Google Summer Code 2016 Proposal

Oppia

Making Oppia usable offline and being more lightweight in transmitting data

Vishal Gupta

Email: [redacted]

Github URL: https://github.com/gvishal

Resume: [redacted]

Project name:

Making Oppia usable offline and being more lightweight in transmitting data.

Project description:

This project contains two parts:

- **Make Oppia Usable Offline**:

  Oppia currently requires users to be connected to the internet at all times while playing explorations. We need to allow playing some explorations without an internet connection. Now, there are multiple things that happen during an exploration: Statistics Recording, Answer Classification, Response Selection and State Transitions. Most of these things happen in the front-end, but we require to communicate with the back-end when it comes to Statistics Recording, Answer Classification and somewhat for Response Selection. Also, we need to fetch the exploration from the back-end. Thus, we need to add persistent front-end storage to enable playing explorations offline.
- **Make Oppia Lightweight**:

  Oppia sends considerable amount of data while playing an exploration and also

when visiting the website. I have attached reports for the same. Most of them are tidbits of improvement which together should make it much more smoother. (Report: Main Page: https://goo.gl/fE3rHz https://goo.gl/AMT4CG Pitch Perfect: https://goo.gl/SJ6DtI https://goo.gl/xSjzVn). We need to minimize the data transfer to enable its use on poor connections. One measurable outcome, could be to fix all the things that the reports mention. This should give us about 3-4% of data savings and with good caching we could do better for return users.

Expected results: Users will be able to play some Oppia explorations without any Internet connection. Also, there is a measurable decrease in data sent between the browser and the server when visiting any pages on Oppia.

What interests you about this project?

When I first tried oppia, I started with the Pitch Perfect exploration. I wanted to know, how oppia was able to handle all the possibilities of answers that a user could enter. Curiously, I opened Chrome Web Tools and checked for any activity in the Network tab. I noticed that there were a bunch of requests being sent, further observation showed that this happened every time I submitted an answer. Then, I saw the list of project ideas on the wiki and this happened to be the first on the list. I immediately knew that this is what I want to do this summer. So, I did a little more digging. Going through the back-end routes it was hitting, I noticed that it's just logging statistics. And, I thought that we should be able to play the exploration even if I disconnect it from the internet. But, then I noticed another event i.e answer classification, happening. Going through its back-end logic meant that we couldn't play this exploration offline. But, when I saw the idea description, it said "play some explorations offline". This meant that we could enable certain explorations to be played offline.

Oppia is a wonderful tool for self-learning, it's like a game which you could play endlessly. In many places around the world, Internet connectivity is not really great or is a little expensive. This restricts people from being able to use the wonderful opportunities that are out there on the internet. Enabling offline usage of Oppia helps us reach many more

people out there and make a larger impact. For me, this is an important thing and it is what attracted me towards this project.

What areas of research will you do in this project (if applicable)?
I have researched various front-end storage solutions. I have also researched techniques to decrease data transfer which involves exploiting caching techniques. There is a good amount of scope in making Oppia lightweight and hence I will spend a good amount of time to research and test which techniques can help us. I've also looked into how Electron can be used to build a completely offline desktop app for Oppia.

Outline of a project plan and implementation schedule.

Proposal

There are two parts to the projects:

Offline Oppia:

I have two different proposals for offline Oppia:

1. **Idea 1: Desktop app for Oppia (using Electron: https://github.com/atom/electron):**
   My earlier idea is mentioned below and it had various flaws when it came to using Oppia offline. A major one was that it was pseudo-offline i.e, we'd require internet to make the basic requests to get the barebones which could be later filled using css/js/json we had saved locally in the browser from earlier visits. In my opinion, this wouldn't be too useful. So, here's something else that I thought about. We can instead make a cross-platform desktop app for Oppia using web technologies. In a nutshell this involves making available all of Oppia's resources offline as an entire package that interacts with the server when the need arises (fetch new/updates explorations, css/js files, etc.). **Due to time constraints my research might not be as thorough as I'd want it to be (it's highly abstract), but I believe using Electron to build a desktop app is the best foot forward when it comes to make it more accessible.**

We will be imitating most of what Oppia is right now for the Desktop app. We will be able to use most of the already written code for the front-end (See: Angular + Electron). This would mean designing it in such a way that it enables us to integrate any changes that happen to the Oppia's front-end automatically without any changes to these assets. We will have to create wrappers around the existing functionality to enable offline access. This wrapper should be such that it allows us to incorporate any change that happens at the front-end along-with providing offline access. We write a sandbox on top of the current front-end implementation. Now, every request for any resource goes through this sandbox. The sandbox is responsible for providing all these resources. It could save all these resources offline or fetch them from the server. It is like a big box of cache you can say.

2. **Idea 2: Save explorations offline in local storage to enable offline play:**
   Right now in the situation in which internet cuts out, we could continue playing as long as there is no "classify_answer" request made. The idea proposed here ensures that we can play the same exploration next time around.

    a. **Save explorations offline to local storage**
       Whenever a user wants to start a new exploration, it is fetched from the back-end. This requires unnecessary data transfer if the user has already played it before and/or the exploration has not been updated since the last time, the user played it. We will store explorations in client-storage using LocalStorage (supported by all the browsers) along-with its version. We wii also store rich-text components required to play the exploration along with the js/css files. LocalStorage has different storage limits (Link) depending on the device and browser, ranging from 2MB - 10MB. An exploration typically takes around 5-20KB for the Json content and another ~1MB for the different components required for it. If we store all the components for an exploration, we will be able to store only a limited amount of explorations. LocalStorage does not have limit on how large an entry can be storage as long as the storage quota is there. If the the exploration changes we fetch the new

exploration. We will not provide the user to download an exploration, it will be totally transparent to the user.

We will stringify and store all the data in LocalStorage. We will write getter/setter to convert the data into the required format as per our requirement.

b. **Load exploration**

Whenever a user starts an exploration, two cases arise:

i. **Exploration is available offline:**

In this case, we need to make sure that the exploration hasn't changed since the last time a user played it. (If internet is available) We make a request to the back-end to check the exploration version. It is necessary to ensure the version is up-to-date as otherwise the resulting statistics are unusable. We compute a checksum to verify whether or not the exploration at the front-end is correct and identical to the back-end. Ben pointed out that we have one backend controller for the player page itself, and another for grabbing the exploration dict. And, the player page gives us the exploration version that is available. (See: Link)

If exploration hasn't changed, we load it from LocalStorage. Else, we goto step b.

ii. **Exploration is not available offline:**

We make a request to the back-end to fetch the exploration. We save this exploration at the client storage and proceed with the exploration. This requires modifications to the PlayerServices.js file in **oppiaPlayerService/init** method. We will add another service which will handle entire interface for getting explorations. This service will store explorations, check if they exist and get the exploration for the initialization.

To test this load exploration in the case of no internet, we can swap out the http interceptor with a custom one which always returns an

error(404). This would trick the service in loading the exploration from local storage.

3. **Reporting statistics (Store at front-end and send all at once)**
   There are multiple(five) events during an exploration for which we send statistics to the back-end. These are mentioned in the StatsReportingService.js file. All five events are independent of any interactions with the exploration being played. The request for recording a submitted answer is also independent wrt the next state (unlike a classify_answer event) and we can safely send it later. These are gathered to perform analytics. We can safely store all the statistics at front-end and send them all at once whenever we connect to the internet. The statistics sent later on will have a version associated with them. To keep track of when the user is online, we can periodically check for connectivity and whenever the user is online to send the statistics. Storing stats has two benefits:
   i. It enables us to play an exploration and gather that data when we are not connected to the internet.
   ii. If we stagger the stats and send them together, it will help us conserve data transfer (since we won't be sending redundant information and will minimize data transferred in HTTP headers). This can be done even if the user is connected to the internet, in this case we can stagger say 'x' statistics before sending to the back-end. This requires changes to StatsReportingServince.js and addition of back-end api's that accept aggregated statistics. We are going to add two services for handling the stats: one to save them locally and another to sync it with the backend. I have read that there are a few performance issues when it comes to LocalStorage as data is written to the disk, so we'll need to do some more research on this front.

Lightweight Oppia:

1. **Support partial response and partial update, avoid sending entire resource representation.**

Currently, we send keys' with empty values to the back-end as part of a request. We can avoid sending these to save data transfer. Let's take this (http://pastebin.com/7KgZAbBz) request json for the Lazy Magician exploration. Not all keys are set in the data sent.

2. **Images**:
    a. Same image is fetched multiple times. Eg. every exploration has some authors associated with it, and if multiple explorations have the same people, it fetches their images again. This image data is sent as base64 in a json object. Refer: ProfilePictureHandlerByUsername method in controllers/profile.py file. (screenshot: Link1, Link2)
    b. There is no caching for (some) images. These include images sent in base64 + json.
        i. We can store this data in the LocalStorage and avoid requesting again.
        ii. We can serve the json as a js file, that can be cached.
3. **Cache Best Practices:**
    I had used chrome web tools and google's page insights to analyze oppia. They showed that some of our resources(images) have short cache expiration times. Also, we embed many scripts into the html body before rendering. Requesting them independently of-course leads to delays but enables us to cache them. So, we'll have to see what's better. Also, refer to above point. There are a no of guides available using which we can tune the caching of different resources.There is a good scope of improving the cache behaviour of current oppia website. Some resources/guides:
    a. https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching#caching-checklist
4. **CSS/JS Minification and Segmentation:**
    a. Not all css and js files are minified.
        i. **Oppia.css** could be minified.
        ii. The js files embedded in the **explorationplayer.html** as part of the <script> tag are not minified. This page is pretty heavy and minifying

all js files embedded in it could lead to data savings. We could also consider providing the js files separately so that they can be cached by the browser. Requesting them independently of-course leads to delays but enables us to cache them for return users. So, we'll have to test what's more useful.

5. **Tips provided by chrome audit:**
    a. Serve static content from a cookieless domain. We avoid sending unnecessary data i.e cookies, to retrieve static content. This, saving on some data.
    b. Many resources are not cached, we could consider caching them if possible. Some resources have short lived cache expiration. And, referring to point 2 and 3 above.
    c. 90% of the css is unused. I tested for the homepage and an exploration using Chrome Audit Tool. (Link)
6. **Measures for Lightweight Oppia:**
    I think performance testing should be helpful here.
    We could measure amount of data transferred, time taken being variant of so many factors. To measure cache behavior, we can measure data sent second time around.
    There are some tools that we can help mimic low-bandwidth connection and some companies also provide different mobile devices for testing apps. I have worked with one such company(see http://www.robustest.com/), where I had helped set up Facebook's augmented traffic control which helps mimic different bandwidth's. I'll try to find some modules out there which do such stuff for us. How about we have an ambitious goal eg. Oppia shouldn't require more than "X MB" of data transfer?

Idea 1 Timeline

First Trimester (April 22st - May 22nd, 2016)

Due Date	Goal	Measure
22nd April - 2nd May	- *	-

3rd May - 10th May	Refine project plan based on discussions with the mentor. Set up Trello Board for project.	Updated Documentation. Add necessary content.
11th May - 18th May	Set up milestones and issues for the project on Github. Create design docs for the features proposed.	Additions to Oppia repo. Design docs for all features.
18th May - 23rd May	Get a start on code to stay ahead of the planned schedule.	

* Due to my final exams, little to no work can be done during this time. Also, I figured since I have my holidays beginning from May, I'd start early. It could be useful in case I didn't foresee anything. I would want to get the design docs and milestones set up on Github to make things easier.

Second Trimester (May 23rd - June 28th, 2016) (timeline not finalized)

Implement Idea 1 (Desktop App).

Third Trimester (June 28th - August 24th, 2016) (timeline not finalized)

Complete Idea 1. Work on lightweight oppia, if time persists.

Idea 2 Timeline

First Trimester (April 22st - May 22nd, 2016)

Due Date	Goal	Measure
22nd April - 2nd May	- *	-
3rd May - 10th May	Refine project plan based on discussions with the mentor. Set up Trello Board for project.	Updated Documentation. Add necessary content.
11th May - 18th May	Set up milestones and issues for the project on Github. Create design docs for the features proposed.	Additions to Oppia repo. Design docs for all features.
18th May - 23rd May	Get a start on code to stay ahead of the planned schedule.	

* Due to my final exams, little to no work can be done during this time. Also, I figured since I have my holidays beginning from May, I'd start early. It could be useful in case I didn't foresee anything. I would want to get the design docs and milestones set up on Github to make things easier.

Second Trimester (May 23rd - June 28th, 2016)

Due Date	Goal	Measure
23rd May - 3rd June	Implement offline storage and loading of exploration for front-end. Implement/augment back-end endpoints required to verify the exploration.	~~PR to the Oppia repo.~~
4th June - 12th June	Check correctness and write tests for offline exploration(both front-end and back-end).	PR to the Oppia repo.
13th June - 17th June	Work on code reviews. Update design docs to reflect the current state.	Improved PR.
18th June - 28th June	Implement Reporting Statistics module. Buffer Week (Prepare for Midterm evaluation). Work ready for submission for evaluation.	Offline oppia ready to be deployed!

Third Trimester (June 28th - August 24th, 2016)

Due Date	SMART Goal	Measure
29th June - 5th July	Write tests for the new Statistics module. Test the new offline Oppia.	PR to Oppia repo.
6th July - 12th July	Get started with making Oppia lightweight. Improve cache behaviour for different elements. Improve caching for images sent in json.	PR to Oppia repo.
13th July - 17th July	CSS/Js Minification and Segmentation.	PR to Oppia repo.
18th July - 25th July	Improve all API calls to minimize data transfer.	PR to Oppia repo.
26th July - 2nd August	Audit and improve Oppia using Chrome Audit and other tools.	“Bug fixes” PR to Oppia repo

3rd August - 10th August	Fix possible bugs found.	PR to Oppia repo.
11th August - 17th August	Buffer Week(Refactor Code/Fix Bugs)	Improved PRs as per code reviews.
16th August - 24th August	Documentation*	Updated Docs for Oppia.
24th August - 30th August	Final Evaluation	Completed Project.

*Before starting I would have all the different design docs. I figured I'd update them at the end too. Also make any changes to the docs on oppia.github.io ?

Contributions to Oppia

Worked on guard against sending duplicate emails #1416, which led to this PR being merged with the codebase.

Related Technical Skills

Please outline any prior experience you have, especially where it relates to technical skills that are needed for the above project. (Depending on the complexity of the project, we may write back and request more information in this category.)

I have worked as an intern at Srujana Innovation Center, Hyderabad where I was involved in a project in collaboration with Camera Culture Group, MIT Media Labs. We built iLabelit-a gamified web app to crowd-source annotations for retinal images, which will be used to train machine learning algorithms for automated screening of retinal pathologies. I was responsible for its full-stack development and deployment. We used the MEAN(Mongo, Express, **Angular** and Node) stack to develop it. I learned and employed various development practices that make it easier for different people to develop and extend the app. We decoupled the back-end and front-end, using **REST Api's**, allowing us to develop apps for different platforms without any hassle. I have gained experience working with asynchronous services and tackling concurrency problems. I also wrote deployment scripts, that made sure we had the latest version of the app running whenever we pushed changes. We had to store some information relating to the user at the client-end for which I had explored a number of **front-end storage** techniques like Local

Storage, Session Storage and Application Cache. We ended up using Local Storage for our needs.

Earlier I had taken a Web Development course at Udacity which through which I learned **Google App Engine**. I made a few basic apps(1,2) using GAE and had also worked as an intern with a startup where I used GAE to prototype an idea.

Apart from this I have worked on quite a few academic projects(refer to my CV/Github) where I have learned different skills. Most of the projects that I have done, have involved with working with something new. It was really intimidating the first few times but now I love taking on new challenges. I have learnt that the best way to learn is by doing and preach this to everyone.

Open Source and Education

Why are you interested in open source development? Why education?

I have been using various linux distros since I was in high school. Once in my school library, I read how can package your linux distro with the tools you require pre-installed. That began my tryst with open-source software. I believe that technology can solve most of our problems and it is sort of a superpower that us humans have. Computers can do what humans cannot (vis-à-vis), and the amalgam, gives rise to unimaginable things. Being able to create something and see its impact is joy for me, code enables me to accomplish it, using only a computer. Developing software, I get to work with interesting people to provide solutions which impact everyone around us. Technology is an evolving area and I love tackling challenging problems everyday.
I have been itching to contribute towards an open-source initiative. But, somehow or the other I would just get bogged down at the initial step. The clarity of Oppia's documentation totally removed that hurdle. The issue that I worked upon was clearly described, this made it a breeze to implement it. I have really enjoyed contributing to Oppia and will continue to do so.

I have always loved teaching and helping out people. I have also been a Teaching Assistant for two courses - Software Engineering and Computer Networks. I have always

explored alternate exciting ways of learning. I tried out the Pitch Perfect exploration on Oppia and was immediately hooked to it. All this has really attracted me towards education (learning/teaching).

To be honest about it, I have never enjoyed as much as I have while working on Duplicate Email Prevention issue. Completing the issue and seeing it being merged with the codebase, has given me tremendous confidence. Also, It was great to have all the help from the mentors esp. Sean, who provided with good feedback on my commits. I always kept learning and re-iterating until it was perfect. This experience has strengthened my interest towards open-source software and Oppia in particular.

Summer Plans

- Which time zone(s) will you primarily be in during the summer?
  I will be in Indian Standard Time (GMT +5:30).
- How much time will you be able to commit to this project?
  I will be contributing 7hrs a day, 6 days a week (42hrs/week) during 23rd May till 10th August. From 11th August till 23rd August, I will put in 6hrs a day, 6days a week (36hrs/week).
- Please list jobs, summer classes, and other obligations you may need to work around.
  I have my summer vacations from May - July. I have no obligations during this period. College starts again in August first week. So, for ~15 days in August, I will be working on the project as well as attending classes. I only have 15hrs of classes every semester and another 10hrs for doing the homeworks. So, that leaves me with plenty of time to work on this project.

Communication

- What is your preferred method of communication and contact information?
  I find using Oppia's Gitter chat extremely easy. Communicating over email is also perfectly fine.
  Github handle/ Gitter username: gvishal

Email Address: [redacted]

Phone Number: [redacted]

Skype Id: [redacted]

- How often do you plan on communicating with your mentor?

  I plan to send weekly updates over email to my mentor. Apart from that, I will remain connected via Gitter and discuss any relevant things.