Final Master's Project Report
Sean Carey, Prateek Kakirwar, Daniel Chiang
Advisor: Erik Wilde
May 6, 2011

iBear: A Mobile Service for UC Berkeley

Abstract

UC Berkeley students face challenges in managing their online experience. They must navigate through a variety of loosely related websites to learn about class schedules, sign up for classes, manage financial information, buy textbooks, etc.

IBear is the name of a proposed suite of mobile applications for UC Berkeley that will enhance the student experience by providing a single interface to access many of the student information systems currently siloed throughout the campus. These mobile applications will capitalize on a set of new web services being developed by UC Berkeley in order to make information available to developers.

For our final project, we designed the first web application in the iBear suite[1], one that enables students to find classes listed in the Schedule of Classes and generate their class schedules. This application (referred to as iBear in this report) utilizes the new RESTful APIs for accessing course information. Through user research and an iterative design process, we optimized iBear to fit the needs of students, many of whom now own smartphones and regularly use mobile applications.

Problem Space

UC Berkeley's vast data assets and information are difficult to access for students and developers alike. We identified the problem of information access to be largely two-pronged:

Problem: Information is siloed and "trapped" in various databases and applications across campus.
Solution: Create easy-to-use web services and APIs (Application Program Interfaces) to "free" the data to allow developers to access campus information.

Problem: Websites and applications are out of date, full of legacy code, and have not adapted to the technologies and context with which students access information nowadays.
Solution: Build mobile applications to replace or supplement aging applications.

[1]The application can be found at http://groups.ischool.berkeley.edu/iBear/

Information Silos

The information used by students to learn about class schedules, sign up for classes, manage financial information, manage class information, etc., is siloed throughout the campus. Currently, there is no easy way to access all this information outside of the services provided by the campus (Telebears, Bearfacts, Schedule of Courses, etc.).

One initiative on campus, the myBerkeley project, describes the frustrating experience that students have of these information silos:

> Students must currently navigate dozens of department and college websites to search for information and to manage basic student functions. A disjointed online student experience is one of the biggest frustrations that UC Berkeley's technology-savvy students complain about, including:
>
> - Trying to manage **reminders and important tasks** that they learn about from a variety of sources including email messages, website alerts, paper syllabi, etc.;
> - **Academic planning** via TeleBears, BearFacts, Schedule, and DARS, etc.;
> - **Online bill paying** for CARS, Library, Cal-1 Card, etc.;
> - **Finding things to do** outside the classroom, and more.
> - Getting **answers to questions** / communicating with departments and program offices[2]

For various historical reasons, many of these applications have been built on completely different systems. For example, data behind Telebears and the Schedule of Courses are housed in different databases, with the Schedule of Courses only being updated once a day to reflect changes in class registrations that are stored in Telebears (Figure 1).

[2]http://campuslife.berkeley.edu/myberkeleyproject_overview

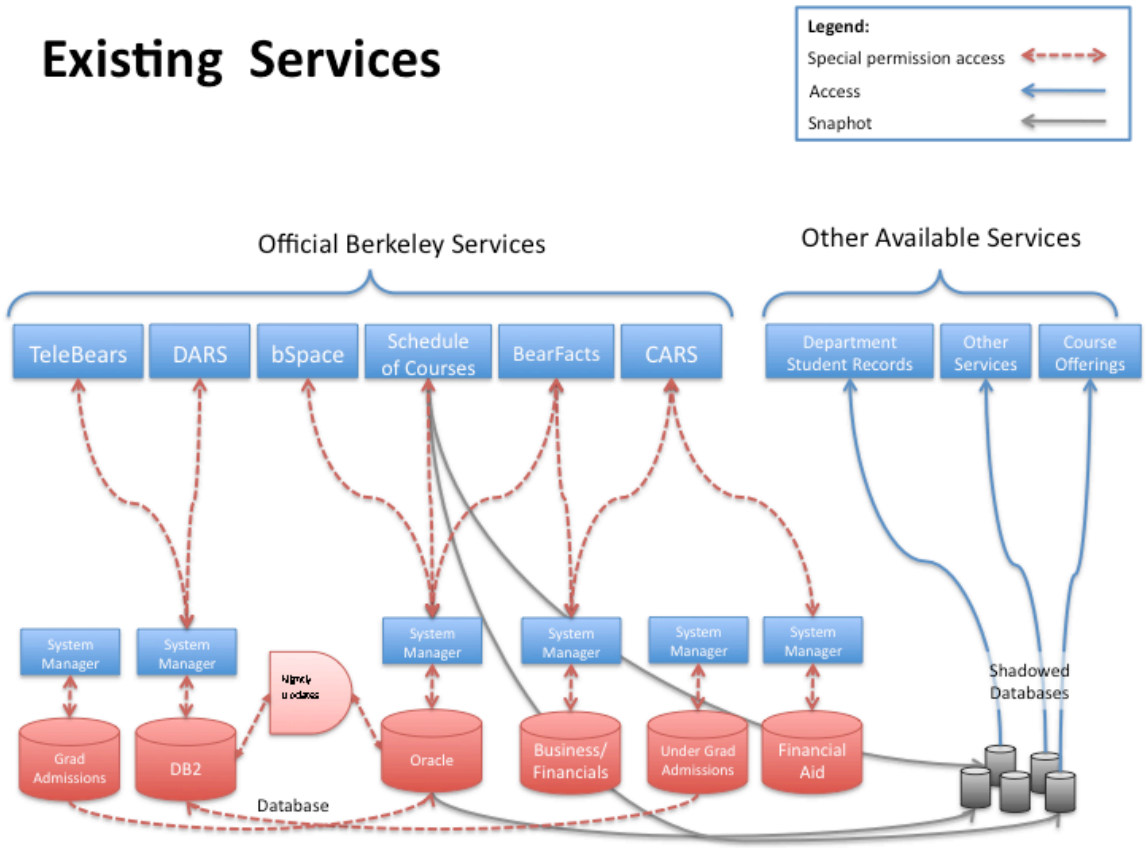

Figure 1. Existing campus services are siloed in different databases and systems.

The Information Services and Technology (IST) group on campus is currently undertaking an effort to "free the data" by providing RESTful (Representational State Transfer) APIs to access campus data. These APIs will ultimately allow developers to create mashup applications that use data from many disparate campus sources, which will create a more integrated and unified online experience for students.

Lack of Mobile Access to Information

The use of mobile devices is ubiquitous. Students use their smartphones to browse the web for information, check email, manage their social network, etc. through a proliferation of mobile applications. Given the widespread use of mobile devices for accessing everyday information, it is notable that there have been almost no mobile applications built to access campus services and meet the needs of U.C. Berkeley students. Aging and outdated campus web applications have not been adapted to the mobile context. This further exacerbates the problem of information access, as students are not able to perform student functions with the devices they use so frequently. Having mobile applications that access information about classes, campus activities, and students' academic progress would be extremely useful.

Other large universities have already demonstrated the usefulness of feature-rich mobile applications. Stanford University (http://mobile.stanford.edu/) and the University of Washington

(http://www.washington.edu/mobile/) provide two examples. UC Berkeley, as a leader in technological innovation, needs to refresh its aging information system to reflect both current technology and the needs of students who are much more tech-savvy than their predecessors.

Project Goal

We embarked on this project largely because we had firsthand experience with the problems surrounding information access on campus. The three members of our project team are graduate students at U.C. Berkeley's School of Information, and two of us were also U.C. Berkeley undergraduates.

Our goal was to use the newly developed class and course APIs[3] to develop a prototype web application that made searching for courses very easy for students. Being the very first developers to actually build something using the APIs, we wanted to evaluate these APIs and provide feedback to IST regarding their design and ease of use. We also wanted to test our new mobile application with a wide range of students to show how useful the application (and other similar applications) could be. Through an iterative, user-centered design process, we hoped to create an effective application that would be attractive to campus administrators and spur IST to accelerate their development of more service APIs.

New Campus Service APIs

As part of the campus-wide Operational Excellence Initiative, U.C. Berkeley is working to open up campus data to provide access to developers. Currently, a number of popular web applications such as Ninja Courses (http://ninjacourses.com) access campus data through programmatic screen-scraping because UC Berkeley allows no other access method. During our research for this project, a member of IST told us that at one point, there were so many hits to the Online Schedule of Classes' "List All" page[4] that the server was crashing, and thus server administrators started to cache the "List All" page. This clearly demonstrates the demand for access to campus data by developers and students using the applications create by those developers.

Creating APIs has been suggested for years, but research and development only began to make significant progress this year. IST's first project was to create prototype APIs for class, course, and registration data.

We built iBear around the course search API, recreating much of the search functionality of the Online Schedule of Classes. The API's query service was limited to just searching by department and course number, but our interviews with students showed that these two parameters were the ones most frequently used to search for classes.

---

[3]https://wikihub.berkeley.edu/display/sbint/Catalog+of+Service+APIs
[4]http://osoc.berkeley.edu/OSOC/osoc?p_term=SP&p_list_all=Y

Using the API, we were able to build a prototype application enabling students to browse for courses by department, search for courses by department or course number, view detailed course information, and see their class schedule on a simple calendar.

Application of Web Services to a Mobile Context

As mentioned above, part of the problem with campus information access is that existing websites and applications are out-of-date and have not adapted to the way students access information nowadays. We addressed this by building a prototype mobile application.

We wanted to design iBear to be a model application that could demonstrate to developers and campus administrators the usefulness of adapting existing web services to mobile devices. We hoped that iBear would encourage other campus developers to build mobile applications using the new service APIs. Knowing that the usability of our application would be vital for adoption—no one would take seriously something just as un-usable as some of the existing campus websites—we adhered to a user-centered design process to ensure our application met real student needs in both user interface design and features.

Technologies Considered and Used

Initially, we had to make a number of important decisions regarding technology. We had to decide what type of application we would build—a native, web, or hybrid application. Because of the diversity of web-enabled mobile devices that students use, we decided to build a web application to work on the most number of platforms. Given our limited time frame to produce a working prototype, we felt the best strategy was to write code once and have it function with many devices, including desktops. Our project timeline was also dependent on the prototype APIs that IST created.

To create the best prototype in the given time frame, we chose to use an existing mobile framework. Creating our own framework would have afforded us more flexibility to develop custom features, but would have ultimately required that we write specific code for every platform we wanted to support. There were a variety of popular mobile frameworks. After reviewing several such as Sencha Touch, we decided on JQueryMobile for its relative simplicity and usage of jQuery, which we were well-versed in.

In hindsight, using jQueryMobile had both advantages and disadvantages. It saved us time from not having to develop mobile-specific styles and features like page transitions. However, given that jQueryMobile was still an alpha release, it created problems with its buggy-ness and quirks. Futher, as a web application using jQueryMobile, iBear did not perform as well as we had hoped on mobile devices.

In order to make iBear function more like a native application, we considered using PhoneGap. PhoneGap wraps web applications in native code, providing them access to native features on mobile devices. Using PhoneGap would have allowed iBear to be downloaded through the

official app stores and optimize it for different mobile platforms. We thought this might give iBear increased legitimacy with consumers. But after testing PhoneGap, we determined that it did not make any difference in iBear's performance. We also determined that if iBear was well built, we could pitch it to UC Berkeley easily enough without the need to go through official app stores. The course search API prototype did not support JSON (JavaScript Object Notation) with padding (JSONP) as a response option. This made interacting with the API with JavaScript difficult due to browser protection against cross-site scripting. As a workaround, we wrote PHP scripts to retrieve the API data and then inject the response into the front-end code. As a result, every page in iBear is built dynamically in response to what courses the user searches for.

For managing iBear's backend database and handling administrative tasks like maintaining logs, we used Django with a MySQL database. Django looked very promising. It followed the Model-View-Controller architecture, which would have enabled us to separate the act of defining and accessing data (the model) from the business logic (the controller), which in turn would allow us to separate the code from the user interface (the view). Hence, we could simultaneously work on different parts of the project. However, in the end, we had trouble integrating Django with the other parts of iBear (jQueryMobile and PHP). Our current iBear prototype does not include a database, but can easily be integrated with one. Without a database, we are storing persistent information with cookies.

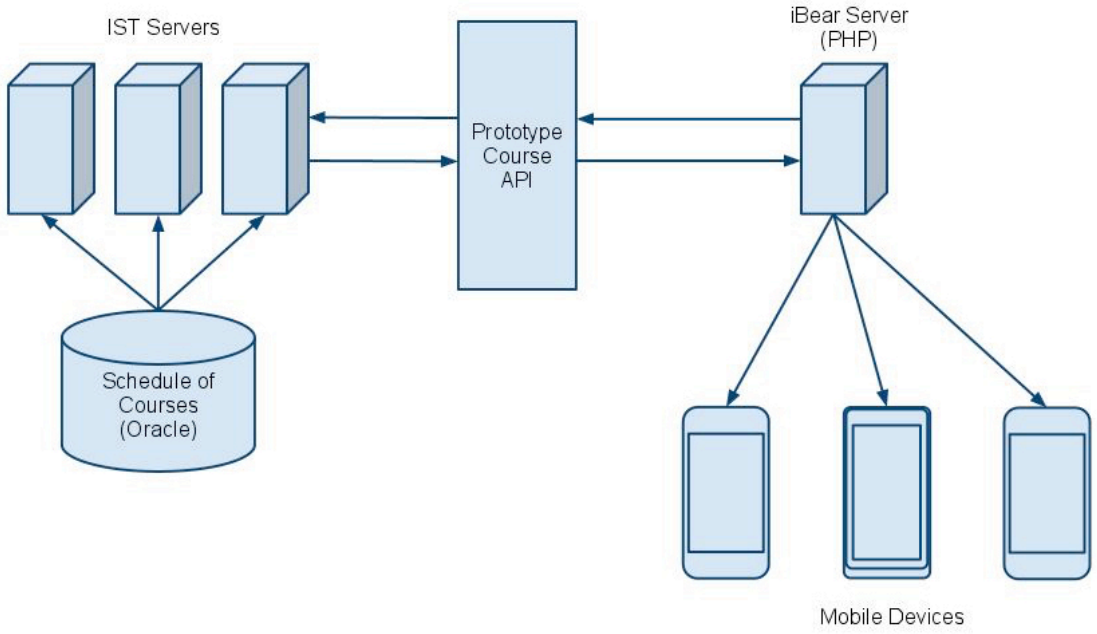

Figure 2. IBear's web architecture.

User-Centered Design Process

Competitive Analysis

At the beginning of our project, we looked at an existing mobile application developed by a contractor for UC Berkeley. While being a great start for an official mobile application, it did not provide essential student features. In fact, one commenter on the application's page in iTunes wrote, "Here is a suggestion: please include Tele-Bears and some other school systems in it!"

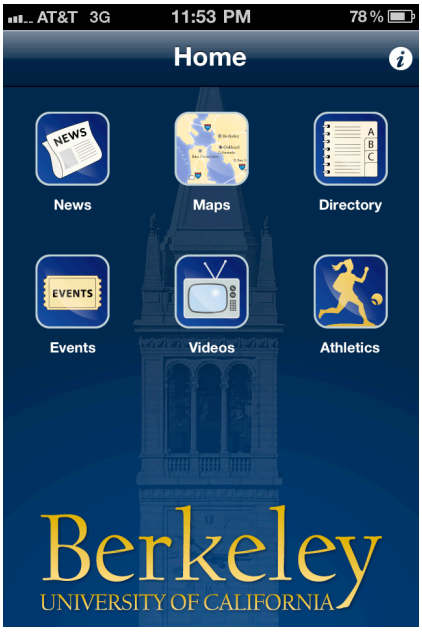

Figure 3. A mobile application for UC Berkeley that does not include course information.

Many other leading universities also have an official mobile application. We studied the applications of such schools as Stanford, MIT, University of Oregon, University of Washington, Virginia Tech, UCLA, West Virginia University, and Carnegie-Mellon University. Specifically, we examined whether or not these applications included functionality to search for courses. For those that did enable course search, we noted how courses were displayed, how browsing and/or searching for courses worked, whether or not courses could be bookmarked, etc.

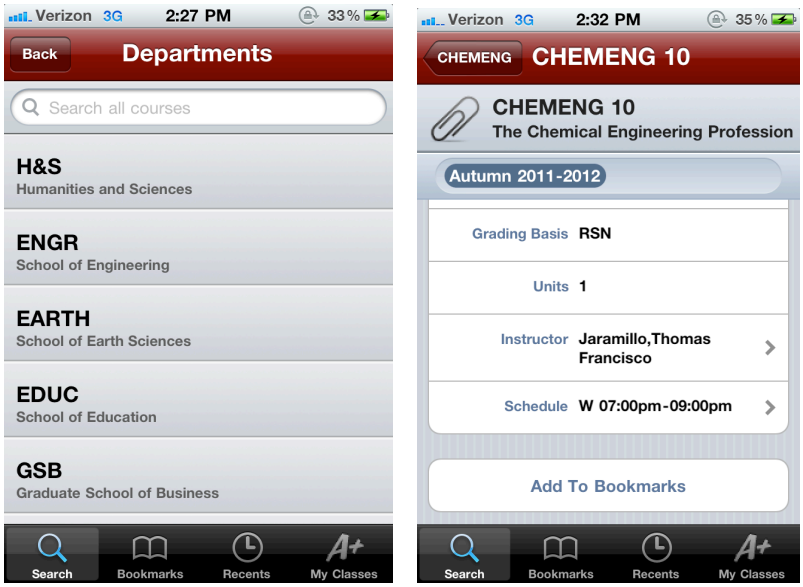

Figure 4. Stanford's mobile course search application.

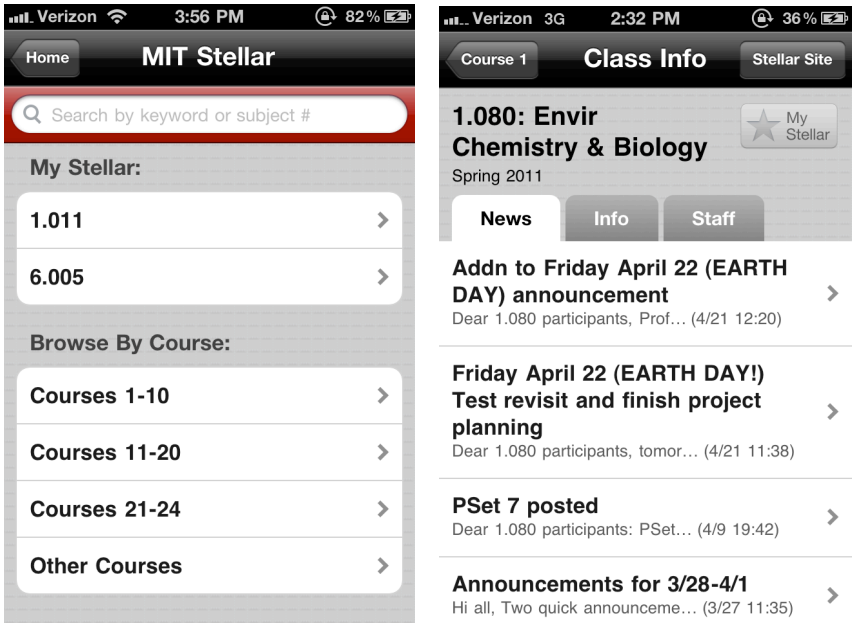

Figure 5. MIT's course search application.

Paper Prototype

We first developed a paper prototype that enabled us to quickly lay out iBear's user interface and try different layouts. We wanted it to make very simple for a student to either browse to or search for courses.

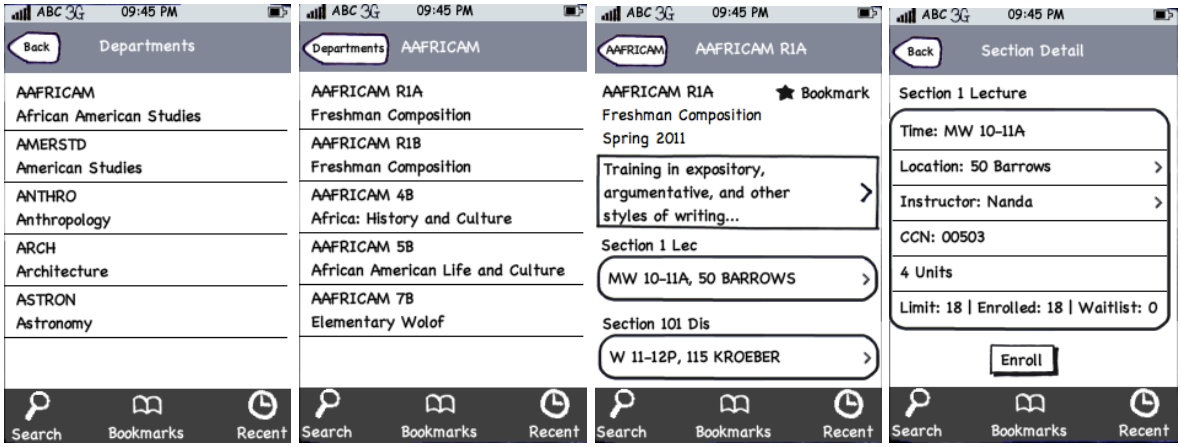

Figure 6. User interface flow for browsing to African American Studies R1A's Section 1 Lecture.

Since iBear was meant to show how one application could bring together many different sources of campus data, we designed individual courses to link to campus map data, textbook information, and instructor information. If future APIs opened up these sources of data, many such useful mashup applications could be created.

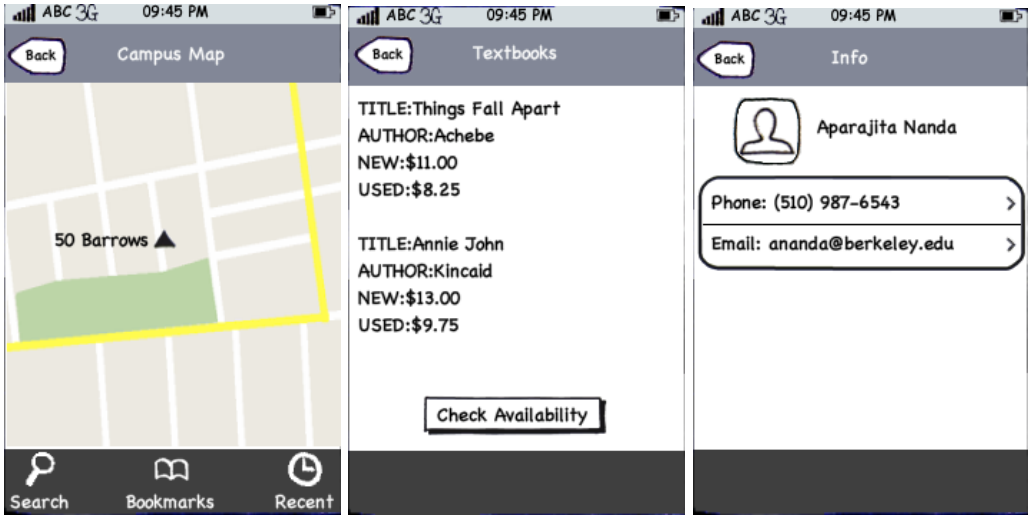

Figure 7. A course would link to its building on campus, required textbooks, and directory information for its instructors and teaching assistants.

Since search is one of the main functions in the Online Schedule of Classes, we also envisioned a robust search interface for our application. We left out search fields found in the Online Schedule of Classes that we didn't think would be useful for students.

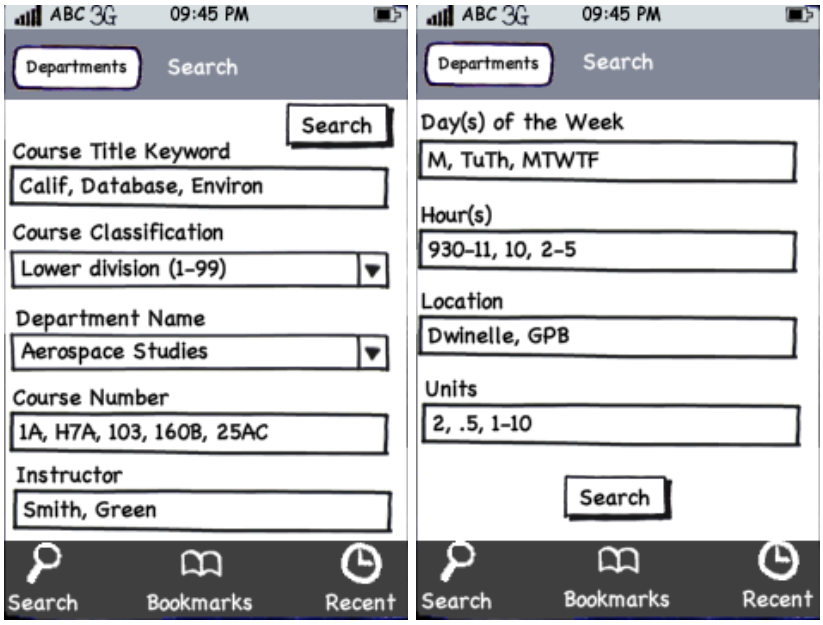

Figure 8. Search interface for the paper prototype.

We also designed an alternate tabbed interface for displaying individual courses. We wanted to test both the tabbed and non-tabbed prototype screens with students to see which they would prefer.

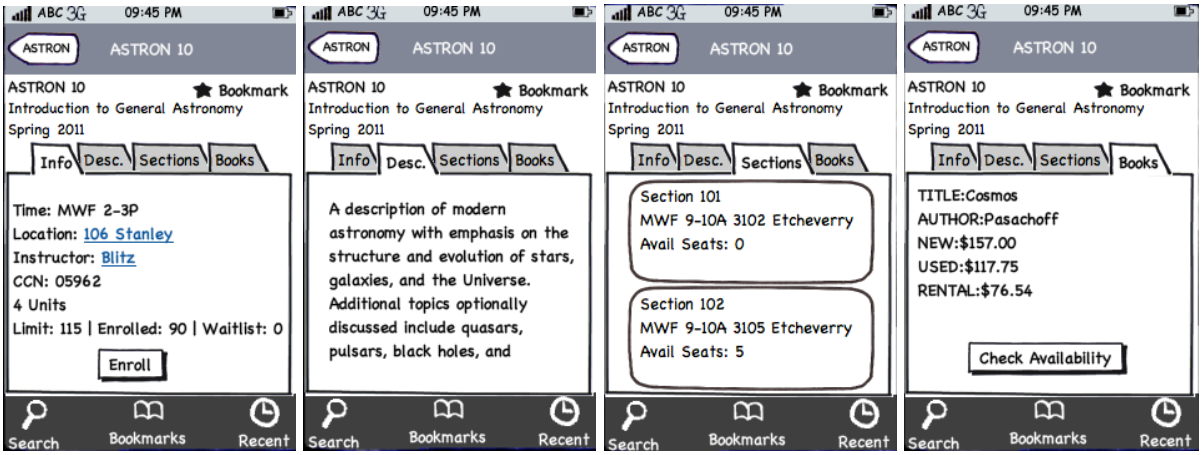

Figure 9. Tabbed interface for Astronomy 10.

User Testing the Paper Prototype

We tested our paper prototype with one graduate and four undergraduate students in different departments. Our primary goal was to see and hear what users thought about our interface and evaluate whether they could navigate it easily to complete common tasks.

We had the users perform several tasks, including navigating through the course directory, searching for a particular class, finding a particular discussion section of the class, registering for the course, looking up textbook information, and contacting the class' instructor. This was to determine if our designed interface would work well for students.

The tests were very useful in informing us about new features that students wanted, along with fixes that desired for the current design. Students mentioned a host of desired features, including auto-completion for search fields, a calendar feature, professor ratings, and additional buttons to enroll in sections. To our surprise, most users preferred browsing for classes instead of searching for them, as they did not want to type anything on a mobile device if they could avoid it. Overall, students said they wanted a clean interface with just the necessary information on a given screen. Presented with both the tabbed and non-tabbed interface for individual courses, students clearly favored the tabs.

The paper prototype was, of course, limited. It could not completely simulate the mobile interfaces that users were accustomed to. For example, users often did not know they could scroll the page to see more information. This functionality was difficult to simulate on the paper prototype.

Interactive Prototype

The next step was to build the application based on our user research. Given the limited time this semester and the fact that the service APIs we used were still in development, we knew that our application would be a work-in-progress prototype. However, we wanted this prototype to be as close to a fully functional web application as possible.

We integrated our prototype with the campus CalNet authentication system to show that our application could easily fit into the existing information landscape on campus. Allowing a student to log in would also enable the application to associate bookmarked classes and a course schedule saved in a database to that student.

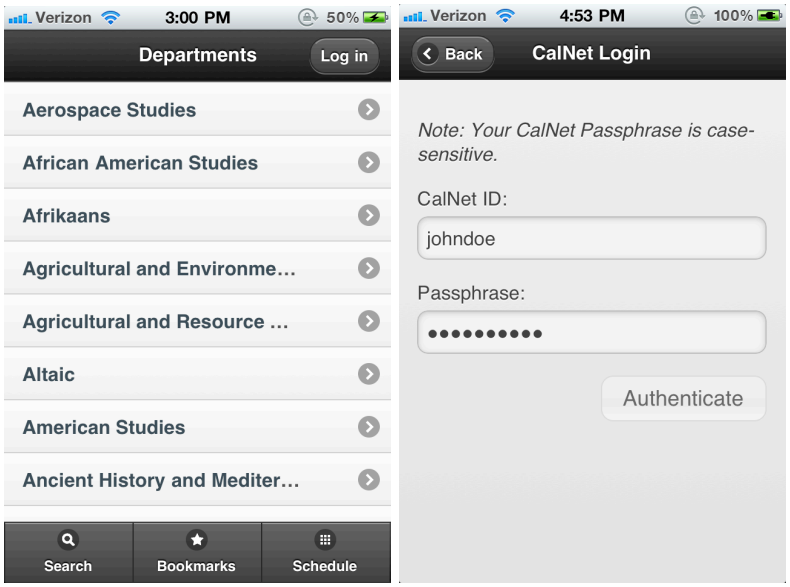

Figure 10. Homepage and CalNet authentication page in the interactive prototype.

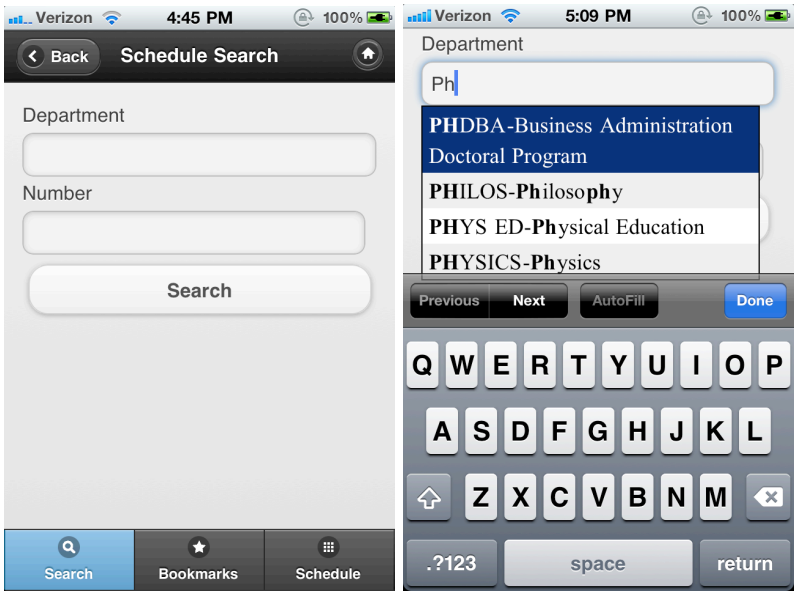

Figure 11. Course search and autocomplete function in the interactive prototype.

Based on our user research, we built a tabbed interface for displaying individual course pages. The new APIs that we used could return general information about courses as well as detailed information about specific Spring 2011 courses. However, since there were not yet any APIs that returned textbook information or directory information about the campus community, we had to use screen scraping to obtain this data. Thus, the “Books” tab in each course returns scraped data from the Online Schedule of Classes about which required textbooks are located in the campus student store. And clicking on an instructor’s name will search for that instructor in the CalNet Directory (https://calnet.berkeley.edu/directory). Because the course API only returns the instructor’s last name and first initial, searching for the instructor on the CalNet

Directory sometimes returns many results because the search cannot disambiguate between faculty with the same last name and first initial.

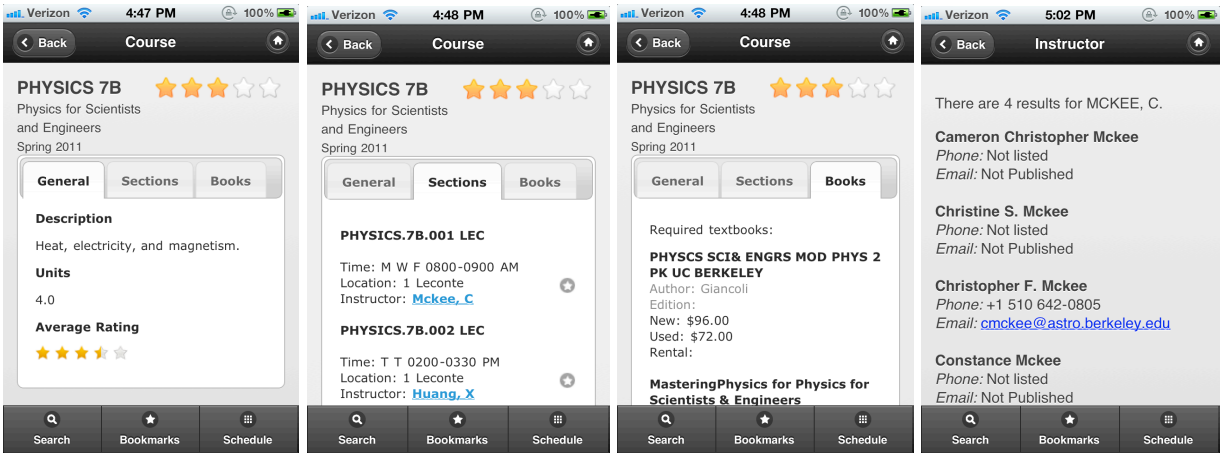

Figure 12. Detailed course information for Physics 7B in a tabbed interface.

IBear allows a student to bookmark courses of interest and generate a schedule based on them. Many students had told us that being able to generate sample schedules and seeing possible class time conflicts was very important to them in any course search application.

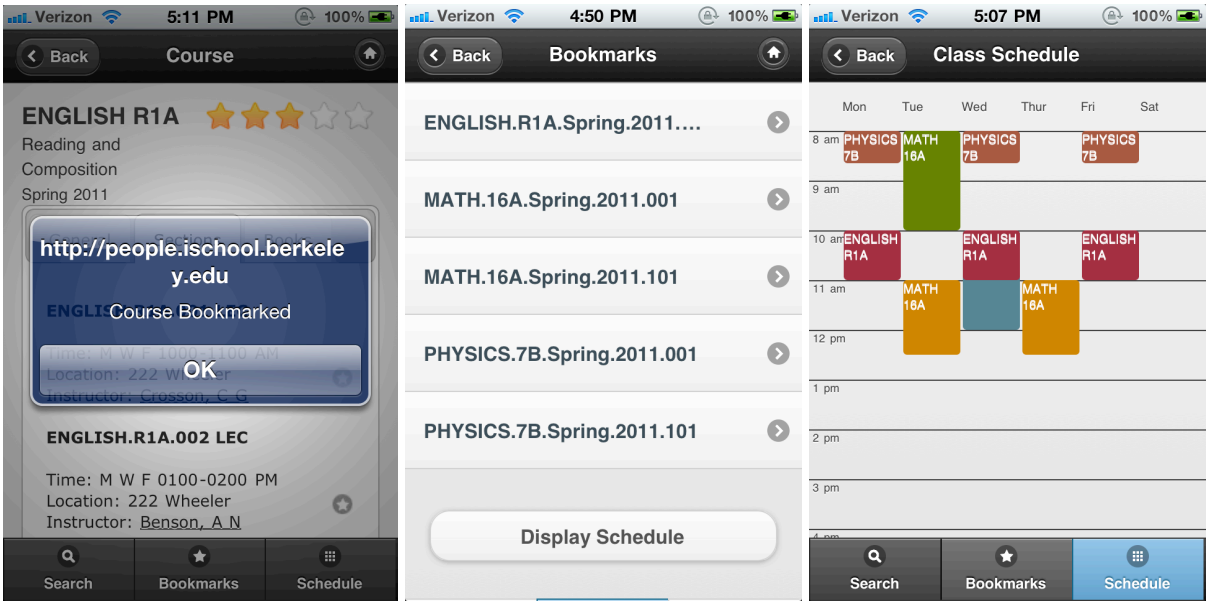

Figure 13. Bookmarking a course, viewing all bookmarks, and displaying a class schedule.

User Testing the Interactive Prototype

To evaluate the interactive prototype, we tested eight undergraduates, only one of whom had also tested our paper prototype. Overall, users were very positive about our program, often stating that it was much easier to use than the Online Schedule of Classes.

Users initially had difficulty with the bookmark feature as the association between a course section and its bookmark button was not clear in our design. Yet once they understood how it worked, they had no trouble with it later. Users were also confused by the wording on the course information page. UC Berkeley considers lectures, discussions, labs, and seminars all as "sections" of a given course. This confused users because they thought "section" only referred to the auxiliary discussion or laboratory part of course and did not think it would include lectures.

Our research confirmed our assumption that students were interested in having campus web services adapted to and optimized for mobile devices. Several test participants mentioned how impressed they were with the interface, knowing that mobile applications could easily be cluttered or clumsy to use.

Evaluation of Course API

We must note that the API powering iBear was a prototype and therefore not as refined as other APIs out on the Web. The API was also constrained by the database it drew upon, resulting occasionally in unpredictable behavior.

Problems Encountered

We ran into several glitches when utilizing IST's course API. These included issues with returning lecture and section data, the lack of JSONP support, and somewhat cryptic responses for certain fields. We found ourselves having to reverse-engineer certain aspects of the APIs in order to make them work with iBear.

Missing Data

The API simply did not return some data such as the Course Control Number (CCN) and the numbers of students enrolled and waitlisted for a particular class. This data would be crucial if the API were to be used in production because the TeleBears course registration system relies on the CCN to uniquely identify courses and enable students to register for them.

Another crucial omission concerned how the API handled cancelled courses and sections. For cancelled courses, the API did not return empty rows for section meeting days and times, but rather omitted the relevant tags all together. This required us to create PHP scripts to check that the appropriate XML tags even existed before returning the response. Otherwise, the missing data for cancelled courses would break our code.

Here is an example from the API's XML response for comparison. This is a snippet of a regular response for a class (Info 203). Note the "<sectionMeetings>" element.

```
<sections>
      <seatLimit>0</seatLimit>
      <seatsAvailable>0</seatsAvailable>
      <sectionFormat>LEC</sectionFormat>
      <sectionId>INFO.203.001</sectionId>
      <sectionMeetings>
            <building>SOUTH HALL</building>
            <endTime>0200 PM</endTime>
            <instructorNames>CHESHIRE, C</instructorNames>
            <meetingDay>  T T</meetingDay>
            <room>0202</room>
            <startTime>1230</startTime>
      </sectionMeetings>
      <sectionNumber>001</sectionNumber>
      <sectionType>PRIMARY</sectionType>
      <studentsEnrolled>0</studentsEnrolled>
      <uri>
            http://esb-
            d1.calnet.berkeley.edu:8181/cxf/asws/classoffering/INFO.203
            .Spring.2011/section/001
      </uri>
      <waitlistSize>0</waitlistSize>
</sections>
```

This is a snippet from the response for a cancelled class (Info 152). Note the missing "<sectionMeetings>" element.

```
<sections>
      <seatLimit>0</seatLimit>
      <seatsAvailable>0</seatsAvailable>
      <sectionFormat>LEC</sectionFormat>
      <sectionId>INFO.152.001</sectionId>
      <sectionNumber>001</sectionNumber>
      <sectionType>PRIMARY</sectionType>
      <studentsEnrolled>0</studentsEnrolled>
      <uri>
            http://esb-
            d1.calnet.berkeley.edu:8181/cxf/asws/classoffering/INFO.152
            .Spring.2011/section/001
      </uri>
      <waitlistSize>0</waitlistSize>
</sections>
```

Human readers of the XML response would gloss over the omission of the “<sectionMeetings>” element. But if developers did not know the API functioned this way and planned for it, their code would break.

As mentioned above, the API’s representation of instructor names (as last name, first initial, and sometimes middle initial) made it difficult to link them to unique values in the campus CalNet directory. We later learned that this was how instructor names were stored in the underlying database.

The Online Schedule of Classes provides rich information such as course description and course prerequisites. The free text search API returned this information, but the information was only available if we made an additional query in addition to the structured course search API. We had hoped to retrieve all the rich information for a given class when using the structured course search API.

Inconsistent Representations of Information

A given physics class has both a laboratory and discussion section. However, given a query for Physics 7A with the course API, the response represents the sections as two different meetings, in two different locations, of the same discussion section. Inconsistent representations of course sections like this were difficult to detect until they broke our PHP scripts.

```
<sections>
      <seatLimit>0</seatLimit>
      <seatsAvailable>0</seatsAvailable>
      <sectionFormat>DIS</sectionFormat>
      <sectionId>PHYSICS.7A.101</sectionId>
      <sectionMeetings>
            <building>LECONTE</building>
            <endTime>0200 PM</endTime>
            <instructorNames>TRAN, L</instructorNames>
            <meetingDay> M</meetingDay>
            <room>0225</room>
            <startTime>1200</startTime>
      </sectionMeetings>
      <sectionMeetings>
            <building>LECONTE</building>
            <endTime>0200 PM</endTime>
            <meetingDay>   W</meetingDay>
            <room>0224</room>
            <startTime>1200</startTime>
      </sectionMeetings>
```

Redundant Data

The API returned a variety of data, including classUID, courseUID, sectionUID and sectionID. These elements often contained overlapping data. For example, CourseUID contained the courseNumber and departmentCode. Although not a big issue, the redundant data was sometimes confusing when we had to parse the response data.

```
<ClassOffering>
      <classUID>ARCH.100A.Spring.2011</classUID>
      <courseNumber>100A</courseNumber>
      <courseTitle>Fundamentals of Architectural Design</courseTitle>
      <courseUID>ARCH.100A</courseUID>
      <departmentCode>ARCH</departmentCode>
      <lowerUnits>6.0</lowerUnits>
<sectionUID>ARCH.100A.Spring.2011.001</sectionUID>
```

Unexpected Representations of Data or Opaque Data

We found in our use of the class API that not all classes had a room number. This is because of certain buildings where the main lecture hall is referred to without a number (such as Wheeler Hall Auditorium). This was an important edge case, because our PHP script would break when querying for a course located in such a room.

The search results returned by the API were sorted by the first digit of the course number instead of the commonsense method used by the Online Schedule of Courses. For example, a number of classes listed in a search result with the following course numbers would be sorted like this: “1”,”10”, “133”, “24”, “32”, “7”, “98”. In contrast, the Online Schedule of Classes that students are accustomed to would order the courses as: “1”, “7”, “10”, “24”, “32”, “98”, “133”. We had to create a workaround in our code that would reorder the incoming results correctly. We alerted the API developers and they plan to change the sorting algorithm to reduce the work developers must do when using the APIs.

The way that that the API represented academic terms was opaque. In the response, “B”, “C” and “D” referred to the Summer, Fall, and Spring semesters, respectively.

The API had a unique way of encoding some data, partly as a result of the structure of the underlying database. This required some code to deconstruct the response for added functionality. For example, a class that meets on Monday and Wednesday would return:

```
      <meetingDay> M W</meetingDay>
```

A class that meets on Tuesday and Thursday class would return:

```
      <meetingDay>  T T</meetingDay>
```

And a class that only meets on Friday would return:

```
<meetingDay>      F</meetingDay>
```

Note the white space used to delineate the locations of the days of the week. It was easily human-readable, but difficult for our code to interpret. This issue was flagged early by the API designers and we have a suggestion as to how the days of the week could be better represented.

Incomplete Documentation

When we started using the APIs, we had a difficult time interpreting the XML responses. Not knowing what some of the XML tags referred to, we would search for different courses in the Online Schedule of Classes to see what fields and details were displayed. Then we would make an educated guess about what a particular XML tag in the API response might refer to.

Overall, we did a lot of reverse engineering and trial-and-error to determine how to use the API properly and how to interpret the returned responses. Fuller documentation about the APIs would have helped us to get oriented about their usage. Each of the prototype APIs included sample HTTP calls and sample XML and JSON outputs. These examples certainly helped us to get up and running with the APIs quickly. But it would have also been very helpful if the documentation provided information about common errors or exceptions and how handle them. Towards the end of coding the prototype, we discovered that each API included a WADL file[5] but we didn't know how to use these files.

Recommendations

We hope that IST will continue to create and develop APIs for access to campus data. These will allow us and future developers to improve the web services for UC Berkeley students, faculty, and staff. We have already relayed many of the above issues to IST, but we also have some additional recommendations.

First of all, we hope that IST can maintain the role of overseeing future development of campus APIs rather than allow different campus entities to develop different APIs. It is vital that one group oversees development of APIs to ensure they are similarly structured and able to integrate with each other. Otherwise, the APIs themselves could become siloed and incompatible. Going forward, the campus also needs to determine who will be supporting the APIs and how API updates will be communicated with developers.

One recommendation is the ability to query for classes based on time, day, professor name, and by whether or not the class fulfills a breadth requirement. Our user testing found that these were

[5]For example, http://esb-d1.calnet.berkeley.edu:8181/cxf/asws/classoffering?_wadl&_type=xml

the next most common search parameters after department and course number. Students were particularly interested in these additional search options because they wanted to take courses from professors they liked, find classes that fit into their schedules at particular times, or quickly find courses that fulfilled breadth requirements.

The documentation on the API wiki page was helpful when we started, but we would appreciate having more details about what elements we could expect in a typical API response as well as what errors might occur. The documentation should include a fuller list of known issues. It should also include some degree of explanation about what a WADL file is and how to use it.

The APIs need to provide real-time, up-to-date information. For example, course enrollment needs to be updated in real-time to show students which courses are currently open during enrollment. We realize this might require restructuring certain databases on campus or have APIs that can call upon the diverse databases on campus.

The API developers from IST who we have been talking to were interested in having some sort of public forum that developers could use to ask questions and give feedback on the APIs. We appreciate this idea and think that such a forum would greatly benefit the consumers of these APIs and spur developers to build new applications with them.

Going Forward

IST is currently pitching the prototype APIs to the university in hopes of securing funding to move the APIs into production and begin creating new ones. The next APIs that IST is planning to build will give access to student profiles and information about what classes students are enrolled in. The course API was recently updated to include course information from the Law School at Berkeley[6]. As their development progresses one step at a time, the prototype APIs hold much promise that many of the problems with information access at UC Berkeley will begin to be alleviated.

[6] https://wikihub.berkeley.edu/display/sbint/Update+Course+Information-Law+School