Bug Reporting System

By: Niharika Kohli
IRC Nick: Niharika

Project Synopsis

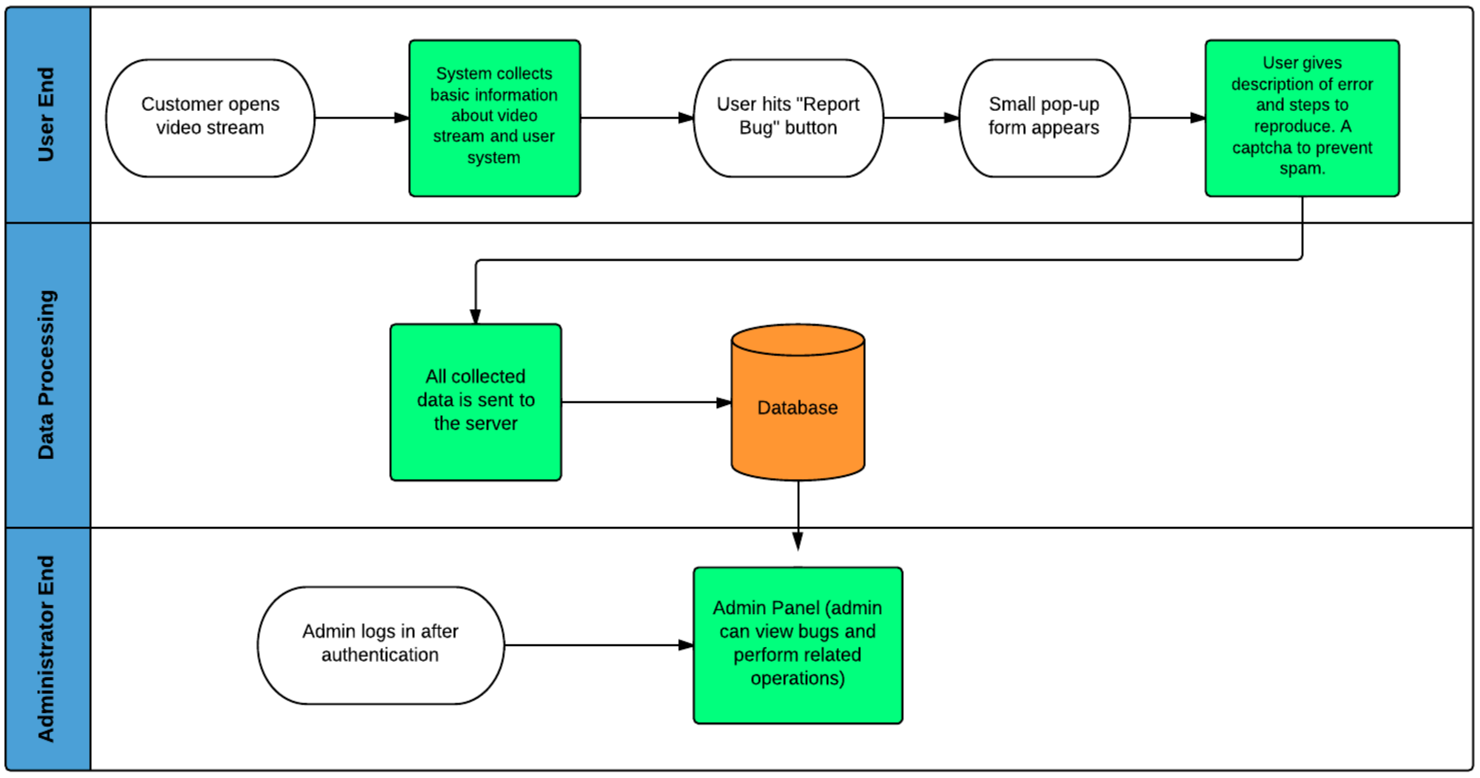
This project aims to create an automated bug reporting system for TimVideos website. Often, when users encounter laggy behaviour in the video or problems with video not streaming or incorrect resolution at the user end (because of incorrect display settings), they would like to report this as a bug. However chances are that this may not actually be a bug at the client end. For instance, the TimVideos server might be experiencing problems which may cause the video to not stream as it should. This is not a bug in the software.

At the heart of it, this project assumes the user has no technical know-how and it does automated debugging to as much extent as possible. It collects information and reports it to the Admin-end for further inspection by the TimVideos team.

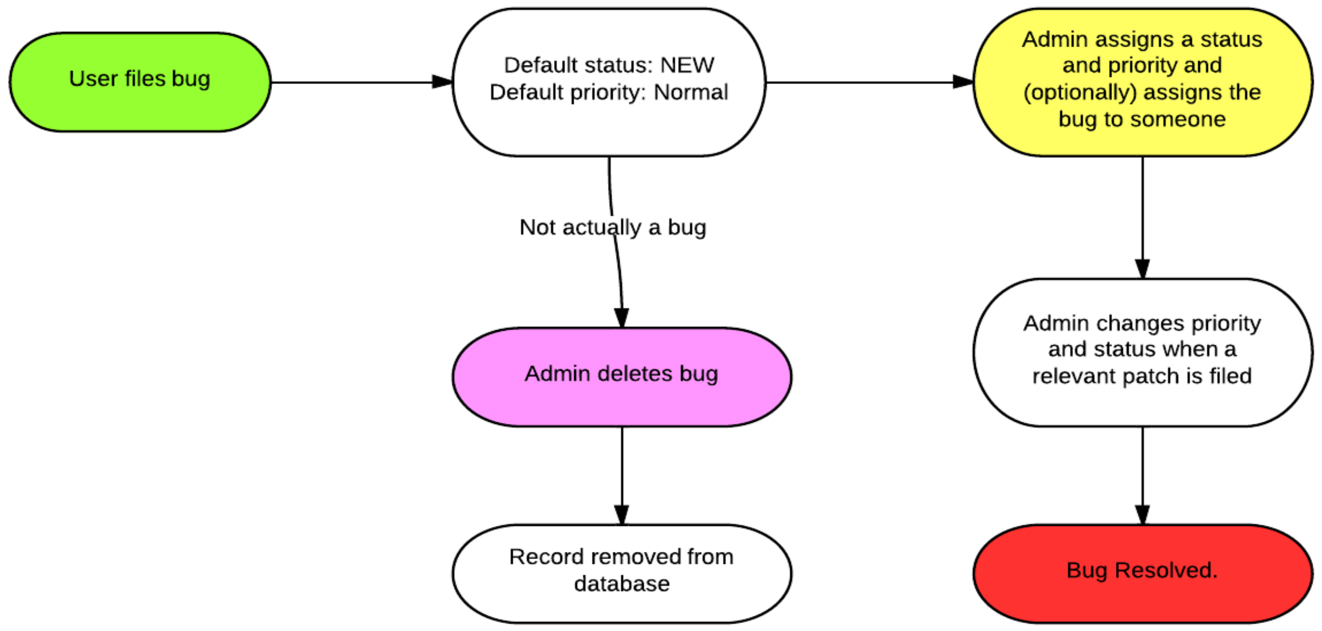
I was thinking of making it into a new module initially. But I realize that almost all data will need to be fetched from the streaming-system module. So this could very well be made into an extension of that module.

Table of Contents

Workflow for the Bug-Reporting System


User End
Data Processing
Administrator End
Customer opens video stream
System collects basic information about video stream and user system
User hits "Report Bug" button
Small pop-up form appears
User gives description of error and steps to reproduce. A captcha to prevent spam.
All collected data is sent to the server
Database
Admin logs in after authentication
Admin Panel (admin can view bugs and perform related operations)

Lifecycle for the Bug


User files bug
Default status: NEW
Default priority: Normal
Admin assigns a status and priority and (optionally) assigns the bug to someone
Not actually a bug
Admin deletes bug
Admin changes priority and status when a relevant patch is filed
Record removed from database
Bug Resolved.

Deliverables for the Project

- A user interface with a form that allows him/her to report a bug and give supplementary information along with screenshot.
- An admin interface where the admin can view bugs, change their status and priorities. The bugs can be sorted by priority, status, date of reporting etc.
- A set of programs which auto-collect data from the client end and push it to a PostgreSQL database at the server end.

Implementation Details

Questions asked from user:

- **Description** of error
- A **screenshot** of it (optional) - I plan to let the user upload a screenshot of the relevant bug if he/she feels the need to do so. Possible options for screen capturing:
    - We can use the html2canvas open source script for doing so. It's light-weight

    and fast. Not 100% accurate but much less intrusive on a user's privacy. It works only on 'canvas' elements. Since currently, our video is not a canvas element, it is not likely this will work.
    - Plugin-free screen sharing by WebRTC This gives the user a button "Share your screen". There are some drawbacks though- the user needs to set a chrome flag and restart his/her browser. This is likely to hinder users from hitting that button.
    - I am looking at PhantomJS Screen Capture too. I need to figure out how to do this at client-side though.
- **Email** (to keep clear of spam-bugs)
  The reasoning behind this is quite similar to why Captchas are required. To **avoid bot-generated bug reports**. I think **we can switch to a Captcha** instead. It will not hinder users from reporting bugs.
- **Steps to reproduce**

Automated Data Collection:

The system does automated debugging and records the following information:

- **Operating System type and version**
  This can be fetched using JavaScript quite easily.
- **Browser type and version**
  This will also be captured in JavaScript.
- **Internet connection speed**
  There are quite a few open-source APIs that do this. I also found simple JavaScript codes online for achieving this. I found Boomerang and SpeedOf.Me. I am planning to give both of them a try and make the decision based on performance. I could write my own script if both of them do not match upto our expectations.
- **Resolution of video and other user-settings**
  From my research the above details can be obtained from the stream.js and whats_on.js located in streaming-system\website\frontend\static\js directory.
- **Date and Time of bug-reporting**
  Easy to capture in JavaScript as well.
- **Location (Geolocation using IP address)**
  Doable in JavaScript provided user allows it else can be estimated using IP address.
- **Information about video frames**
  See https://github.com/timvideos/streaming-system/issues/44

- **Video metrics collected from jwplayer**
  See https://github.com/timvideos/streaming-system/issues/12

Some or all of the above information might be collected before the user actually hits the 'Report Bug' button

Information collected is sent back to the server by means of a Python framework, Django

Information is recorded in a PostgreSQL database

I propose having two database tables:

User supplied information

Field	Description
BugID	Auto-generated unique ID for each bug
Bug Summary	Description given by user(mandatory)
Steps to reproduce	Provided by user(optional)
User Email/Captcha	Provided by user(optional)
Screenshot	Captured by us
Similar bugs	A link which displays all similar bugs. The link will be a number. Number of similar bugs.

Automatically collected information

Field	Description
BugID	Auto-generated, same as in above table
Name of video stream	

OS name and version	
Browser name and version	
Detected internet speed	
Video Resolution	
Video Format	1. Flash 2. HTML5 3. Justin.tv 4. Audio only
Video Quality	1. HD 2. SD
Date and time of bug reporting	
Geo-location	
Bug priority	1. Critical 2. High 3. Low 4. Normal 5. New
Bug status	1. Resolved 2. New 3. Assigned
Assigned to	

Similar-bugs are aggregated together

- Implement an algorithm to aggregate bugs together if they are
  - Reported close to each other (Say, within an hour or two) and
  - Have similar keywords in Bug summary
- Every bug will have a “similar bugs” field which holds the number of similar-bugs. Clicking on the number will take you to a listing of all the similar-bugs and their details.

The admin(s) can access the complete list of recorded bugs and perform the following functions:

- Dismiss bugs which are not really bugs (they will be deleted from the database)
  - There will be a confirmation button(“Are you sure?”)

- Assign bugs a priority (Critical, High, Low, Normal, New-the default priority)
- Assign bugs to someone
- Mark bugs as Resolved.

Security and Testing

Unit Testing

- Test the admin panel
- Test the aggregation of bugs
- Extensively test bug priority and status handling
- Check working on mobile website
- Automated testing of JS files using Qunit

Integration Testing

- Test if the module works on all browsers as expected
- Create use-cases and check functionality

Security

- Secure the inputs against cross-site scripting and SQL-injection attacks

My only experience with testing has been manual. But I would love to learn more about how automated testing works. I shall spend the time I get before the project officially starts to research automated tests and how I can get them to work on this project.

Timeline for the Project

Before official start of project:

1. Complete the development environment setup including PostgreSQL
2. Brush up my Django skills
3. Create PostgreSQL database and necessary tables
4. Familiarize myself with community and take design-inputs for the User and Admin interfaces
5. Nail down exact requirements and finalize it through discussions with mentor(s)

19th May to 29th May:

1. Create User and Admin frontend- JavaScript, HTML, CSS and AJAX.

30th May to 10th June:

1. Write code to take operating system details and browser version details as input- JavaScript and jQuery will be needed.
2. Record the date/time of bug submission
3. Test and full-proof the above data

11th June to 30th June:

1. Fetch the debug-log, if any is created.
2. Write a script to test connection speed or use an existing API.
3. Fetch information from user-settings: resolution of video, HD/SD, video format etc.

----------At this point of time I will be done with the client-end coding--------

1st July to 20th July:

1. Implement functionality to allow admin to dismiss(delete) bugs
2. Implement functionality to give bugs a priority- Critical, High, Normal, Low, New (default)
3. Allow to set a Bug Status- Resolved, In progress, Dismissed

21st July to 18th August:

The most demanding and important phase of the project- Testing and Deployment.

1. Write test cases
2. Solve bugs
3. Take community feedback
4. Repeat

I have given this almost a month because it will be the most crucial phase of the project. Everything, from User interface to Admin interface, information storage, etc will be tested thoroughly to weed out as many bugs as possible.

Open Questions

- Will the Database tables be merged into the existing Database (if any) or we shall create a new database?
    - I could work with either. The answer is dependent on the answer to the question below. If this is merged into the streaming system module then the database tables should be too.
- Would this be better done in a new module or merged into the streaming-system one?
    - It will be better to isolate this from the streaming-system module in my opinion. It is easier to handle and debug smaller components. But this might turn out a little harder seeing that a lot of information is fetched from streaming-system module.
- For the bug screenshot- does user upload it himself or we take one by ourselves?
    - Status: According to discussion with Tim on IRC, many users might not know how to take a screenshot or might not be willing to do so. Capturing one ourselves is a better thing to do.
- Captcha or Email or neither?
    - People would be more willing to file a bug if it's a Captcha instead of an email
- Who all can report bugs? Some form of authentication?
    - I think it should be open to all then, without any authentication.
- Is there any need to split the information in two tables? Should I maintain only one table instead?
    - No preferences as such. One table might make more sense than two.

Future Considerations

- Connect the web-app to the github issue tracker
- Connect to the IRC channel and echo new bugs on channel
- Allow comments on bugs
- Optimize functioning on mobile-browsers
- Sorting of bugs
- Searching of bugs

About Me

I am Niharika Kohli, a third-year Information Technology undergraduate student from Indira Gandhi Institute of Technology, New Delhi, India. I have a keen interest in web-development and design. My experience with Python is limited to algorithmic programming but I feel that I can learn it for web development quickly. As GSoC is as much of a learning experience as a programming one, I hope to have both this summer.
Languages I can program in: JavaScript, Python, C++, HTML, CSS, PHP
Languages I can speak and write in: English, Hindi

Past Projects & Open Source Experience

OPW (Outreach Program for Women) intern for Wikimedia Foundation

(10 December, 2013 - 10 March, 2014)

My project involved customizing the languages list seen on every wiki page and showing only those languages which are relevant to the user. For example, if you are in India, you will see Hindi, English, Punjabi etc and the rest of the languages will be collapsed in a "More" list which is sorted by region and searchable.

- Previous language choices are also stored and placed in list
- Languages used: PHP, Javascript/jQuery and CSS
- Link to code on Gerrit
- Project Progress Report can be found here
- Project Proposal can be found here
- My blog posts for the project can be found here: http://niharika29.roon.io/
- Currently the feature is in Beta and can be accessed if you enable it in the Beta Features tab after logging into any wiki (provided the wiki allows it)

Web-Development Contest hosting portal (June, 2013)

This project was done as part of a college workshop. Teams of 5 or less people can sign up and work on a given project till the given submission deadline. Then the respective evaluators can rank the websites. After the results are declared, each team gets auto-generated certificates.

- Languages used: PHP, JavaScript, HTML, CSS
- Database: MySQL
- The code can be found on GitHub: Link

- This project is live and hosted here. (Kindly retry after sometime if it does not open immediately. It is hosted on a college server which goes down if there is a power outage.)

Contact Information

Name: Niharika
IRC nick: Niharika on Freenode
Email: ...
Preferred means of communication: IRC, Hangouts, GTalk, Email etc.
Typical Working Hours: GMT+5:30 timezone. Working hours: 3 am to 7 pm, UTC
Blog: http://niharika29.roon.io/