SANTA CLARA UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING
SOFTWARE ENGINEERING

Santa Clara University Information Technologies Bug Reporting System

by

Daniel Okazaki
Dana Suri
Sam Rietz

Santa Clara, California
December 14, 2018

Contents

List of Figures

Santa Clara University Information Technologies Bug Reporting System

Daniel Okazaki
Dana Suri
Sam Rietz

Department of Computer Engineering
Santa Clara University
December 14, 2018

ABSTRACT

Currently the IT department at Santa Clara University (SCU) has no way of streamlining bug report submissions on SCU online products and services. The IT department at SCU would like a clear way for clients to send bug reports to be processed internally and eventually resolved. Our solution will allow bugs to be discovered and fixed in a timely fashion reducing the impact of bugs that are detrimental to the user experience. The end result will provide clients with a frustration-free user experience and help the IT department maintain the existing SCU infrastructure.

Chapter 1

Introduction

1.1 Motivation

Currently the IT department at Santa Clara University (SCU) has no way of streamlining bug report submissions on SCU online products and services. The IT department at SCU would like a clear way for clients to send bug reports to be processed internally and eventually resolved. Our system will create a layer of transparency that allows clients and IT staff to find and resolve all bugs that negatively affect the user experience. This new system will be used alongside any of SCUâĂŹs online products and services. Some potential clients for this service are prospective and current students; as well as faculty and staff. The stakeholders invested in our product are the IT department managers, testers, and developers, as well as SCU, which is represented by its software.

If a prospective student or parent is introduced to a bug when they are looking at the SCU website, it will dissuade them from considering SCU as an option for their higher education. The only solution for reporting bugs is sending an email to the IT department highlighting the bug. This method is inefficient, giving the client no guarantee that the IT department will even read the email in the first place. The IT department has no way of knowing how to replicate the bug without many emails back and forth from the client and various IT representatives who will dodge responsibility with the possibility of days passing in between each interaction. In the end, the client has no idea if or when the reported bug will ever be resolved, resulting in a frustrating experience for all parties.

1.2 Solution

Our program will allow bugs to be discovered and fixed in a timely fashion which will reduce the impact of bugs that are detrimental to the user experience in the SCU system. The end result will provide clients with a frustration-free user experience and help the IT department maintain the existing SCU infrastructure.

With our system, clients will fill out a form whenever they encounter a possible bug in software. This form will have information on the nature of the bug, as well as information that can help IT services replicate the bug. The client will receive a unique code that they can use to access the bugâĂŹs status at a later date. The form will then be sent to a tester in the IT department who will determine if the bug needs to be fixed. If the bug is not a significant issue it will be marked as a non-issue. On the other hand, if the bug can be replicated and is a significant issue, the tester will send the form as well as details on how to replicate the bug to a manager. The manager then will choose which developers will work on a solution. Once the bug is fixed, the report and solution will be sent back to the testers to confirm that the issue has been resolved. If the testers confirm that the solution works as intended, the status will be marked as resolved. In the case that the bug has not been fixed, the report will be sent back to the developer to find a new solution. The managers will oversee the progress of all current bugs while the client can see the status of the bug they reported using its unique code.

In order to implement this system, we will create a web based application using an online database to store all of the relevant information to create and maintain the application. Our system will emphasize performance over security, and can be easily changed by future developers when needed.

Chapter 2

Requirements

2.1 Critical

- Client can send form on bugs found on SCU IT software (Functional)
- Client can see current status of the bug they found (Functional)
- Status of bug is updated routinely (Functional)
- Manager can assign bugs to tester/developer (Functional)
- Tester/developer can read bugs and progress bugs to next stage (Functional)
- Generate reports on current bugs and bug history for manager (Functional)
- Bug reports are passed from testers to managers to developers and back systematically (Functional)

2.2 Recommended

- Clean web-based UI for the client (Non-Functional)
- Clean web-based UI for the database (Non-Functional)
- User account system (Functional)
- High performance (Non-Functional)
- Secure (Non-Functional)
- Cross platform support (Non-Functional)

2.3 Constraints

- Needs to run on school Linux computers
- Needs to run in an internet browser
- Initial demo must be complete by October 31, 2018
- Full system must be complete by November 28, 2018

Chapter 3

Use Cases

The use case for our clients is to be able to send forms containing bug information to a database. The database sends back a unique code for the client to view the status of the bug at a later date. Testers can verify bugs, as well as add additional comments to bug reports to be sent to managers. Managers can assign reported bugs to a tester and verified bugs to a developer. They can also create new accounts for testers and developers. Developers can add additional comments about the bug, and send the report back to testers to verify the solution.

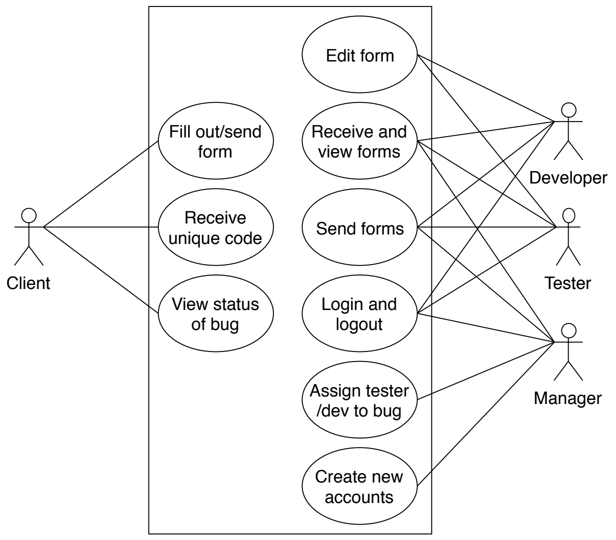

Figure 3.1: Use Case Diagram

3.1 Client

- Fill out and send form: allows the client to fill out a form with the necessary information to inform the SCU IT department of bugs in their software. Reporting bugs is the goal of this project. T
    - ⋆ Pre-conditions: Link to send bug must be clicked.

- ⋆ Post-conditions: Generates unique code for bug
  - ⋆ Exceptions: Client sends in blank form

- Receive unique code for bug: gives client the code necessary for them to view the status for the bug they reported. This helps ensure that the client only sees the bugs that they reported while still allowing for transparency.
  - ⋆ Pre-conditions: Form is sent
  - ⋆ Post-conditions: Unique code sent to browser from database
  - ⋆ Exceptions: Code is not unique, or does not get sent

- View status of bug: the client, if they have a code for a bug, can see its status. Again, this makes the bug reporting system more transparent for the clients.
  - ⋆ Pre-conditions: Form is sent
  - ⋆ Post-conditions: Status of bug is seen
  - ⋆ Exceptions: Code entered is incorrect

3.2 Tester

- Login: the tester logs into their account
  - ⋆ Pre-conditions: Username and password are filled out
  - ⋆ Post-conditions: None
  - ⋆ Exceptions: Incorrect credentials

- View report: the tester can view information about any bug they were assigned that is in the testing stage.
  - ⋆ Pre-conditions: Form is sent by manager or developer, and is in testing stage
  - ⋆ Post-conditions: Form is sent to tester to view
  - ⋆ Exceptions: Bug is completed, and therefore not visible to tester

- Send report: once the tester determines that a bug is an issue, they must send the report to managers, who will determine which developer should fix the bug, or to the developer it was assigned to.
  - ⋆ Pre-conditions: Form is sent and saved into database

    - ⋆ Post-conditions: Form is updated and sent to manager or developer
    - ⋆ Exceptions: None

- Edit report: when the bug is sent back to the tester, they can edit the initial report, rather than creating an entirely new one.
    - ⋆ Pre-conditions: Form is sent and saved into database
    - ⋆ Post-conditions: Form is sent to tester to edit
    - ⋆ Exceptions: None
- Logout: the tester logs out of their account
    - ⋆ Pre-conditions: Tester was logged in
    - ⋆ Post-conditions: None
    - ⋆ Exceptions: None

3.3 Developer

- Login: the developer logs into their account
    - ⋆ Pre-conditions: Username and password are filled out
    - ⋆ Post-conditions: None
    - ⋆ Exceptions: Incorrect credentials
- View report: the developer can view information about any bug they were assigned when it is not in testing.
    - ⋆ Pre-conditions: Form is sent by manager or tester, and is not in testing stage
    - ⋆ Post-conditions: Form is sent to developer to view
    - ⋆ Exceptions: None
- Edit report: the report can be edited by the developer to add additional details.
    - ⋆ Pre-conditions: Form is sent and saved into database
    - ⋆ Post-conditions: Form is sent to developer to edit
    - ⋆ Exceptions: None
- Send report: the developer can send the report back to the tester, along with a proposed solution.

  - ⋆ Pre-conditions: Form is sent by manager
  - ⋆ Post-conditions: Form is sent to tester to test proposed solution
  - ⋆ Exceptions: None

- Logout: the developer logs out of their account
  - ⋆ Pre-conditions: Developer was logged in
  - ⋆ Post-conditions: None
  - ⋆ Exceptions: None

3.4 Manager

- Login: the manager logs into their account
  - ⋆ Pre-conditions: Username and password are filled out
  - ⋆ Post-conditions: None
  - ⋆ Exceptions: Incorrect credentials

- Assign testers: Once the bug has been reported by a client, the manager gets to decide which tester is assigned to the given bug. Once chosen, the report will be sent to the testeer that was assigned to the bug.
  - ⋆ Pre-conditions: Form is sent by client
  - ⋆ Post-conditions: Form is sent to tester to
  - ⋆ Exceptions: None

- Assign developer: Once the report is marked as an issue by a tester, the manager gets to decide which developer is assigned to the given bug. Once chosen, the report will be sent to the developer that was assigned to the bug.
  - ⋆ Pre-conditions: Form is confirmed by tester
  - ⋆ Post-conditions: Form is sent to developer to work on
  - ⋆ Exceptions: None

- View report: the manager can see the reports on each bug, which detail how the bug was created and who has been working on it. This helps the manager decide which developers should work on future bugs.

- ⋆ Pre-conditions: Form is sent and saved into database
  - ⋆ Post-conditions: Forms sent to manager to view
  - ⋆ Exceptions: None
- Create account: the manager can create a new account, choosing the username, password, and permission level of the new user (such as tester or developer).
  - ⋆ Pre-conditions: All fields are filled out
  - ⋆ Post-conditions: New user is created
  - ⋆ Exceptions: None
- Logout: the manager logs out of their account
  - ⋆ Pre-conditions: Manager was logged in
  - ⋆ Post-conditions: None
  - ⋆ Exceptions: None

Chapter 4

Activity Diagram

This activity diagram is meant to provide an easy to understand decision graph for our project. It includes a start and end node with arrows to indicate path, boxes for actions, diamonds for decisions, and solid bars to indicate forks and joins. This diagram includes all clients, testers, developers, and managers.

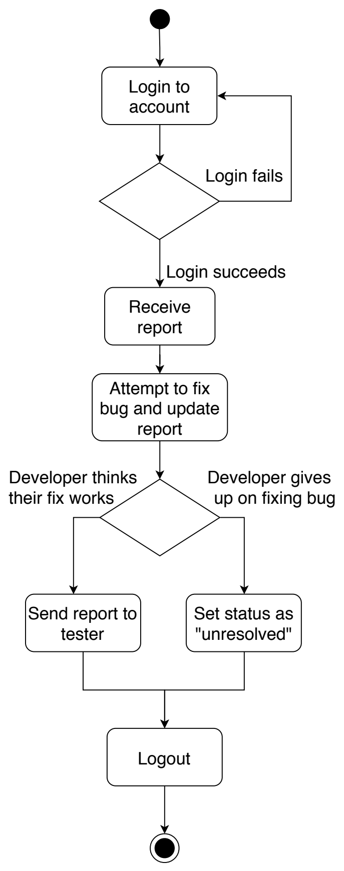
Login to account
Login fails
Login succeeds
Receive report
Attempt to fix bug and update report
Developer thinks their fix works
Developer gives up on fixing bug
Send report to tester
Set status as "unresolved"
Logout

Figure 4.1: Developer Activity Diagram

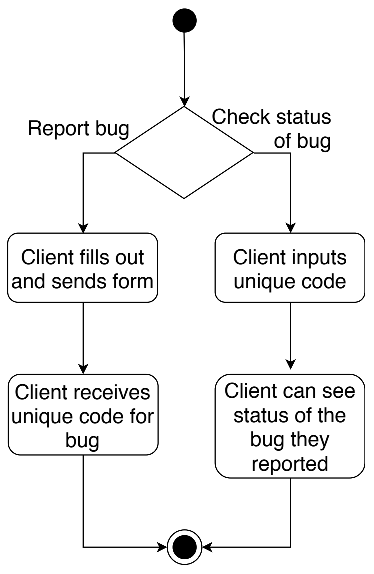
Report bug
Check status of bug
Client fills out and sends form
Client inputs unique code
Client receives unique code for bug
Client can see status of the bug they reported

Figure 4.2: Client Activity Diagram

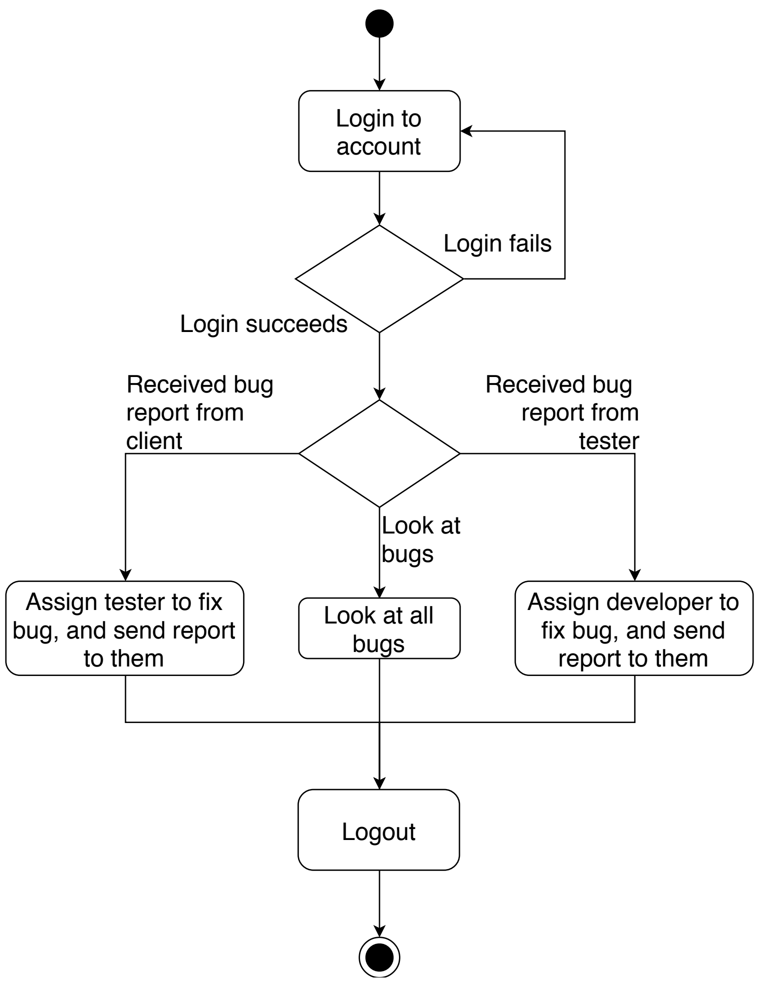
Login to account
Login fails
Login succeeds
Received bug report from client
Received bug report from tester
Look at bugs
Assign tester to fix bug, and send report to them
Look at all bugs
Assign developer to fix bug, and send report to them
Logout

Figure 4.3: Manager Activity Diagram

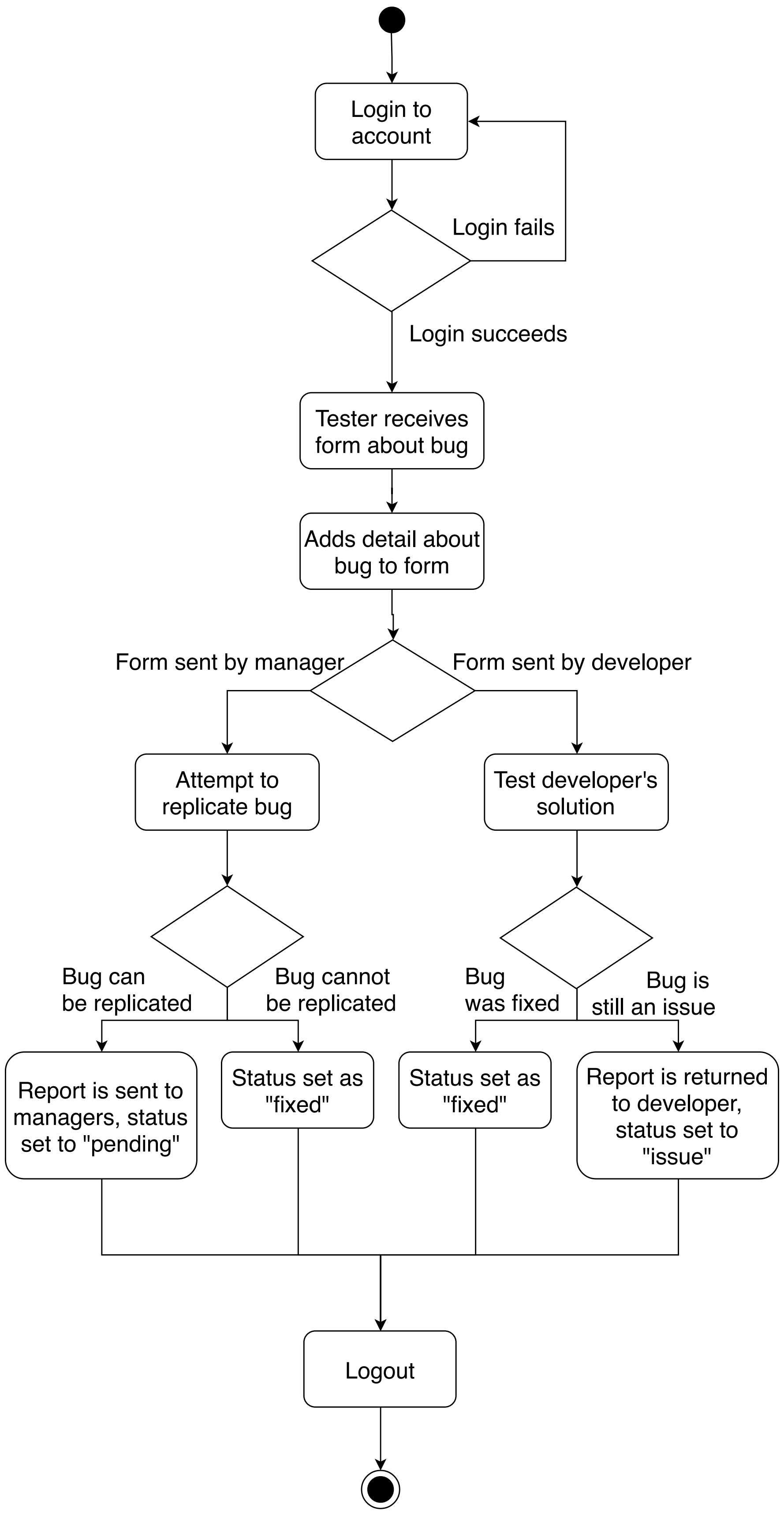

Figure 4.4: Tester Activity Diagram

Chapter 5

Technologies Used

This section highlights the technologies that we will be using to build our project.

5.1 Web Application

We will be using Bootstrap to create our web client. Bootstrap is an easy to use open source toolkit to create web applications. We will be programming in JavaScript, HTML, and CSS and using Amazon's API gateway to interface with the main web client.

5.2 Database

In our project, we will be using a database in order to store forms, reports, logs, and the statuses of bugs. We will be using AWS in particular to handle all of our storage. We will be using a database in Amazon DynamoDB to store Python dictionaries. One database will store bug report forms containing the date sent, information sent by the client, as well as the unique code attributed to the bug. Additional entries will be created that will store additional details from the tester or developer, the current status of the bug, and who is working on the bug. We will be able to search this database based on the unique code given to each bug.

5.3 API

We will be using the Amazon API gateway to create RESTful API calls that will be able to be accessed anywhere in the world. Clients will PUSH bug reports to the database, that can be manipulated by developers and testers. The Web application will be able to PUSH updates to the database and PULL information from said database. Amazon Lambda will be used to store the back-end code. Amazon IAM handles permissions for the functions to use.

Chapter 6

Architecture Diagram

This section highlights what type of software architecture we are using for this project.

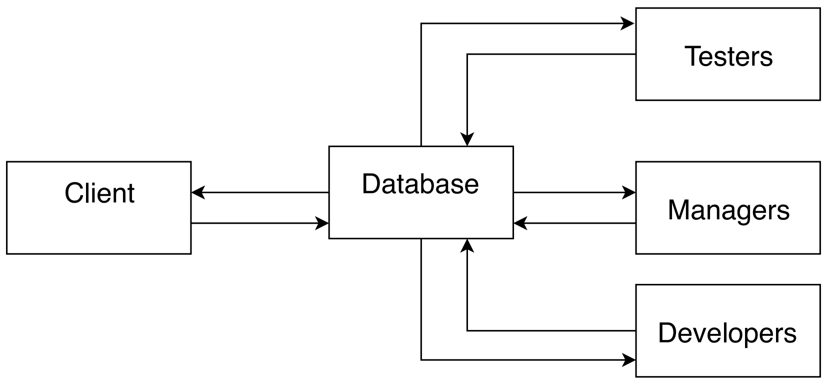

Figure 6.1: Architecture Diagram

We will be using a data-centric architecture to complete this project. All of the data processing is done on the server with clients being independent and requesting information from that server. The client interfaces with the web form, which then sends it to the API gateway. The form is saved on the database, and a unique code is sent back to the user to view the status of the bug at a later time in the opposite order. The developers, testers, and managers can interface with a separate web application that also goes through the API gateway. The data is saved on the database afterwords and can be accessed at any time by testers, developers, and managers depending on their priority levels.

Chapter 7

Design Rational

This section highlights the technologies that we will be using.

7.1 AWS Products

The reason why we have decided to use AWS as our database and API platform solution is because of the ease of use of AWS, cost, and support. AWS supports both Python and Javascript, two relatively easy languages compared to the other solutions like using the SCU databases, which requires PHP. Python is a language that we have previous experience in Programming Languages, and it is also an easy language to learn and manipulate. Using the Amazon API gateway, we can use Python to create functions that will be called using the RESTful API. REST is fully supported by AWS, and provides us with an easy way to create functions that interface with our database. All AWS products that we are using are free and comes with plenty of support online. AWS being one of the biggest cloud computing companies in the world provides us with security and piece of mind that the AWS services will always be up and running. We will be using Bootstrap to create our web client. The alternative method would be to use SCU databases and PHP, however none of us know PHP enough to be able to use this. We have previous experience using AWS, so this platform provides a familar starting ground for our project to start.

7.2 Bootstrap

Bootstrap is an easy to use open source toolkit to create web applications. We will be programming in HTML, CSS, and JavaScript, and using Amazon's API gateway to interface with the main web client. The alternative method would be to use HTML and JavaScript by itself, which would be much harder and inefficient.

Chapter 8

Description of System

Our system works like a state machine. Once the bug is created, it moves from state to state until it is fixed. Once a client reports a bug, the state is ”reported”. After this, the manager assigns a tester, and the state is ”testing”. If the tester decides that this bug is not an issue, the status is ”fixed”. If the tester decides that the bug is an issue, then the state is ”pending”. The bug then goes to the manager, where the state changes to ”issue” when a developer is assigned. Once the developer creates a solution, the state changes to ”testing”. The bug goes back to the original tester. If the tester decides that the bug is fixed, then the state of the bug is ”fixed”, otherwise the state of the bug is ”issue”. The bug then goes back to the developer, and continues to go between the assigned tester and developer until the tester accepts the solution and changes the state to ”fixed”. At any point along this process, the client is able to check on the state of the bug. The manager can check the state of the system of every bug past and present in the ”table” tab.

Chapter 9

Testing

9.1 Unit Testing

AWS has its own unit testing for every function in the backend. There is unit testing first for every Lambda function, and then again after we connected the Lambda function to the API. Unit testing it with Lambda makes sure that the code is correct, and unit testing it with the API Gateway allows us to find out if any errors in the implementation of the API.

9.2 Integration testing

After combining both the front-end and back-end code using $ajax functions in JQuery, we tested the functionality of the website.

9.3 Alpha Testing

We had various friends look over the system to check for basic functionality.

9.4 Verification Testing

We checked that the system meets the given requirements and constraints as we understood them.

9.5 Acceptance Testing

We had our wonderful TA try to break our system to no avail, proving that the countermeasures we had put in place were effective security measures for the time being. He also tested the functionality of the project to make sure that it it met all of the requirements.

Chapter 10

Difficulties

- Scheduling enough time to work on project together
- Amount of time needed to complete the project
- Learning how to use AWS
- Enabling CORS
- Working with API Keys
- Learning HTML, Javascript, and Python
- Setting up AWS parameters

Chapter 11

Suggested Changes

- Passing in API keys instead of saving them to the source code
- Ability to delete forms
- Ability to delete testers and developers
- Ability to change passwords
- Ability to sort bugs by various fields
- Improved UI aesthetically
  - Background colors
  - Different shaped buttons
  - Using more templates from Bootsnipp

Chapter 12

Lessons Learned

- Set up concrete deadlines for features
- Follow up meetings on work done weekly
- Keep naming conventions consistent
  - Dev vs. Developer
- Constant communication
- Fully determine scope before starting project
- Spend time researching the best technologies to use before starting the project

Appendices

Appendix A

Installation Guide

A.1 Setting up Databases (Only needed if setting up from scratch)

1. Sign into the AWS Console
2. Under AWS Services, search for ”DynamoDB”
3. Click ”Create Table”
4. Name Table ”bugTracker”
5. Name the Partition Key ”code”
6. Click ”Create”
7. Navigate to ”Tables” in the right hand navigation bar
8. Click on the table that was just created
9. Click on ”Indexes”
10. Click on ”Create Index”
11. Name the Primary Key ”developer” the Index Name should be ”developer-index”
12. Click ”Create Index”
13. Repeat steps 10-12 for the indexes: ”status”, ”manager” , and ”tester”
14. In the upper left hand corner, click on ”Create Table” to create another table
15. Name the table ”bugTrackerAccounts”
16. Name the Primary Key ”username”

17. Repeat steps 10-12 on this new table for the index ”permission”

18. The databases are now setup

A.2 Creating API (Only needed if setting up from scratch)

1. Sign into the AWS Console

2. Under AWS Services, search for ”API Gateway”

3. Click on ”Create API”

4. Make sure ”New API” is checked

5. Name the API ”bugTracker”

6. Click on ”Create API”

7. Under ”Actions” click ”Deploy API”

8. Click ”New Stage”

9. Name stage ”firstTest”

10. Click ”Deploy”

A.3 Copying Code into AWS Lambda (Only needed if setting up from scratch)

1. Navigate to the folder in BugTracker/backend/databaseCode/clientRoles provided with the source code

2. Open up createForm.py in a text editor and copy the contents

3. Sign into the AWS Console

4. Under AWS Services, search for ”IAM Management Console”

5. Click on ”Create Role”

6. Choose ”AWS service” for Type of Trusted Entity

7. Choose ”Lambda” as service

8. Click ”Next: Permissions”

9. Search for ”AmazonDynamoDBFullAccess” and check the box next to it
10. Click ”Next: Tags”
11. Click ”Next: Review”
12. Name the role ”MasterDynamoDB”
13. Click on ”Create Role”
14. Under AWS Services, search for ”Lambda”
15. Click ”Create Function”
16. Click ”Author From Scratch”
17. Name the function ”createForm”
18. Choose the Runtime ”Python 3.6”
19. Click on ”Choose an Existing Role”
20. For the existing role, click on ”MasterDynamoDB”
21. Click on ”Create Function”
22. Paste the code that you had saved into the ”Function Code: section
23. Change the ”Handler” to the ”name of the file”.”name of the function”
    (a) In this case, the name is ”lambda_function.createForm”
24. In the ”Designer” section, click the ”API Gateway” trigger
25. Under ”Configure triggers” pick the ”bugTracker-API”
26. Under ”Deployment Stage”, click ”firstTest”
27. Under ”Security”, click ”Open with API key”
28. Click ”Add”
29. Click the ”Save” button at the top right side of the screen
30. Open up a new browser tab and navigate to the ”API Gateway”
31. Click on the ”bugTracker-API”, and click on the resource ”/createForm”

32. Click on ”Actions”
33. Click on ”Create Method”
34. In the drop down menu click on ”GET”
35. Click on ”GET”
36. Click on ”Method Request”
37. Set ”Authorization” to ”NONE”
38. Click on ”Validate body, query string parameters, and headers” for ”Request Validator”
39. Set ”API Key Required” to ”true”
40. In AWS Lambda, take note of the inputs for the function, signified by the event[] variable.
    (a) In this function, the inputs are ”description”, ”details”, and ”system”
41. Going back to the API, click on ”URL Query String Parameters” and add a ”query string” for every input variable
    (a) Click the check boxes for each of the input variables called ”required”
42. Click on ”HTTP Request Headers”
43. Click on ”Add header” and name it ”x-api-key”
    (a) Make sure this is also checked ”required”
44. Click the back arrow named ”Method Execution” to get back to the four-paneled screen
45. Click on ”Integration Request”
46. Under ”HTTP Headers”, add a header called ”x-api-key
47. For ”Mapped from”, type ”method.request.header.x-api-key”
48. Under ”Mapping Templates”, check mark the box that says ”When there are no templates defined(recommended)”
49. Click ”Add mapping template” and name it ”application/json”
50. Copy and paste the following code into the template
    (a) Each line corresponds to a different input variable, so this template must be modified for eery function depending on the input variables

```
{
    "description": "$input.params('description')",
    "details": "$input.params('details')",
    "system": "$input.params('system')"
}
```

51. Click the back arrow named ”Method execution” to get back to the four-paneled screen

52. Click on ”/createForm” under the resources area

53. Click on ”Actions”

54. Click on ”Enable CORS”

55. Checkmark ”Default 4XX” option under ”Gateway Responses for bugTracker-API API”

56. Click ”Enable CORS and replace existing CORS Headers”

57. Click ”Actions”

58. Click ”Deploy API”

59. Repeat steps 1-55 for every function in the BugTracker/backend/databaseCode folder, except for ”makeAccount.py” and ”login.py”

A.4 Setting up makeAccount.py and login.py

1. Sign into the AWS Console

2. Under AWS Services, search for ”API Gateway”

3. Click on ”Models” in the left navigation bar

4. Click on ”Create”

5. Name the model ”Input”

6. Set the ”Content type” to ”application/json”

7. Copy the text below into the ”Model schema”

```
    {
    "type":"object",
    "properties":{
        "username":{"type":"string"},
        "password":{"type":"string"},
        "permission":{"type":"string"}
    },
    "title":"Input"
    }
```

8. Create a new model named ”Output”

9. Set the ”Content type” to ”application/json”

10. Copy the text below into the ”Model schema”

```
{
    "type":"object",
    "properties":{
        "data":{"type":"string"}
    },
    "title":"Output"
}
```

11. Create a new model named ”Result”

12. Set the ”Content type” to ”application/json”

13. Copy the text below into the ”Model schema”

```
{
    "type":"object",
    "properties":{
        "input":{
            "$ref":"https://apigateway.amazonaws.com/restapis/052stnmylg/models/Input
                ↪ "
        },
        "output":{
            "$ref":"https://apigateway.amazonaws.com/restapis/052stnmylg/models/
                ↪ Output"
        }
    },
    "title":"Output"
}
```

14. Follow steps 14-21 in Section A.3 for both makeAccount.py and login.py

15. Follow this tutorial for uploading the code to AWS Lambda using the zip file ”login.zip” in the Bug-Tracker/backend folder

   (a) https://docs.aws.amazon.com/lambda/latest/dg/lambda-python-how-to-create-deployment-package.html
   (b) The name of the function is ”login”
   (c) After uploading the code to AWS Lambda, copy the code in login.py, and replace it in the function on AWS
   (d) Repeat step 15 for makeAccount.py, but naming the function ”makeAccount”

16. Navigate to the resource ”/login” in the AWS API Gateway
17. Click on ”Actions”
18. Click on ”Create Method”
19. Click on ”POST”
20. Click on ”Method Request”
21. Set ”Authorization” to ”NONE”
22. Click on ”Validate body, query string parameters, and headers” for ”Request Validator”
23. Set ”API Key Required” to ”true”
24. Click on ”HTTP Request Headers”
25. Click on ”Add header”
26. Name the header ”x-api-key”
27. Check the box that says ”required”
28. Press the back arrow named ”Method Execution”
29. Click on ”Integration Request”
30. Click on ”HTTP Headers”
31. Click on ”Add header”
32. Add ”method.request.header.x-api-key” for ”Mapped from”
33. Repeat steps 16-32 for the ”/makeAccount” resource
34. Click on ”Actions”
35. Click on ”Deploy API”
36. The API is now fully setup

A.5 Front End Code (Only needed if setting up from scratch)

1. All front-end code is stored in BugTracker/finalProject
2. Simply drop this folder into the server that is running the website and linking a page to the ”index.html” file in this folder
3. Sign into the AWS Console
4. Under AWS Services, search for ”API Gateway”
5. In the left navigation bar, click on ”API Keys”
6. Click on ”assignTester-Key”
7. Click ”Show” for the ”API key”
8. Copy this API key
9. In ”tester.html” file in BugTracker/finalProject find the ”assignTester” function
10. Replace the API key in the ”$ajax” function with the new API key
11. Repeat this process for every function

A.6 Front End Code (If system is already set up)

1. All front-end code is stored in BugTracker/finalProject
2. Simply drop this folder into the server that is running the website and linking a page to the ”index.html” file in this folder

Appendix B

User Manual

B.1 Client

1. Enter Website
2. Choose the system being used
3. Type a description in the bug where the form says ”Description of Bug”
4. Type details about the bug where the form asks ”What were you doing when you encountered the bug?”
5. Click ”Send Form”
6. Save ”id” for use at a later time
7. Inserting this ”id” into the ”Unique Code” section of the website and pressing enter will list the current status of the reported bug

B.2 Tester

1. Login using credentials
   (a) Username: tester1, Password: password
   (b) Username: tester2, Password: password
   (c) Username: tester3, Password: password
2. Bugs that are currently assigned to this specific tester will appear
3. Testers can edit the ”tester description” by clicking the ”expand” button, and editing the description and/or view the developer description if it exists

4. Testers can edit severity by clicking on the dropdown menu. The menu is labelled by the chosen severity level, or ”Severity” if no label has been chosen yet. This can be edited multiple times.
5. If a bug is not found to be an issue, the tester clicks ”fixed”
6. After a bug is confirmed to be an issue, the tester clicks ”issue”
7. If no developer has been assigned, the bug goes to the manager
8. If there is a developer assigned, it goes back to them
9. If tester has received the solution from the developer, and the solution works, the tester clicks ”fixed” and the bug exits the development process

B.3 Developer

1. Login using credentials
   (a) Username: developer1, Password: password
   (b) Username: developer2, Password: password
   (c) Username: developer3, Password: password
2. Bugs that are currently assigned to this specific developer will appear
3. Developers can edit the ”developer description” by clicking the ”expand” button, and editing the description
4. Developers can edit severity by clicking on the dropdown menu. The menu is labelled by the chosen severity level. This can be edited multiple times.
5. After a bug solution is found, the developer clicks ”Send to Testing” to send the bug to the tester who originally tested the bug

B.4 Manager

1. Login using credentials
   (a) Username: manager, Password: password
2. Any of the bugs that need to be assigned a tester and/or a developer will be listed for the manager to see

3. Managers can assign either a tester or a developer depending on the development process
4. The manager pulls down the drop down menu on the respective bug and assigns a tester/developer
5. Clicking the "Table" tab displays all bugs, both previous and current
6. Clicking the "Create Account" tab allows the manager to create more tester or developer accounts by inputting the username and password and choosing the type of account (tester or developer)