Milestones

Milestone 0: Describe the problem that your application solves.

Smartphones and tablets have been adopted widely among coders, but there isn't yet a convenient way to read code on them. Both Dropbox and GitHub, two popular services used to store code, do not have mobile applications or websites which are optimal for code reading on the go.

Numerous apps and services exist to allow users to edit code, either natively or in the browser. However, editing is a task much better suited to a full-fledged computer than a mobile device. Our application aims to solve the problem of reading code on the go, regardless of device or platform.

Milestone 1: Describe your application and explain how you intend to exploit the characteristics of mobile cloud computing to achieve your application's objectives, i.e. why does it make most sense to implement your application as a mobile cloud application?

m(arc) aims to allow users to have the features and convenience of a modern text editor - syntax highlighting, code folding, and their choice of colour scheme - in a web browser and on the go. m(arc) can also connect to Dropbox and GitHub to display code files stored there.

Our app makes sense as a mobile cloud application instead of a native app, as it allows users to have the same experience regardless of platform or device. In addition, we sync user data (the files they've imported) and preferences to our servers, allowing them to persist their actions across multiple devices: a user can import files on their smartphone and view them while on the go, then pick up in the same place on a tablet or computer once they've reached their destination.

Milestone 2: Describe your target users. Explain how you plan to promote your application to attract your target users.

Our target users are developers just like us: people who often find themselves in transit or away from a computer and who want or need to review code files for any reason - business or pleasure. A lot of the code they view comes from GitHub, which has a less-than-ideal mobile site.

Our target users will also be acutely aware that there are a number of mobile code editors available today, both native or on the (mobile) web. Though some also offer integration with GitHub and other sources, they mainly focus on editing: a task we feel isn't well-suited to a mobile device, and is better served by the existing development environments available on computers. m(arc) hence focuses on making the code reading experience as convenient as possible.

We plan to promote m(arc) through forums, such as Hacker News. We are targeting places where code is discussed and shared among individuals. Visitors to these sites care about code, and make up the group of our key target audience.

Milestone 3: Pick a name for your mobile cloud application. *(Not graded)*.

m(arc): m(arc) reads code.

Milestone 4: Draw the database schema of your application.

We are using MongoDB via the Mongoose object modeling library, so our schema has no relations, and consists simply of:

```
var userSchema = new mongoose.Schema({
    githubId: { type: Number, index: { unique: true}},
    githubTokens: { type: [String] },
    fileList: { type: String },
    preferences: { type: String },
    updatedAt: { type: Date, default: Date.now }
});
```

Milestone 5: Design and document (at least 3) most prominent requests of your REST API. The documentation should describe the requests in terms of the triplet mentioned above. Do provide us with a brief explanation on the purpose of each request for reference. Also, explain how your API conforms to the REST principles and why you have chosen to ignore certain practices(if any).

GET /authenticate/:code

Gets access token from GitHub, since GitHub prevents us from implementing its OAuth Web Application Flow purely on client-side.

Params:

code: the temporary code provided by GitHub as part of its OAuth Flow

Return value:

{ token: 'jca0088480ac72d519b66620bca186a109a5f541' }

Error:

Forwards the error from the GitHub API if any.

GET /user

Gets the authenticated user

Headers:

Authorization: token jca0088480ac72d519b66620bca186a109a5f541

Return value:

{
  "_id": "524439cd8b0ba948affb612c",
  "fileList":
[{\"path\":\"joyent/node/.gitattributes\",\"source\":\"github\",\"id\":\"github/joyent/node/.gitattributes\",\"metadata\":{\"sha\":\"a185d697627566fe7e991acb9c26f82b4939cc3c\",\"user\":\"joyent\",\"repo\":\"node\",\"relpath\":\".gitattributes\"},\"cached\":true},

…
]
  "updatedAt": "2013-09-26T13:50:32.834Z"
}

Error:
If token does not match any existing user, the API returns 401 unauthorized.

PATCH /user
Updates the authenticated user

Headers:
Authorization: token jca0088480ac72d519b66620bca186a109a5f541

Params:
{
  "fileList":
[{\"path\":\"joyent/node/.gitattributes\",\"source\":\"github\",\"id\":\"github/joyent/node/.gitattributes\",\"metadata\":{\"sha\":\"a185d697627566fe7e991acb9c26f82b4939cc3c\",\"user\":\"joyent\",\"repo\":\"node\",\"relpath\":\".gitattributes\"},\"cached\":true},
…
]
}

Return value:
{
  "_id": "524439cd8b0ba948affb612c",
  "updatedAt": "2013-09-26T13:50:32.834Z"
}

Error:
If token does not match any existing user, the API returns 401 unauthorized.

Since we only need to persist each user's data for our app, our REST API is quite minimal. The only resource is the user, accessed via the /user endpoints. Instead of using the more commonly used PUT verb for updating users, we have chosen to use the relatively recent PATCH HTTP verb, which is recommended for usage when a resource is supposed to be partially modified. As opposed to PUT, which mandates the entire resource being replaced by a fresh representation, we can choose to modify only specific fields to use less bandwidth, which could lead to a better user experience on mobile devices. This complies with the semantics of the PATCH method, keeping the API RESTful while still suiting our needs better.

Milestone 6: Tell us some of the more interesting queries (at least 3) in your application that requires database access. Provide the actual SQL queries you used.

Here are the Mongoose queries, which map quite closely to how it would be done in the MongoDB console.

```
User.findOneAndUpdate({ githubId: user.id },
    { $addToSet: { githubTokens: token } },
    { upsert: true });
```

This query finds the one user with the given GitHub ID and adds the given token to their set of GitHub tokens. It is an "upsert", so it would update if the user already exists, otherwise it would insert a new user.

```
User.findOne({ githubTokens: token }, 'fileList preferences updatedAt');
```

This query finds the one user with the given GitHub token and retrieves only the fields fileList, preferences and updatedAt, similar to a SELECT query in SQL.

```
User.findOneAndUpdate({ githubTokens: token }, update).select('updatedAt')
```

This query finds the one user with the given GitHub token and updates with the given object, and returns the user with only the updatedAt field.

Milestone 7: Find out and explain what [QSA,L] means. Tell us about your most interesting rewrite rule.

When we use the RewriteEngine, we can choose to change some part of the URL to part of the query string. This new query string will override the original query string of the URL. The [QSA] flag stands for Query String Append, and thus tells the RewriteEngine to combine the old and new query strings instead.

The [L] flag tells the RewriteEngine that if this rule matches, it should stop processing the rule set.

We are using nginx. Since m(arc) is a single page app, we don't really use any rewrite rules for it. However, we do rewrite http uris to the corresponding https uris, for security reasons:

```
server {
        listen 80;
        server_name marc.beng.me;
        return 301 https://$host$request_uri;
}
```

Milestone 8: Create an attractive icon and splash screen for your application. Try adding your application to the home screen to make sure that they are working properly.

Milestone 9: Style different UI components within the application using CSS in a structured way (i.e. marks will be deducted if you submit messy code). Explain why your UI design is the best possible UI for your application.

Sidebars

Both sidebars can be retracted to allow the central CodeView to take up the full width of the device. This provides more real estate on the mobile device for the content.

Buttons

All the buttons have a minimal size of 44px x 44px. This provides the user with ample space for interaction[1]. The controller is no longer a precise pointer, but a big fat finger.

Navigation

m(arc) is a single-page web application. There are 4 main navigation paths.

1. Switching between different files.
2. Downloading files from the server and cloud services.
3. Authentication.
4. Changing different options in Settings.

We followed certain guidelines found in the iOS developer guide[2].

When the user is redirected out of m(arc) and back, the previous state is presented. This allows the user to continue from where he left off.

Minimal user distraction was considered when designing the in-app navigation. Two examples include switching between different files and changing the options in the Settings pane. Users are not interrupted with modals or different screens, and can continue interacting with the rest of the application.

Milestone 10: Implement and explain briefly the offline functionality of your application. Make sure that you are able to run and use the application from the home screen without any internet connection. State if you have used Web Storage, Web SQL Database or both in your application. Explain your choice.

m(arc) synchronises the user's file list to our backend periodically (roughly every thirty seconds). When offline, the app continues to track changes made to the file list, but (obviously) doesn't sync with the server. Upon regaining a connection, the app will resume synchronisation.

The app's reading capabilities remain unaffected by the user going offline: when they add folders or

---

[1] https://developer.apple.com/library/ios/documentation/userexperience/conceptual/mobilehig/LayoutandAppearance.html

[2] https://developer.apple.com/library/ios/documentation/userexperience/conceptual/mobilehig/Navigation.html#//apple_ref/doc/uid/TP40006556-CH53-SW1

repositories, we download and store a copy of each file (instead of accessing the file on demand each time.) The user cannot, of course, add new folders or repositories while offline.

To ensure the app remains usable while offline, we have made use of WebStorage. Since we are largely storing plain text strings (representing cached copies of online files) or simple JSON objects and do not need any complex SQL joins or other such features, WebStorage serves our needs better than other solutions and in a much simpler way, being more performant for this use case.

There was one more alternative we felt was worth considering for m(arc)'s particular needs: the HTML5 FileSystem API. It would have been very well suited to storing all the data that we use, but it is also currently only available on Google Chrome. Using this API would have negated one of the key benefits of working on the mobile web: being able to support a wide range of platforms and devices instead of being limited to a particular subset of them.

Hence, we chose localStorage. For lack of a better alternative and to get around the ~2.5MB limit there is on localStorage, we created a library to use localStorage over multiple domains in hidden iframes, storing and retrieving items via cross-domain messaging.

Milestone 11: Explain how you will keep your client in sync with the server. Elaborate on the cases you have taken into consideration and how it will be handled.

If a user has logged in with a GitHub account, m(arc) uses simple periodic polling (on a thirty second interval) to sync their file list to the server. The synchronisation process consists of retrieving the server's copy of the file list and checking the "last modified"-equivalent timestamp included with it. If the server's copy of the file list is more recent than the client's, we simply replace the client's copy. Similarly, if the client's copy is more recent than the server's, we upload it to the server.

Any change a user makes to the file list (regardless of if they are offline or online) results in the client's timestamp being updated - this allows us to persist changes they've made while offline to the server once they resume their connection.

As m(arc)'s main use case only involves reading (and so the only actions a user can take are to add new files or remove existing ones from the app), we feel this simple method of conflict resolution is sufficient.

Milestone 12: Compare the pros and cons of basic access authentication to other schemes such as using cookies, digest access authentication or even OAuth. Justify your choice of authentication scheme.

Basic Access Authentication

Pros: easy to implement

Cons: no control of authentication UI. Need to use HTTPS, otherwise it is not secure (and not accepted by some browsers over HTTP).

Cookies

Pros: Can control authentication UI. Session cookies can be set to persist so that users don't have to

log in every time.

Cons: Extra effort to implement. Session cookies can be easily compromised if used over HTTP.

Digest Access Authentication

Pros: easy to implement, no need to make any UI. More secure than basic auth over HTTP.

Cons: no control of authentication UI.

OAuth

Pros: Can piggyback on other platform's OAuth authentication. Saves users from having to create another account, and gives them peace of mind and the security of not giving someone else their password. Also saves effort having to create an account creation/management page. Can be revoked if compromised.

Cons: Reliant on third-party service - if they go down, the user might not be able to log in if they are not already logged in. Some users may prefer to have accounts that are not linked to an OAuth provider. Can be tricky to implement, and has multiple version standards.

We chose to piggyback on top of GitHub's OAuth for our own authentication. Given our target audience, it is quite likely that they have a GitHub account. Since we are making requests from the client directly to GitHub's API, we have to store GitHub's OAuth token in some form on the client anyway, so reusing it for our own authentication is no more or less secure, since we are doing so over SSL. Doing so allows us to treat our own API as an extension of GitHub's API, just with different URI endpoints, and reuse the same authentication logic in our requests.

Milestone 13: Describe 1-3 user interactions within the application and explain why those interactions help make it better.

Code View

Interactions in the code view are similar to those found in popular code editors, minus the ability to edit code. Reading code is the **main** feature in our application. The intuitive interactions meant packing in more without confusion for the user.

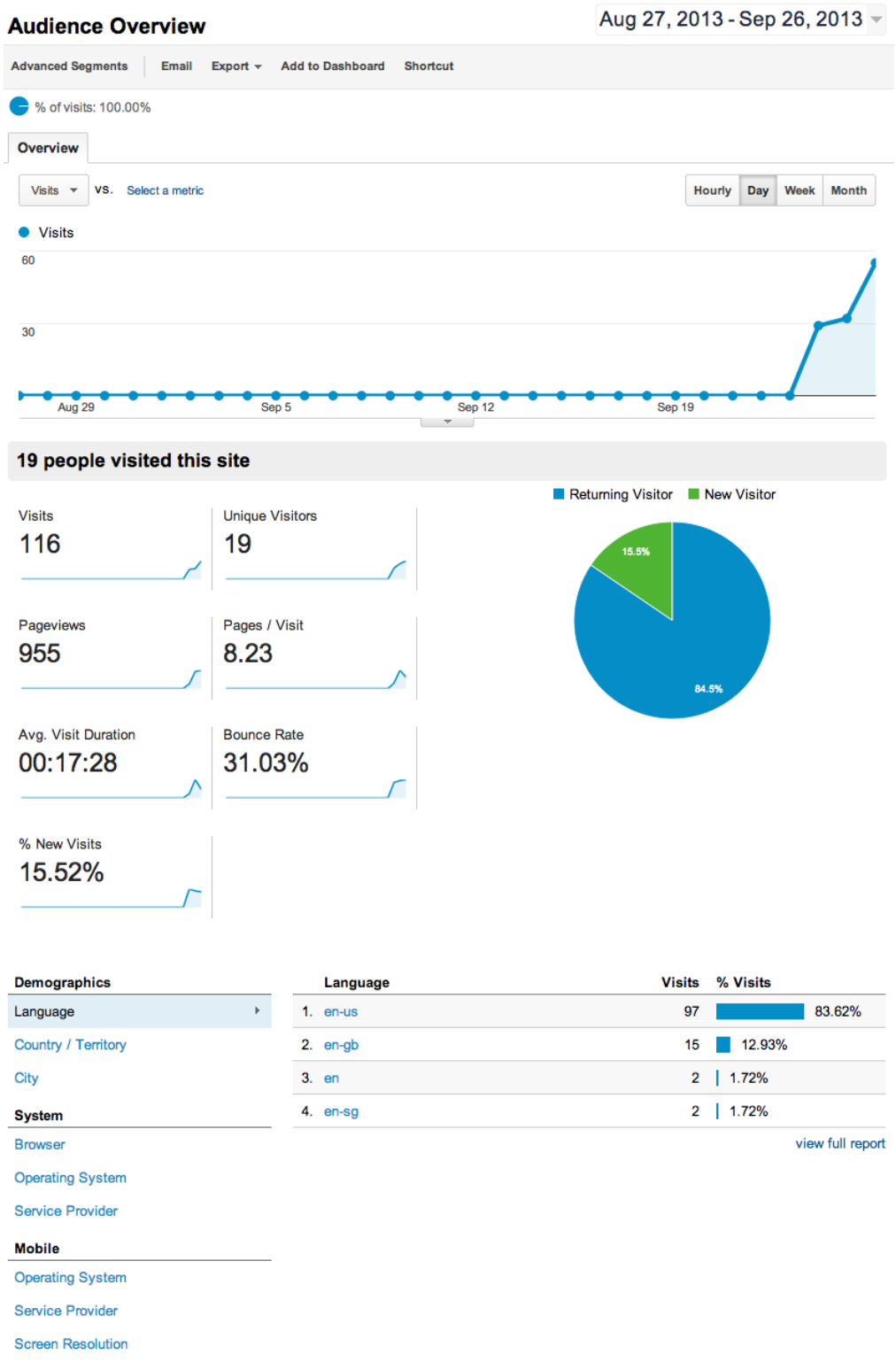
File Tree

The file tree was designed to emulate the interactions of existing file systems. This makes it easy for the user as there is no learning curve involved when navigating between different files. This was the important as file navigation was the second main feature in our application.

Sidebars

Toggling the visibility of the sidebars allows more real estate for the content when needed. The single button toggle is easy for the user, and reduces clutter on the User Interface. They also responsively show or hide themselves based on the horizontal resolution available, so that the user would not even need to toggle them if they have enough screen space. Of course, they can still choose to hide the sidebar if they wish.

Milestone 14: Embed Google Analytics in your application and give us a screenshot of the report. Make sure you embed the tracker at least 48 hours before submission deadline as updates are reported once per day.



Milestone 15: Identify and integrate any social network(s) with users belonging to your target group. State the social plugins you have used. Explain your choice of social network(s) and plugins. (Optional)

Our target users are likely to also be users of GitHub (a.k.a. the social network for coders). We have thus integrated GitHub login to allow them to access repositories hosted on the site; we also make use of this to sync with our backend.

We have also integrated Dropbox as these users often use Dropbox as an easy way to bring files that are on their PCs to their mobile devices, and they may not want to set up a repository on GitHub just to read some simple code files.

Milestone 16: Make use of the Geolocation API in your application. Plot it with Bing or Google Maps or even draw out possible routes for the convenience of your user. (Optional)

Not applicable. Our application does not have any convincing use for Geolocation.

Milestone 17: Justify your choice of framework/library by comparing it against others. Explain why the one you have chosen best fulfils your needs. (Optional)

We have not used a mobile-specific framework/library, as their interfaces are purely targeted at mobile clients and we did want to target all form factors (despite our emphasis on the mobile use case). Even though some frameworks are very usable on desktops, they still look a bit out of place.

Hence, we are using Twitter Bootstrap 3 along with a responsive design, with widgets such as bootstrap-select and bootstrap-switch which are more familiar on mobile and yet still usable on the desktop. We have used the fastclick library to eliminate the 300ms lag that many mobile browsers have while they are waiting to detect a double touch, and the snap.js library to provide sliding sidebars similar to Facebook and Path's.

The enquire.js library is used to respond to CSS media queries in JavaScript, so that we can adapt certain aspects of the interface without the relatively more expensive "classic" method of listening to browser resize events. For example, the sliding gesture for the sliding sidebars is disabled on larger form factors, and the left sidebar containing the file tree is left open for the user's convenience.

The careful integration of all these libraries allows us to have a good experience across mobile and desktop devices, while having a lighter combined footprint for those libraries than most mobile-specific libraries.