ByteMap: Phase 1 Design Document

Changes made to Design Document:
- all changes are in green color
- added cron deletion to context diagram
- modified Offering, Vote, and Pin key concepts to reflect updated design and critique feedback
- updated “low barrier of participation for Byters” feature
- creation of offerings through email design challenge

OVERVIEW:

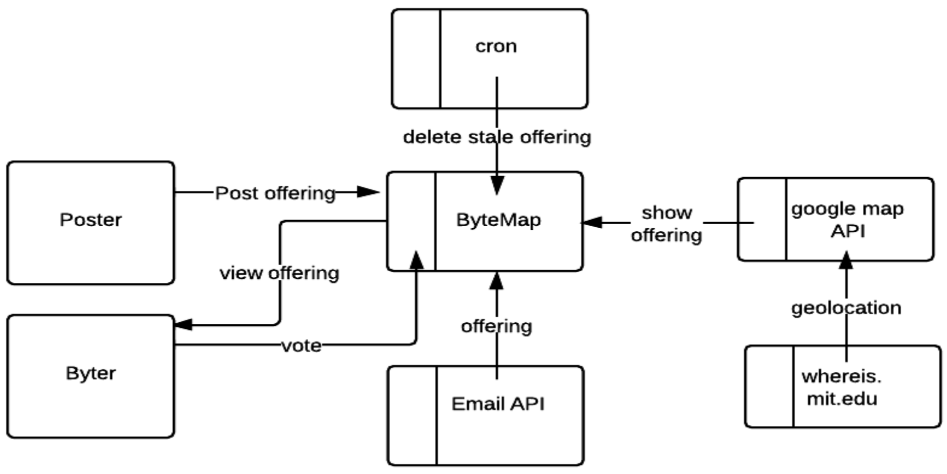
Purpose and Goals-Nitya

Motivation
ByteMap’s primary purpose is to eliminate the high-volume, low-relevance entity that is the MIT free food mailing list (and every other list that is occasionally used to announce the presence of free food on campus). Oftentimes, we receive emails announcing free food at times we don’t care about, or at locations that we won’t be able to get to before food has run out. And if we manage to get to the free food, it’s sometimes all gone, making the time and effort required to get there wasted.

Insight
Our major insight into this problem was to create a visual that organizes the varied times and locations of sent emails into one easy to digest visual: an MIT map. Using a familiar, intuitive interface, users can view free food that is in places they care about at times when they are in a position to take advantage of it. We think that a visual approach will eliminate the need for noisy email lists and instead allow users to look at offerings when they need them.

Goals
Our goal for ByteMap is to provide an easily accessible, intuitive interface for users to check where free food currently on campus is located. We would like to leverage existing channels of communication(the current free food email list) to ease integration of the app, and we want to make the app very usable with a large, active user base so that we can leverage crowdsourcing to provide real time information regarding free food locations and status.

Context Diagram- ido



CONCEPTS:
Key Concepts -Cynthia
Map: An interactive map representing locations around MIT's campus. A user can view the locations of buildings and zoom in to explore pins representing various food offerings currently in that location.

Offering: An instance of available food and its description, represented via a pin dropped at it's corresponding location on the map. Offerings only exist for a fixed duration of time, to be cleaned by either a cron job or are removed earlier than that duration if an eater indicates that it is no longer present.

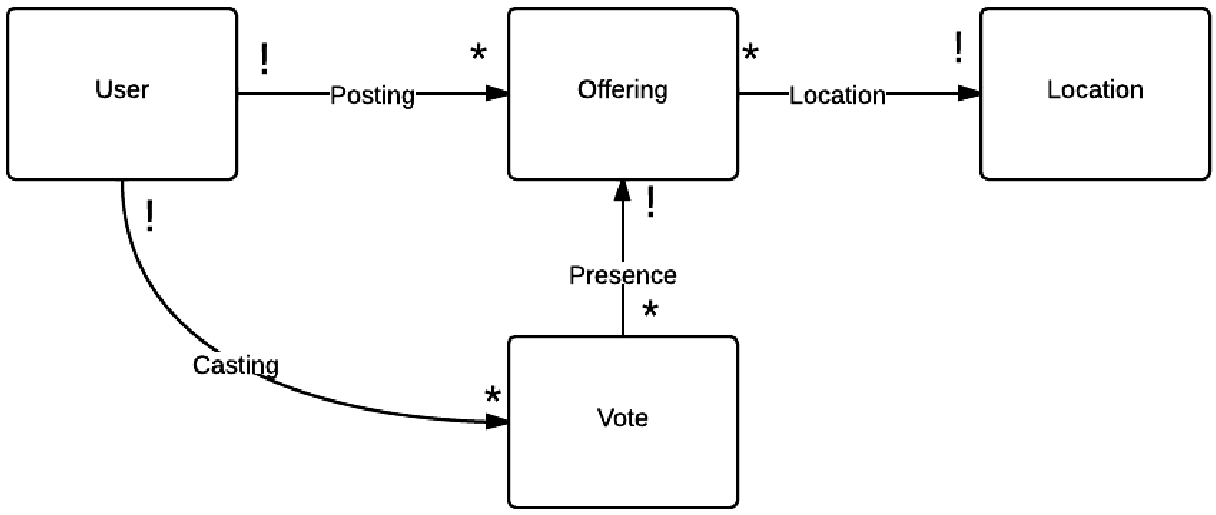
Vote: an indication if an offer is still available at the location or not.

Pin: Icon representing a location that can be seen on the map. There can be many offerings at the location of the pin. Each location has a custom ID gotten from whereismit.

Poster: Member of the MIT community who creates offerings.

Byter: Member of the MIT community who can view the offerings on the map. They can assess the pin of an offer and vote on the state of that offering.

data model - ido



BEHAVIOR:

Feature Descriptions - Cynthia
Visualization of all available food on campus: A map with pins allows byters to visually see all available food offerings. This lets byters easily determine the proximity of offerings through a geographical representation of all the information.

Flexible input methods (manual or email): Through either an easy input form on the app or through directly emailing to a list ByteMap subscribes to, posters of offerings will not have to spend significantly more time to enter their offerings to the app. ByteMap will be able to receive emails from relevant food lists and scrape that information and create an offering.

Crowdsourced verification of presence of offerings: Byters can indicate if an offering is all gone by a simple action to the offering. If enough Byters verify that the offering is gone, ByteMap will remove the offering.

Quality control for posters: Posters must sign in to add an offering. This allows ByteMap to keep track of information on the creators of offerings. ByteMap will keep track of the relevant mit email address of offerings automatically created through an email. This provides the same amount of quality control as when users currently email out about free food on mailing lists.

Low barrier of participation for Byters: Unlike Posters, Byters (eaters) are not required to sign-in. This means that they can instantly see all available offerings, lowering the barrier to participation.

Furthermore, all information is centered on the map page and requires no actions of the part of an Byters.

Minimal Viable Product- Nitya
For our minimal viable product, we have decided to implement our basic Map and Pinning functionalities without users. In this phase, we will allow for anyone to post offerings, view offerings, or make vote on the status of offerings(mark if food is ‘still there’ or ‘all gone’). The primary challenges of this step will be integrating the Google Maps API and making the user interface intuitive. For this phase, we will only be allowing postings to the list via a form on the website, saving the added complication of parsing emails for a later iteration.

Design Challenges and Considerations- Nitya

Authentication: Users
We extensively debated whether or not to have users(or what types of users we would have). We want to make the app easy to get started with because we want to encourage as high of an active user base as possible, and therefore don’t want to burden our users with signing in. We do, however want to have a way of controlling the quality of content on our application, and want to therefore have some barrier to entry, especially for posters of offerings.

We will implement a standard login procedure that requires posters of offerings to make an account. Because the risks associated with low content quality are much higher on the poster side, requiring an account might drive away spam more effectively than allowing open posting. On the other hand, we will not require accounts for viewers of offerings (Byters), as the sign-in process would be very cumbersome for as dynamic an application as ours. In order to ensure quality control of viewer interactions of offerings (i.e. crowdsourced indication of whether an offering still exists), we will use a redundancy model, requiring a minimum number of indications from distinct users before an offering is removed from the system. We plan to use session hashes as a way of determining whether viewers are distinct, thereby getting around the necessity of user accounts.

We have decided to implement a standard login for this phase of the project. We allow users to view offerings and mark them “all gone” when they are depleted, and when they view an existing location with offerings, we do not give them the option of adding a new offering at that location. Users that have not logged in can still interact with the map by searching for locations, but instead of having the option to create an offering upon viewing a location, they are instead prompted to login first. Once logged in, a user may then post a new offering at a location.

Updating Offering Status: (click button vs upvote/downvote)
One important aspect of our application is notifying users when it is no longer worth the trek to get an offering that is completely depleted. We plan on having a default duration time-out for all offerings, but we would also like to crowdsource the task of reporting offering status. We considered two methods of doing this: notification buttons, and an upvote/downvote system

The notification buttons ‘still there’ and ‘all gone’ would be clickable by consumers of the offering, and would serve as an additional source of information for prospective consumers. If a consumer obtained food from a source and noticed that there was more food left, they could click the ‘still there’ button, and users would be able to see that at the time the ‘still there’ button was last clicked, food was still present at the given location. The other button we would have would be the ‘all gone’ button, which could either be clicked by the last person to get food or by someone who happened upon the empty food containers at the specified location. We could then augment the specified duration feature by taking into account the crowdsourced data we get, allowing for a more realistic picture to be presented to our users. One challenge of this option would be how to appropriately weight the crowdsourced data such that we don’t trust one downvote too much and remove an offering that still exists.

The upvote/downvote system would be somewhat simpler to implement, as it would just have users give positive or negative points to an offering. However, this system was not time-sensitive enough for our specific use, and we were also unsure as to how we would properly weight upvote/downvote data in order to determine if an offering was still valid.
For these reasons, our team decided to move forward with the 2 button offering feedback system for the MVP.

Lifespan of an Offering
One other challenge we faced was determining the duration for which we would keep an offering on our map. As described above, we will consider the duration input by the user as well as the crowdsourced feedback received and keep the offering up for as long as either parameter requests we keep it up(we will take the max of user feedback and scheduled cleaner duration). This does mean that occasionally we will keep offerings up that are no longer valid, but we think that this small variation will still be a major improvement over the current system, which gives very little intermediate feedback(typically one announcement and sometimes one termination email are sent for a given offering)

How to Post an Offering
We debated three different methods to post offerings to our website- emailing to free-food@mit.edu(the current mailing list), bytemap@mit.edu(a new email we would create), and having a form submission on the site. We will definitely have a form on the site to begin, but as we progress, we will integrate emailing to bytemap@(with a specifically formatted email), and if we have time, we will then try to parse the unformatted messages sent to free-food@.

Location controller vs. Offering controller modularity
Because we use whereis.mit.edu’s API to obtain MIT building/location information on the fly, the lack of a pre-existing list of MIT locations meant that any particular location should only be created at the time the very first offering at that location is created by a user. This raised the issue of whether the location creation logic should be placed entirely inside the offering

controller’s create method.

We decided it would be best to keep both location creation and offering creation inside the offering controller (for the final project, not the MVP). Although keeping them separate would maintain better logical separation of concerns, this would also require redirects between the two controllers: offering#create would first instantiate an offering, then redirect to location#create, and finally the instantiated offering would need to update its location_id field based on the location object created.

For the MVP, we kept location creation logic in the location controller because, for simplicity, users allow locations in an initial step before they can create an offering at that location.

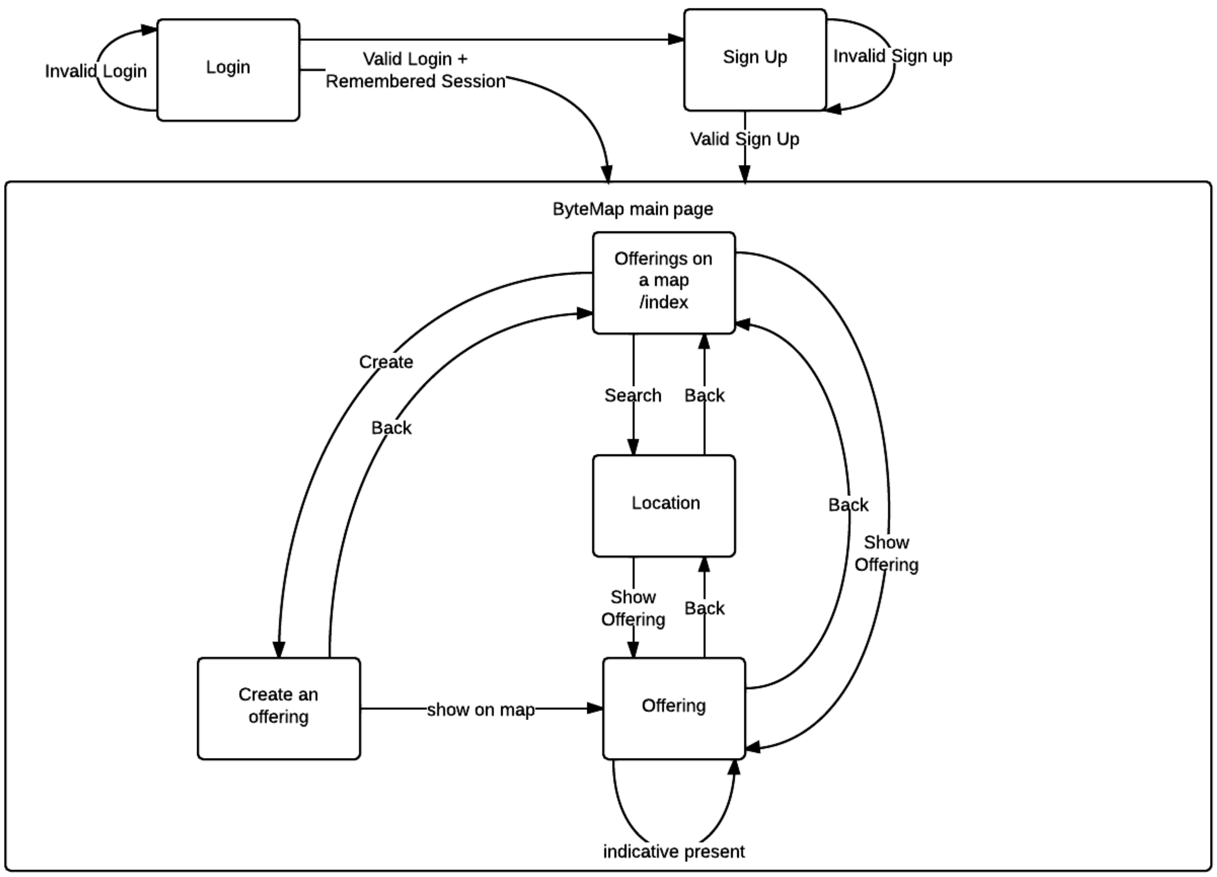
Pin Location/Food Availability Representation
In the MVP, we assign pins to represent a specific location and then allow users to add food offerings to existing pins or to create new pins defining new locations on MIT’s campus. This current system is fully functional, but creates a situation where we have pins on the board without any offerings currently attached to them.

We considered other options, including not displaying pins on the board if they didn’t hold any offerings, but decided against this implementation at this phase because we did not want to create complications with creating multiple copies of a location in our database.

In the future, we would like to establish a more fluid system for representing pins on our board only when we have food items currently located at that location.

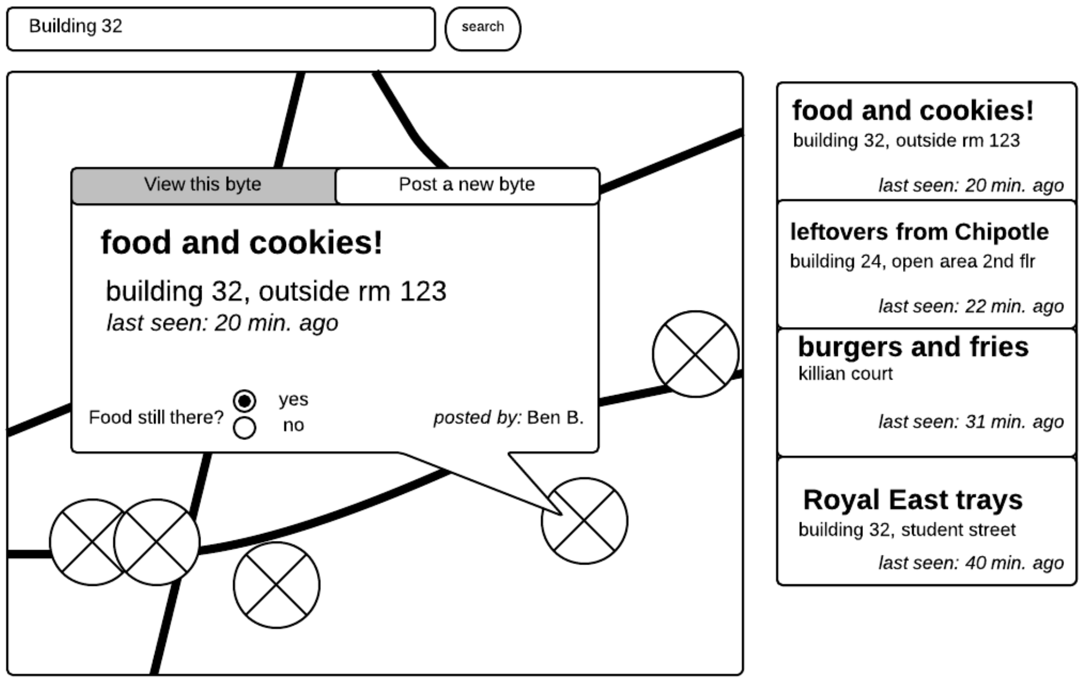
Creation of an Offering from Email
ByteMap has it set up so that it has an unique email that can receive any emails in a parsed JSON format. Because this transfer of information is through an email, there is no other user to create an offering. Because of this, all creation calls must come from the server side.
Server side creation: With the use of the rest-client gem, the server can make ajax calls to whereismit to get the needed location information. From there, the controller can make the needed calls to create an location if it has not already been created, and create a new offering.

state machine -ido

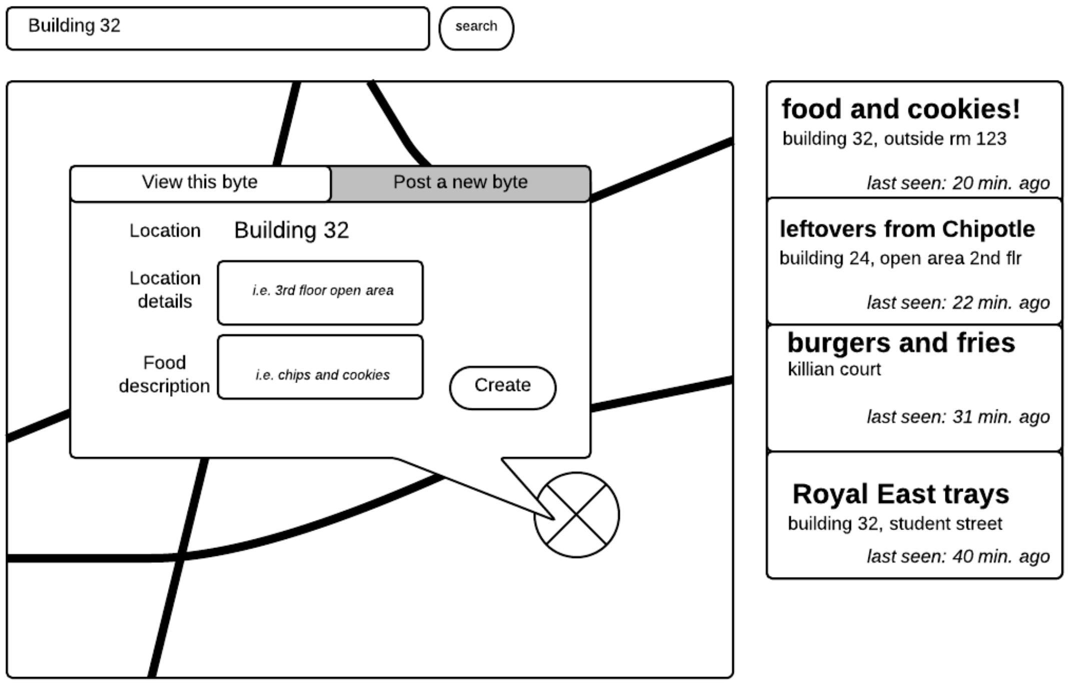


User Interface and Wireframe - Carrie

Interface for posting a new offering (“byte”)



Interface for viewing an offering:



Security Issues - Cynthia

Some security requirements are that:

-Byters and posters must be part of the MIT community

-Posters must use a valid MIT email/certificate when posting

-No one needs to see the information on a poster, byters verifications are anonymous

Potential Risks:

- Trolls placing faulty “pins” or reporting that food is still there, to misguide other eaters.
- Byters faultily reporting that food is all gone to keep the food to themselves.
- Hackers who attempt SQL injections, cross-site scripting, CSRFs

Mitigations:

- Standard rails/ruby strategies to address code vulnerabilities (such as injection, XSRF, etc)
- Attribute offerings to a creator (poster) so that posters can be held accountable if need be.
- No important information is stored and all users are assumed to be part of the MIT community, so there is already a level of privacy and trust between users

- Use access control to allow only MIT people to see and post to app.
- Crowdsourcing verification of the presence of offerings will allow users to quickly indicate if an offering is gone. Since offerings are so transient, only a few people are needed to indicate an empty offering for it to be removed from the site.

Teamwork Plan:

Stakeholders: In order for ByteMap to become successful, we rely on members of the MIT community to both use and post to the app. Specifically, stakeholders might include hungry students/faculty in search of food, department administrators who frequently announce free food, and facilities staff who may be concerned about unattended leftovers or food sanity. Lastly, the team members of this project are the administrators of the Bytemap system.

Resources: We will use mainly our laptops to create the app and heroku to deploy it. If we get MIT Certificate validation working, we will host our app on scripts and use the scripts locker of one of our team members to host the application.

Tasks:

	phase 4.1	phase 4.2	phase 4.3	phase 4.3 (following week)	phase 4.4
Carrie	Wireframes	customization of GMaps to MIT buildings	server pull requests, cron job	Parsing real-world emails, search, testing	Parsing real-world emails, testing
Cynthia	Features, Key Concepts, Security Concerns	Parsing custom template, voting	UI/UX	testing, Parsing real-world emails	Parsing real-world emails, testing
Nitya	Design Challenges/ Main purpose and goals	Creating offerings via form, Google maps API integration	login/ certificates	newsfeed, testing	newsfeed, testing
Ido	State machine, data model, context diagram, database design	Google maps API integration, bootstrap integration	UI/UX	testing, search	Texting (optional), prezi presentation

As stated in the team contract, we will spend all the time possible to finish our own tasks, but if there is an instance where we can't finish our personal milestones, we will tell the rest of the team a day or two beforehand so that the whole team can work together to finish the milestone.

Risks:

Design risks:
Unreliable pins, either of food that is no longer exists or ones that were posted by a troll, might put off users. If there are constantly offerings without the advertised food, users might become unhappy with ByteMap for wasting their time. We have timestamps and a voting system to remove offerings as soon as they are consumed or deemed unreliable.

Implementation risks:
Push notifications: As ByteMap depends on real time updates, it is important to synchronize users as fast as possible.

Development risks:
Complicated map APIs: We will be using Google API, which is well documented and has many developers. If this feature proves infeasible, we can go back to a static map of the campus.
Getting content from emails: Email scraping may be difficult to achieve a high rate of success. We are allowing user to post directly to the site, so there is an alternative source of inputting offerings.