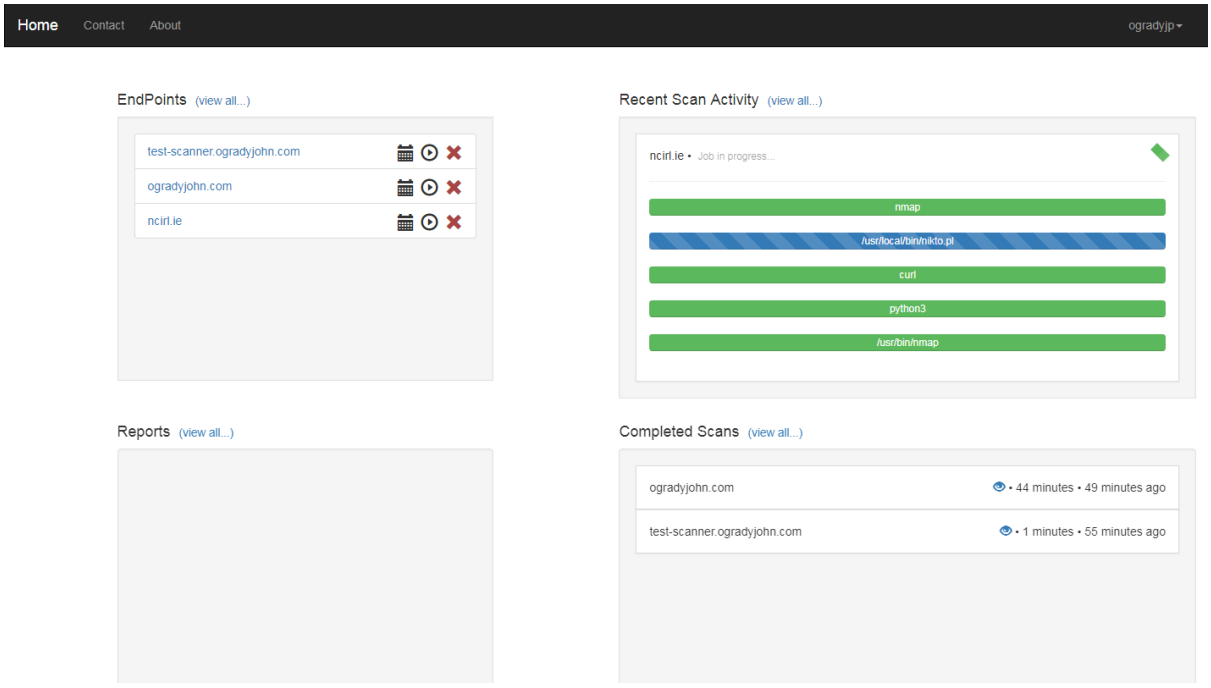


When a user logs in they must provide their username and password, but in order to login the also need to provide their multifactor authentication one-time password using the google authenticator smart phone application.

User Dashboard

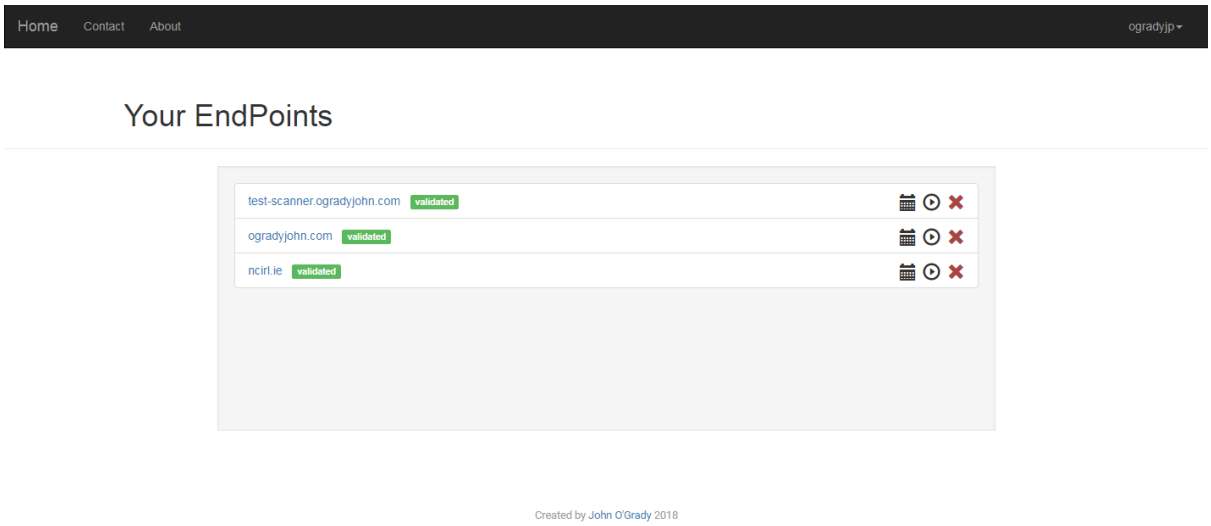


The dashboard is a central page for taking an overview of recent activity on your account. Here you can view the most recently added endpoints, any scans that are currently running and their status, a list of recently completed scans, and a list of any reports.

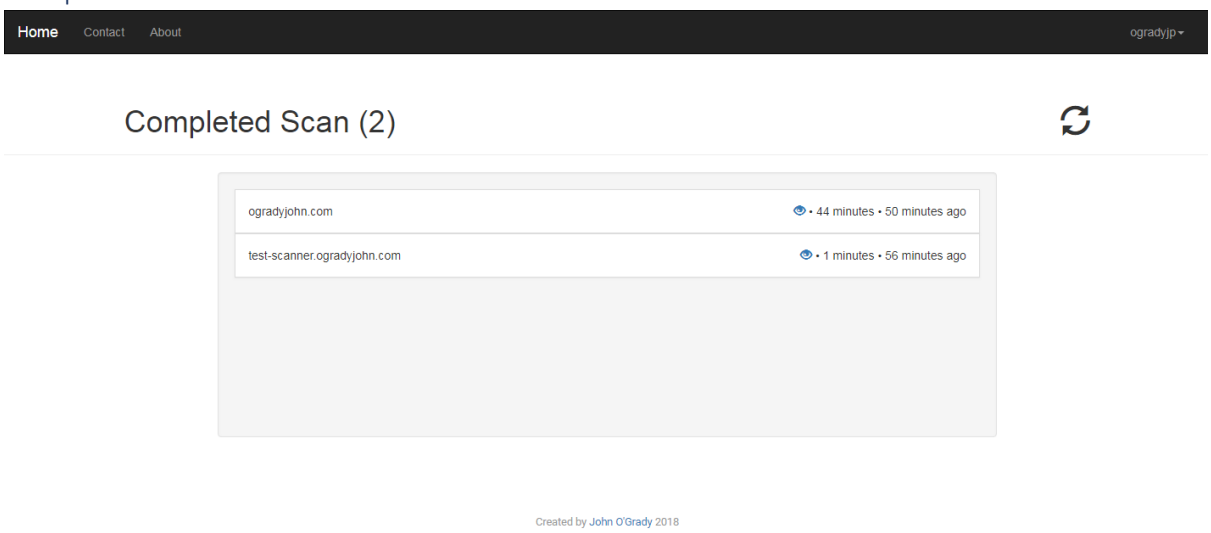


As you can see, the dashboard only shows a subset of the information available. But there are links on the dashboard that shows all information stored about the user's scans.

All Endpoints

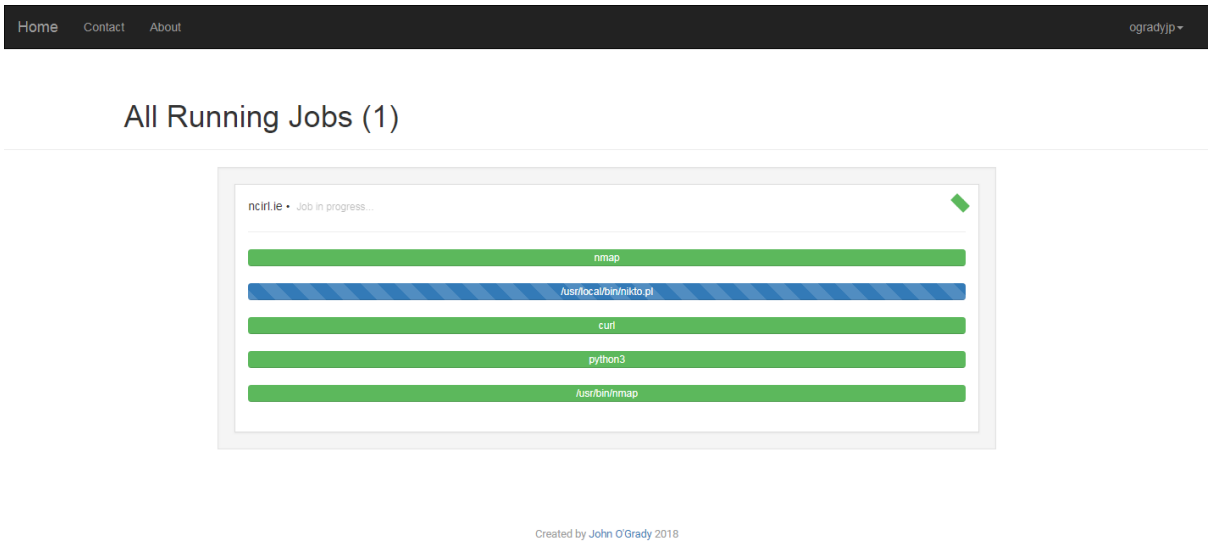


Completed Scans



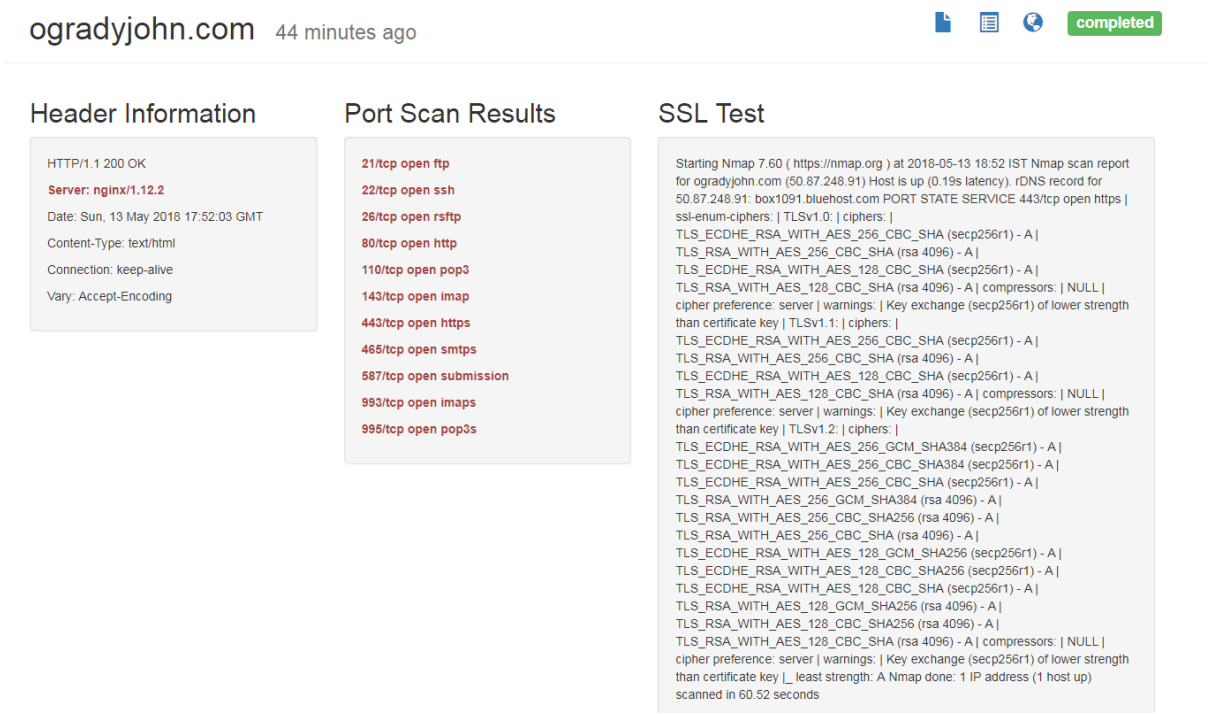
This shows a list of all completed scans for your endpoints.

All Running Jobs



The user dashboard is the main page the user will be on. Here they can add new endpoints and initiate new scans/scheduled scans. They can also see a list of the previous scans they have run, which contains basic information on the scan details. They can also open other pages that will give the full report on the scan (which is also emailed to the user when the scan is complete).

Scan Details Page



The scan details page gives the full output of the scan carried out. As you can see it provides information on the headers that your website is exposing publicly. It enumerates all open ports, and if you have SSL on your website, it enumerates all cipher suites used, and advises if any should be removed from your list of supported ciphers.

2.6 Testing

2.6.1 Unit Testing

This project will be tested using continuous integration. What this does is every time a change is committed to the GitHub repository Travis CI will run a predefined set of tests that I have created. These tests will test all sections of the application, and if any change causes and issue with other sections of the application, the CI build will fail, and I can see the exact issue that caused the tests to fail. This is great because as I add new features and changes to the application, I don't have to test all features, because Travis CI will be responsible for testing other sections of the application.

2.6.2 Security Testing

To ensure the web application is secure as possible a manual Penetration Test was carried out against the website. By using the OWASP Top Ten,

Injection

For the web application interacting with a MySQL database is necessary for storing persistent web data. MySQL databases are susceptible to MySQL injection. Using an ORM Framework, that uses prepared statements will help to overcome SQL Injections issues, but a test of this will need to be carried out to ensure any SQL Injection flaws are corrected.

Cross-Site Scripting (XSS)

By escaping all html characters for user input, XSS was impossible. Tested, and the scripts were outputted as raw data, and were not added as DOM elements

Cross-Site Request Forgery (CSRF)

By implementing tokens for every user and every form on the website it was impossible to perform cross-site request forgery.

3 Further Developments & Research

This is the completed product, although there are some features I would have liked to add, but due to time constraints I didn't seem viable, and some features were removed due to the time constraints. One tool I really wanted to develop was an SSL grading tool. To allow users to get a grade between A, B, and C depending on the TLS versions, and cipher suites they were accepting.

4 Conclusions

The advantages of this application are that customers are not required to manually run vulnerability scans on their website. By using this application, they can schedule scans at specific intervals. At different intervals, they can choose how intensive the scan will be. For example, they can have weekly scans, that don't go into much detail. Then another scan running every month that does go into some detail, and a yearly scan that goes into as much detail as possible that the application can.

5 References

6 Appendix

6.1 Project Proposal

Project Proposal Final.docx

6.2 Project Plan

software-project.mpp

6.3 Monthly Journals

ReflectiveJournalSept.docx

ReflectiveJournalOct.docx

ReflectiveJournalNov.docx

6.4 Project Poster

poster.PNG

6.5 User Manual

Web_Vulnerability_Scanner_User_Manu

6.6 GitHub Details

6.7 Other Materials Used