Developing Cross-platform Mobile and Web Apps

Xiang Mao [1] and Jiannong Xin *[2]
[1]Department of Electrical and Computer Engineering, University of Florida
[2]Institute of Food and Agricultural Sciences (IFAS), University of Florida
University of Florida PO Box 110495, Gainesville, FL 32611, USA
mx@ufl.edu, xin@ufl.edu

ABSTRACT

The rapid growth of mobile usage in the world has created many opportunities to leverage the ubiquity of mobile network. Mobile applications have gained momentum in many sectors including agriculture. However, development of mobile apps is still facing many challenges, including platform fragmentation and rapid changes of the mobile technology. It hence has a great benefit to identify common frameworks and solutions that allow developers to leverage a single codebase and deploy it on multiple platforms, both on different mobile devices and mobile web. This paper explores current mobile technologies and design options of mobile databases to develop cross-platform mobile apps. Discussions are made using the proposed framework for a mobile project named Southeast Landscape Pests developed at University of Florida/IFAS. The project leverages cross-platform technologies and JSON as an alternative for database storage. The solution is a cost effective approach to deploy apps to iOS, Android and Web and could be adopted for other small size mobile projects.

Keywords: Mobile, web, cross-platform, pest

1. INTRODUCTION

Nowadays, high-performing mobile devices are easy to find on everyone's hand. Pervasive high-speed data support from the network architecture and introduction of easy-to-use market-places for apps, e.g. Apple App Store and Google Play, enable people to realize their underlying needs for simple targeted apps and hence result in a highly increasing usage of mobile apps (Tracy, 2012). Research has shown a wide and growing adoption of mobile apps also happening in agricultural industry, especially in developing countries (Qiang et al., 2012; USAID, 2011).

Following this trend, companies offering traditional web services, including giants like Google and Facebook, have increased their effort on mobile application development to make the applications widely available and easy to use on mobile devices. Hence, a common framework will benefit development cycle and resource consumption and allows developers to leverage a single codebase and deploy it on multiple platforms, both mobile browsers and different mobile operating systems (OSs). This paper explores current mobile technologies and design options of mobile databases to determine the feasibility of such solution. Discussions are made about the proposed framework applying on a mobile project, Southeast Landscape Pests, developed by Office of Information Technology, University of Florida/IFAS.

2. MOBILE DEVELOPMENT

2.1 Web, Native vs. Hybrid Mobile App

Technology challenges to deliver cross-platform apps lie in the essential difference between the two types of applications: mobile web app and native app. Mobile web apps reside on server without installation on devices, and both debugging and fixing are possible in real-time execution. However, mobile web apps usually cannot access the mobile device's features such as camera and motion sensors.

Native apps, on the other hand, are developed for a specific type of mobile device and could take advantage of the device's functionalities including file storage. The main drawback of native apps is that they must be developed separately for each platform, hence increasing development time and cost. Also, for a native app, once it is modified, the developer needs to push all users to update their apps to receive upgraded services.

Hybrid app, taking the advantage of the two, is a native mobile app embedding web content inside a thin native container. Hybrid apps are installed through an app store, run on the device and provide access to enhanced native device hardware, but are written primarily using HTML, CSS and JavaScript.

2.2 Technology Challenges to Deliver Cross-platform Apps

Development of native apps for different mobile OSs requires the usage of different programming languages and architectures. Take the two dominating mobile OSs, Android and Apple iOS, as example: Android apps are developed using Java in Eclipse with the Android Developer Tools (ADT) plugin support, while iOS apps are usually developed in Xcode, which is tied to Mac OS, using Objective-C, Swift and Cocoa framework.

2.3 The Proposed Solution

To overcome the challenges mentioned above, we introduce a solution that builds on a single codebase and deploy cross-platforms, both mobile browsers (as a mobile web) and different mobile OSs (as a hybrid mobile app), without any change of configuration. The core concept involves Adobe's PhoneGap framework, which enables wrapping up of HTML, CSS and JavaScript code into different mobile packages for cross-platform deployment. Instead of using a database management system, the solution uses JavaScript Object Notation (JSON) for data storage and transport.

2.3.1 User Interface (UI)

With the help of PhoneGap Build service, development will only need to be focused on mobile web with consideration of app compatibility, but also the functionality is limited to the scope of HTML5, CSS3 and JavaScript for PhoneGap supported file types.

To build a mobile web looks and behaves like a native app, and can be automatically rendered fluidly to fit various resolutions of devices, it requires a lot of work if starting from scratch. To accelerate the development, our design is based on jQuery Mobile (JQM) framework, while other options include Bootstrap and Sencha.

The JQM framework provides many features such as an Ajax navigation system that brings animated page transitions and a core set of UI widgets. It also offers several custom events that build upon native events so that, for example, developers can have the page load and display dynamic content from JSON objects or database without breaking the layout.

2.3.2 Online vs. Offline Mode

For many apps, users only consume data stored in a database that is not frequently updated. In this case, the database (or data files) can be placed on the client side. That is, from the perspective of the native app, the app can always run in offline mode since no information is required from the server. The benefits are obvious that the data traffic of using the app is minimized and the system does not require any server-side configuration to support the app.

2.3.3 Data Storage

In order to deploy the mobile web as an app, we need a data storage solution that works on both the web and mobile devices. We tried to adopt client-side database (the difference is that website needs to download the database from the server first before operation while app is operating on the local database). However, a common database solution such as SQLite and IndexedDB has fragmented support from different browsers.

So we turned our eyes to no-database solutions such as Extensible Markup Language (XML) and JSON. Comparing the two formats, and using this project as an example, JSON is easier to represent complex data relationship among hosts, pests, and damage types and is more flexible in defining varying number of elements associated with an attribute. It also has a simpler specification in comparison with XML. As a result, the JSON file size is smaller than XML, which is a factor for app size and performance.

More importantly, JSON has the advantage in processing simplicity. It is essentially serialized JavaScript object that makes it fit better client side scripting, which is primarily done in JavaScript. The XML APIs in the browser are comparatively clumsy and the natural mapping from JavaScript objects to JSON eliminates the serialization issues that arise if the developer is careless with XML format.

Given JSON as the storage format, the idea is to construct object-based data representation and store the data object in JSON files (the detail of the JSON objects will be introduced in the next section). Upon using the data, the app loads the JSON files from the same relative file path via Ajax. We also use HTML5 local storage feature to store some “session” variables since JavaScript variable is lost when page changes, and it is more robust than passing the value of variables directly in Uniform Resource Locator (URL).

2.3.4 Deployment

As shown in Fig 1, we developed the mobile web in JQM framework using JSON as data storage, and then use PhoneGap Build Service to pack the same codebase into different packages for cross-platform deployment.

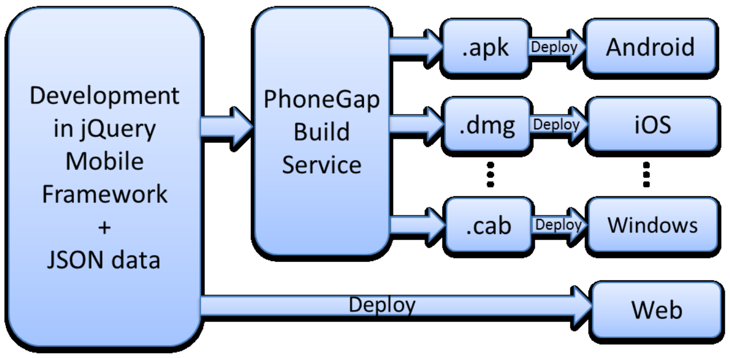


Figure 1. Deployment for mobile web and app

3. SYSTEM DESIGN

In this section, we are going to introduce a mobile web and app project, Southeast Landscape Pests, which applied the proposed solution.

3.1 Purpose of the Project

The Southeast Landscape Pests mobile project is intended to support preliminary diagnosis of commonly established arthropod plant pests through various combinations of user-initiated search options. Concise behavioral and biological information is provided for each pest species and high quality images of life stages, characteristic features and damage symptoms are provided to facilitate diagnosis by non-experts, along with hyperlinks for further information. It's currently focused on native, adventive and naturalized pest species in the southeastern United States with a database of close to 250 pest species and 300 plant hosts. Since many users will use this web/app as an initial source of information, it is an ideal opportunity to disseminate the knowledge of high risk invasive species.

3.2 Data Object Architecture

All pest information is stored in the *pests* object, which is an object array of *pest* objects. Each *pest* object contains a property of unique pest ID (p_id) that serves as a similar role of primary key in a database. Other properties, including detailed biological information, damage to agriculture, detection & management, references and associated images, can be found in Fig 2(a). We also extract some key information from the *pests* object and store them in a separated object named as *index* (Fig. 2(a)). The purpose of doing this will be discussed in the next section.

Hosts, slightly different from *pests* in storage, are grouped in categories (Fig. 2(d)). Each *host* object, besides ID and name, has a property of pest ID (p_id) array that indicates which pests are harmful to the host. Based on this knowledge, we can provide pest searching by host function.

Other display and search functions require pest information to be indexed by either pest type or damage symptoms. We in turn created two objects to store the *pest-type* relations and *pest-damage* relations (Fig. 2 (b) and (c)). Creating these two objects is not necessary but to improve runtime performance. We could also associate pest type and damage information with the *pests* object, which however would result in more sorting process. Considering the number of pests is much larger than it of pest types or damage symptoms, creating independent index will minimize the size of data file.

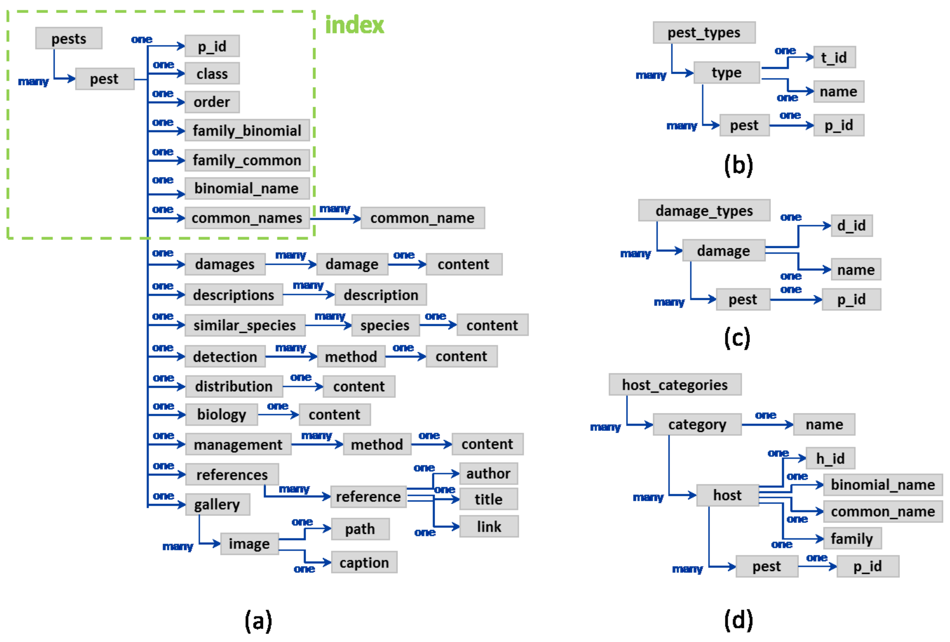


Figure 2. Pest data in JSON object representation: (a) pest, (b) pest-type, (c) pest-damage, (d) host.

3.3 User Interface

The user interface is designed using JQM framework. There are five main pages: *Home, Pest List, Search, About* and *Pest Detail*. Every page shares the same Ajax navigation header for quick switching as well as the footer. History tracking and back button is also enabled on every page so that the app version can have the same “go back” function.

3.3.1 Home

The *Home* page is designed for a relative short introduction of the project and the sponsor groups (logos at the bottom) compared to the *About* page. In the remaining space, we put several highlighted pests in a slideshow and allow users to access the corresponding pest detail page by tapping on the image. A script for the home page is implemented to automatically adjust the display size according to the browser/device’s resolution as demonstrated in Fig. 3.

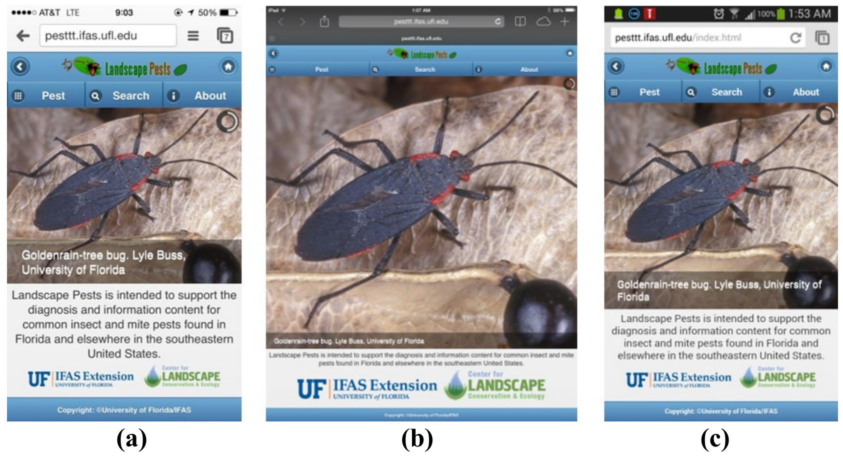

pesttt.ifas.ufl.edu
Landscape Pests
Pest
Search
About
Goldenrain-tree bug. Lyle Buss, University of Florida
Landscape Pests is intended to support the diagnosis and information content for common insect and mite pests found in Florida and elsewhere in the southeastern United States.
UF IFAS Extension
LANDSCAPE
Copyright: ©University of Florida/IFAS
(a)
(b)
(c)

Figure 3. Dynamic display of home page on different device resolutions: (a) iPhone 5, (b) iPad mini, (c) Samsung Galaxy S3 (Android System).

3.3.2 Pest List

Pest List page can be accessed by tapping the “Pest” button on the navigation bar. It provides a categorized view of different pests in a two-level menu. Tapping on a pest tab on the second-level menu will redirect the user to the corresponding *Pest Detail* page, as illustrated in Fig. 4. JSON objects of *pest_types* and *index* mentioned above are loaded via Ajax before the page is rendered in JQM CSS.

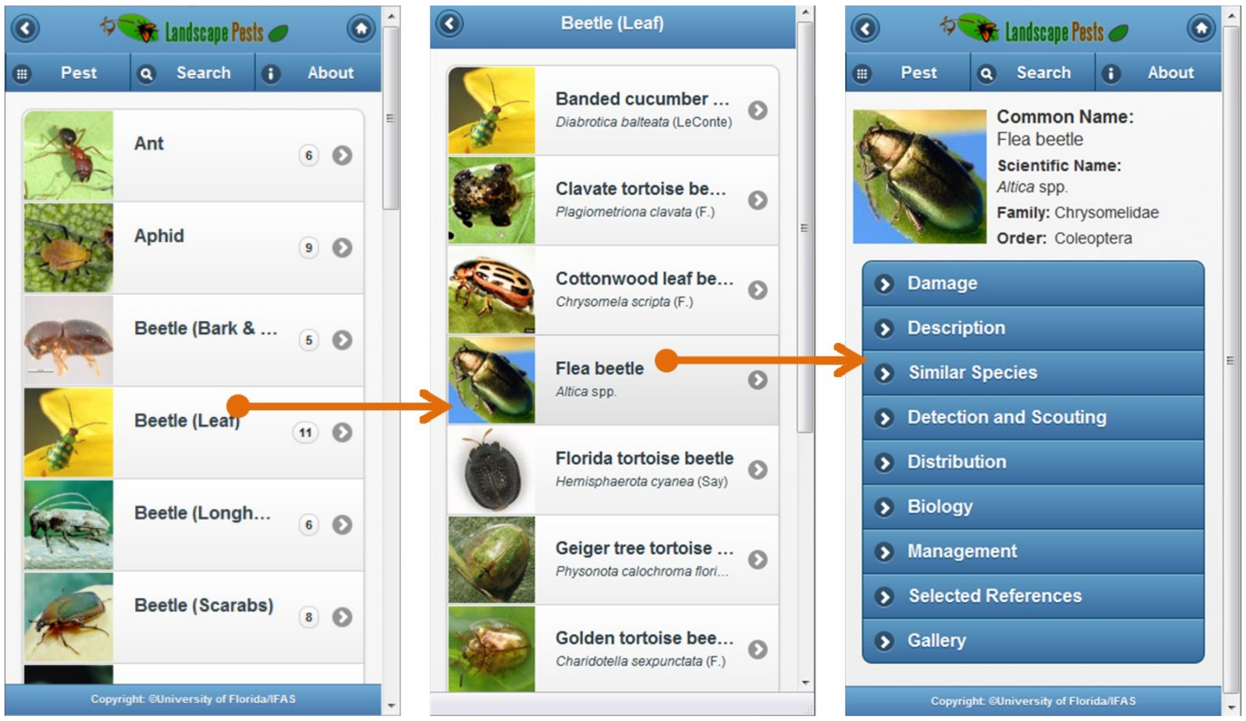

Landscape Pests
Pest
Search
About
Ant 6
Aphid 9
Beetle (Bark & ... 5
Beetle (Leaf) 11
Beetle (Longh... 6
Beetle (Scarabs) 8
Copyright: ©University of Florida/IFAS
Beetle (Leaf)
Banded cucumber ... Diabrotica balteata (LeConte)
Clavate tortoise be... Plagiometriona clavata (F.)
Cottonwood leaf be... Chrysomela scripta (F.)
Flea beetle Altica spp.
Florida tortoise beetle Hemisphaerota cyanea (Say)
Geiger tree tortoise ... Physonota calochroma flori...
Golden tortoise bee... Charidotella sexpunctata (F.)
Landscape Pests
Pest
Search
About
Common Name: Flea beetle
Scientific Name: Altica spp.
Family: Chrysomelidae
Order: Coleoptera
Damage
Description
Similar Species
Detection and Scouting
Distribution
Biology
Management
Selected References
Gallery
Copyright: ©University of Florida/IFAS

Figure 4. Pest list page demonstration.

3.3.3 Search

As Fig. 5(1) shows, the app has four search functions by: 1) plant host, 2) damage symptom, 3) common name, or 4) scientific name. Tapping on the “by Damage Symptom” button, the page will switch to a list of damage symptoms. After choosing a symptom, it will show a list of pests related to the symptom and grouped in categories, same as the layout of the *Pest List* page. And once a pest is selected, the page will redirect to the *Pest Detail* page.

“Search pest by host” works in a very similar way except host itself also has one level of indexing, shown in Fig. 5(6). Text filter are added to the long host lists to make it easier to locate a host. Searching by common name and scientific name subpages are about the same in the layout except that on the right side, a sorter bar is also added to avoid too much scrolling in a long list.

Host, index, pest-type and *pest-damage* objects in JSON representation are all loaded before page rendering. When choosing search by common name or scientific name, *index* objects will be sorted by the corresponding name so that the list can be shown in ascending order of the pest name initials.

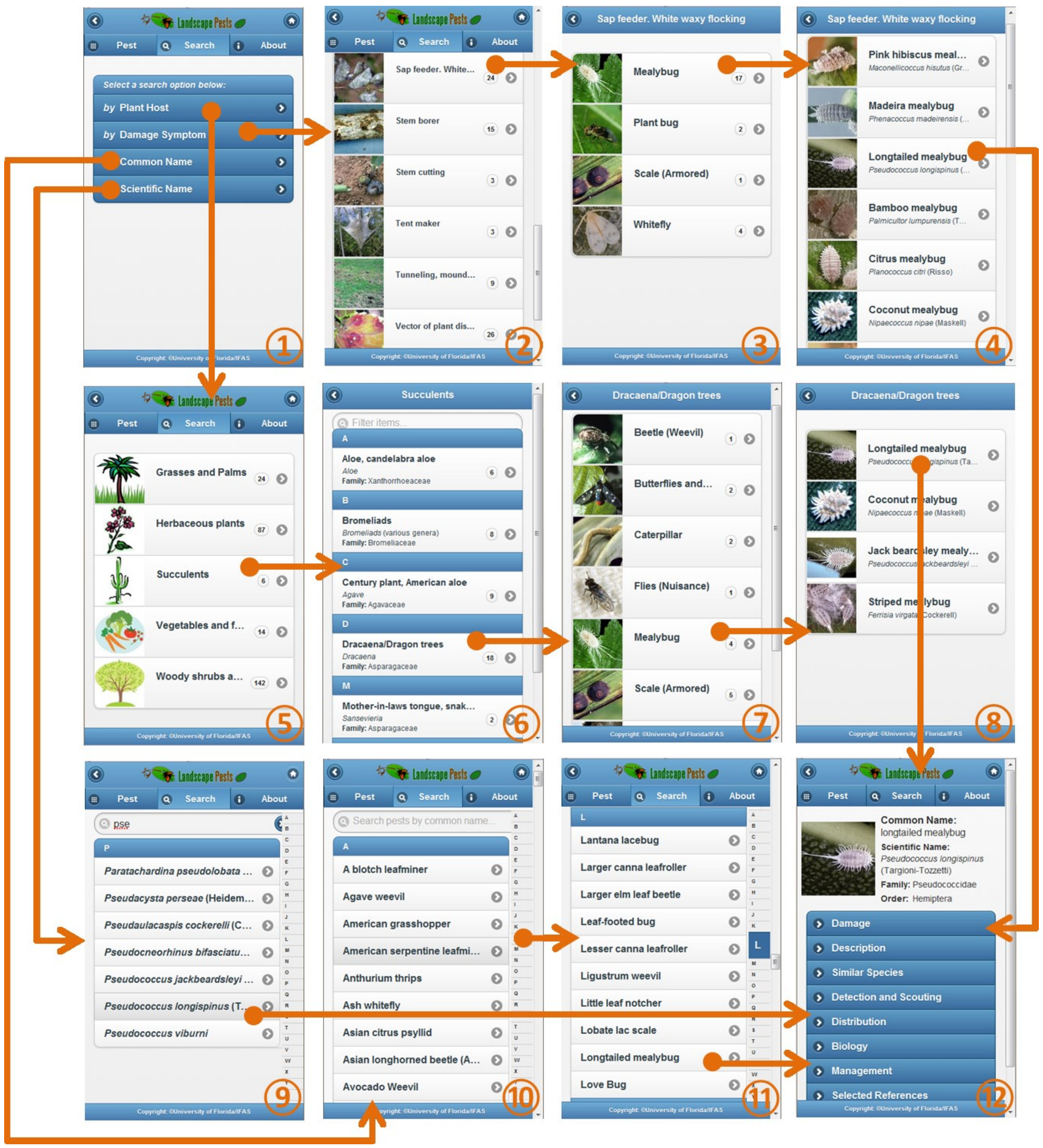
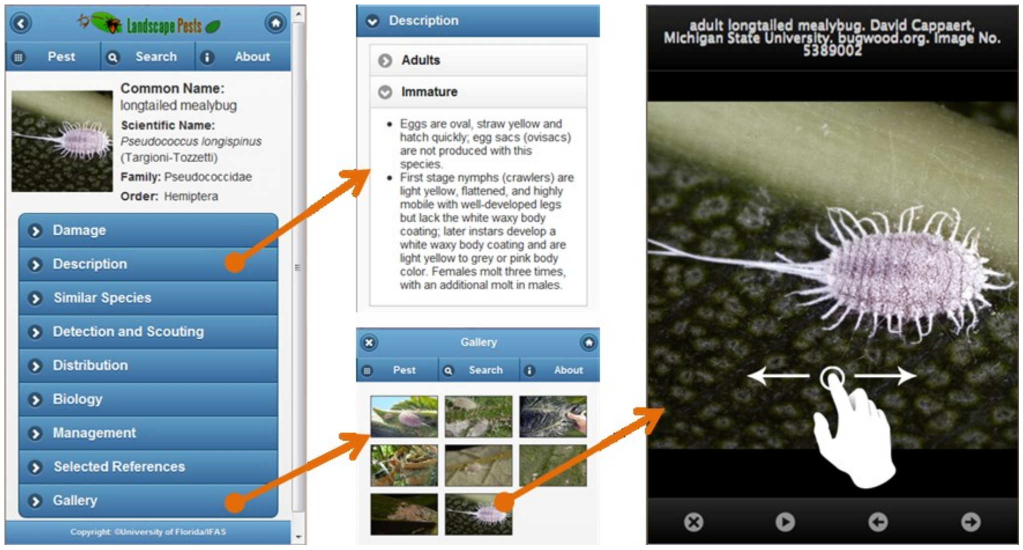


Figure 5. Search page demonstration.

3.3.4 Pest Detail

Pest Detail page is the only place that shows the detailed pest information other than basic biological information. When the page is called by the *Pest List* or *Search* page, the pest ID (p_id) is passed through URL, e.g. “/pest.html?p_id”. Until this page, *pests* object will just be loaded and the render will take out the (p_id–1)th element from the *pests* object for content. Therefore, it will take a little longer to download the data file for the first time when users view this page from web (app is not affected). Once the JSON object is stored in the cache, the page will show faster. Fig. 6 shows the *Pest Detail* page. Tapping on a title will reveal the related content by expanding the block while the next tap on the same title or any other title will fold the current block. Another feature of the *Pest Detail* page is the high definition image gallery. The gallery simulates the finger swipe effect on the native app for the web page.



Common Name: longtailed mealybug
Scientific Name: *Pseudococcus longispinus* (Targioni-Tozzetti)
Family: Pseudococcidae
Order: Hemiptera

- Eggs are oval, straw yellow and hatch quickly; egg sacs (ovisacs) are not produced with this species.
- First stage nymphs (crawlers) are light yellow, flattened, and highly mobile with well-developed legs but lack the white waxy body coating; later instars develop a white waxy body coating and are light yellow to grey or pink body color. Females molt three times, with an additional molt in males.

Figure 6. Pest detail page demonstration.

4. SUMMARY

This paper proposed a cost effective development approach for cross-platform mobile development, for web and mobile devices. The solution has been successfully implemented to the Southeast Landscape Pests project. However, this solution is most suitable for relative small size projects since the speed of dynamic page generation is highly affected by the bottleneck of dynamic data insertion on the HTML page. When data size gets larger, the response time may increase to seconds that could affect user experience. But still in many cases, users may not be viewing all the data on the page's initial loading, but gradually seeing the remaining contents when they scroll down the page. So with lazy load technique, the speed problem could be solved.

On the other hand, limitation of using JSON instead of database will be revealed in the maintenance since the simple add or delete operations cannot be done using a simple query like in a database, but requires manually looking up and formatting. In the future, when IndexedDB is supported by Safari mobile, it could be a better cross-platform data storage solution.

5. REFERENCES

Hammershøj, A., A. Sapuppo and R. Tadayoni, 2010. Challenges for mobile application development, Proc. of ICIN 2010, Costa Rica.
Qiang, C. Z., S. C. Kuek, A. Dymond and S. Esselaar, 2012. Mobile Applications for Agriculture and Rural Development, Washington D.C.: World Bank.
Tracy, K.W., 2012. Mobile application development experiences on Apple's iOS and Android OS, IEEE Potentials, Jul-Aug, vol 31, issue 4, pp 30-34.
USAID, 2011. Software platforms for mobile applications for agriculture development.