UNIVERSITY OF KANSAS

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

EECS 448 - Software Engineering I
Fall 2020
Project 4: MAPKU
(Team 14)

Team Number: 14 Date 10/31/2020

Deployment Plan:

Before we go ahead and deploy MapKU, we would like to work more on the testing part, try to find more bugs and fix them, make it as easy as possible to interact with the users. We are using Google Maps Javascript API—for the map and other features like Information box, Markers, etc., Google Directions API—to get directions and the route from point A to B, and Google Places API—to autocomplete the address. Google gives a free monthly limit of $200 and it is like free allowance up to 100,000 loads of Static maps, 28,000 loads of Dynamic Maps, 28,000 panos of Static Street View, 14,000 panos of Dynamic Street view. Currently, we fall under all these limits, so it doesn't cost anything as of now however, as we move forward and add more features, we might have to pay for the APIs. We would first like to deploy MapKU as a website (using a hosting company like bluehost or Greengeeks). The cost of hosting a website will be $80-$120 approx. and a domain name would cost $30-$60 approx. (This is just the cost of hosting the website maintenance will cost much higher than this and it is explained in the next section). Initially, we are hosting this as a website but if we see a good response, we would consider hosting it as an app. Hosting an app in IOS costs onetime $99 fee and hosting an app in android costs onetime $25 fee. Cost of developing an app depends on various things like (functionality, UI requirements, Data Storage, etc.) so it is hard to tell the exact cost, but it would be approximately $50,000-$80,000 to hire an app developer to do all the things.

The potential market for this product is all the students, faculty, and staff at the University of Kansas. MapKU is very helpful for the students (especially freshmen) to locate their classes and to get around the campus. Before the first day of classes, students usually go around the campus and locate their classes, so they don't miss anything on the first day of their class. But it is not always possible for everyone to do that because of variety of reasons. MapKU helps students do the exact same thing on their devices and also give them an idea of what their daily route will look like. Students can all their add their classes and location of their housing and MapKU will give them proper time approximations of how much time it is going to take for them to reach their classes every day. MapKU will be highly in demand among out of state students and freshmen since they are new to the campus, they would want to explore more and learn more about the campus. Out of state students can see what their daily commute will look like even before visiting the campus. It will also be very useful for the faculty because there are a lot of people commuting from different places (eg. Lenexa, Kansas City, etc).

The best way to market MapKU would be through organizations in the university. We can hand out pamphlets (that contain all the information about MapKU) to various organizations like KU visitor center, ISS, AAP, KU Engineering and request them to hand these out to freshmen in orientations and also put it up on student boards. We will also try to get this approved by the university and put a small box or a hyperlink in KU portal. Apart from pamphlets, we can also use google advertising (initially starting with a budget of $1000- $2000) and it would be a big plus because we can make the algorithm target more people around the campus.

Maintenance Plan:

The Software Development Life Cycle (SDLC) refers to a methodology with clearly defined processes for creating high-quality software. SDLC consists of five focused phases

of any software development: Requirement analysis, Planning, Software design such as architectural design, Software development, Testing, and Deployment. However, the proposed plan for the software products doesn't always turn out to be perfect when it meets reality and that's when the Software Maintenance plan emerges. The real time factors change rapidly and so the maintenance plan accommodates the fluctuation in the world by updating and advancing the software to match the fluctuations. The project that Team 14 is working on has different functionalities that requires maintenance over the years. The main functionalities that Team 14 has implemented in project four ranges from searching the database to find classes to utilizing Google Map Application programming interface (API) to calculate the shortest distance between buildings and provide the user with directions to their dissentions. Team 14 progressed with the project as team 14 created a friendly user interface to help users maneuver over the University of Kansas (KU) campus with no problems. Fortunately, Team 14 doesn't have to maintain any changes to the routes, new KU building, or new path as google maps API will take care of any of those changes for the team. The main concern that Team 14 is dealing with is the functionality of finding the building name from the class number provided by the user. This functionality was implemented by first creating a web crawler (selenium) to obtain the information from the official KU classes (classes.ku.edu) to be exported to an excel sheet. This information included course, number, course title, course topic, course number, section number, min credit hours, max credit hours, seats available, total enrolled, etc. The next step for team 14 was to store all the information obtained from the excel sheet to a database. The final step was fulfilling the promise every time a user enters a class number and from the object obtained being able to get through and add the building to the route list. The maintenance plan for this specific functionality will require team 14 to go through the web crawler every semester to obtain the new courses and the different locations and so on. This is required because data on the database is not automated, in other words, the database is not linked to the KU website but rather connected to the excel sheet. The excel sheet needs to be updated every semester to be able to have the new information transferred over to the database. The maintenance plan plays a major rule in our project and requires at least one developer to access the project every semester to update/run the web crawler to obtain the new information for each class. The maintenance plan doesn't only stop here as the project will be viewed very differently when published and stress tested. The foreseen updates that Team 14 is expecting would be the abilities that users would always be able to see their location , adding class schedule sets, adding more methods of transportation, and traffic congestion control.

Team 14 expects more maintenance and more updates as users will have different perspectives and will be able to determine what is necessary. The cost to hire a web develop for maintaining a website $68,524/yr. The database is another cost to be considered, Firebase was used to store the information of class in this project. The storage cost is split into two main parts: Realtime Database and Cloud Firestore. Realtime Database has a subclass called Document reads which costs about $0.06/100K. There are 28,447 in KU and the average number of classes that students is enrolled in every semester is 4; assuming that each student will search for the 4 classes every day. This is estimated to be have 28,447*4= 1,153,788 reads from the database, and so this would cost a total of about $0.80. For the Realtime Database, the cost for a GB/yr is $5, which is sufficient to store all classes. Finally, it costs about $731.94 per year to run an average server.

Code integration explanation:

Our team utilized the top-down integration for implementing MapKU. Top-down integration is a code integration strategy that initializes with one main artifact and proceeds down to lower level branches for implementation and integration. The head artifact of our top-down integration would be the initialization of the map and its direction services. Team members would use this head artifact to add additional features to the map. The stubs in the lower level are implementations of additional features such as the sidebar, markers, info panel and other user interface related elements. After a team member has implemented a new feature on a separate branch, we then merge it into one. When working on these branches, our sessions were usually split into groups of 2-3 and we did not have a lot of merges since most of the code being pushed was worked together on Visual Studio Code's LiveShare. As the stubs traverse down to the lower levels, the artifacts become more operational, such as adding current location to the map or adding a building. This makes it easy to add and remove any functional features that is map related.

We chose this top-down integration strategy because it's complementary to the structure of our codebase. This is because we were using the map and its direction service as the foundation and then additional implementations become new branches. We were also able to have the core functionality of our project at the early stage of development. This integration structure helped us to see flaws early in development as well as having a centralized system for interchanging features.

Defect Tracking List:

Date Reported:		Brief:	Date Fixed:	Who Fixe	How it was fixed:
10/27		white line	10/30/2020	Daniel	Setting map to absolute, top: 0, left: 0 in styling
10/27		+ ", Lawrence Kansas"	10/29/2020	Daniel	Added option to format data
10/29		Not trimming input in search box			
10/29		accepting empty value in search box	10/30/2020	Daniel	If statements
10/29		add GPS to route not working	10/29/2020	Group	Removed a console log of undeclared variable.
10/29		add additional buildings for firebase cloud			
11/1		Autocomplete adds an extra +", Lawrence"			

Figure 1: Defect List

Explanation of estimation of person-hours:

We used the Agile story point method of calculating person hours. The previous project's story points and hours are as follows:

Project 1: 8 story points, about 42 hours

Project 2: 5 story points, about 30 hours

Project 3: 8 story points, 53.33 hours

From this data, We believe that project 4 will take about 60 hours. Project 4 is an 8 story point project just like projects 1 and 3. However, I believe that project 4 is a little bit more complicated than the others. It is still on the same order of story points, but will likely take slightly longer.

So, our final estimate is 60 hours.

Accounting of person-hours:

person-hours

DATA	Daniel	Trieuanh	Maokun	Saher	Rahul			
10/27	20	20	20	250	0	Meeting		
10/28	150	30		420				
10/29		90	90		90			
10/30	290		120	20	70			
10/31		40	95	40	95			
11/1	30	30		65				
11/2		30		40	60			
11/3			30		120	30	Meeting 1	27-Oct
11/4	45					40	Meeting 2	29-Oct
11/5						20	Meeting 3	1-Nov
11/6		60				80	Meeting 4	3-Nov
11/7						20	Meeting 5	5-Nov
11/8						20	Meeting 6	8-Nov
						0	Meeting 7	
						0	Meeting 8	
						0	Meeting 9	
						0	Meeting 10	
Total Minutes	745	510	565	1045	645			
Total Hours	12.41666667	8.5	9.416666667	17.41666667	10.75			
Total Days	0.5173611111	0.3541666667	0.3923611111	0.7256944444	0.4479166667			
Total Years	0.001417427702	0.0009703196	0.00107496194	0.0019882039	0.00122716895			
Total Lives(~80 years)	0.000017717846	0.000012128	0.00001343702	0.0000248525	0.00001533961			
UNIVERSAL AGES	0	0	0	0	0			
Total Hours for the Team		58.5						

Figure 2: Accounting of hours