Calvary Chapel Corvallis

Progress Report

CS 463 Spring 2017

Kevin Stine, Courtney Bonn, Maxwell Dimm

Group #62

Abstract

The purpose of this project is to produce an iOS/Android application for Calvary Chapel of Corvallis that will allow members to access a plethora of information all in one localized space. The Church's current website does not provide an interface where current members of the church can very quickly access important information such as events, bulletins, and messages from the service. The desired application will be simple enough for anyone to use while providing back end access for staff to easily upload new information to the app. The priorities lie in maximizing the usability of the app and providing bulletin, schedule, video, and giving functionality. We will work with the existing Calvary Chapel web development team to create a product that is seamlessly integrated with their already existing network.

I. Purpose and Goals

The purpose of our project is to create an application for Calvary Corvallis Church that will act as a connection between the congregation and the administration. The church already has a website that has some of this information, but they want the website and app to serve different functions. The website will be to introduce people to the church. The app will be used for the existing congregation as the go to place to access the most commonly used or needed information. Some of the features being provided within the app are: having sermons available, listing the bulletin, having the church schedule, and allowing members to donate to the church. Our client explained to us that these were the features that they wanted in the app as they are the most needed services by their members.

Our client has requested that the app be as automated as possible in regards to updating the information hosted within it as to reduce any upkeep as much as possible. So we will be working with their existing infrastructure as much as possible to pull our information from. We are also creating both an iOS and Android application and we want to make the applications as functionally similar as possible. This will allow for greater understanding of the app across users who may or may not be super tech savvy. Our final goal in this project is to reduce costs wherever possible for our client. If that means suggesting newer cheaper infrastructure or setting up our app in a way that reduces how often it will need to be updated, we want to do it.

II. Fall 2016

During the first few months of working with this project, we focused on learning what our client wanted, determining the exact requirements, designing the projects, and learning how to develop mobile applications. Though no implementation took place during this time, a large bulk of the design decisions and specifics about the applications were completed which set us up for a smooth transition into development. The requirements of the project were decided between us and the client. The main requirements included a bulletin page, a calendar, the ability to donate, and the ability to watch the most recent sermon. Additionally, the client wanted a simple interface that looked similar to their current website and was not cluttered with too much information. Other requirements included having the application available on both iOS and Android smart phones and working with their existing back end software and website.

To meet the requirement of having the app available on multiple platforms, we decided to create two native applications using Xcode and Android Studio. We used the incorporated Interface Builder to design the iOS app and we used a navigation drawer system in Android Studio. The bulletin page was originally going to work with their back end software, Church Community Builder, but after the church changed their current website to Wordpress, we were able to pull directly from their website page. The events are being extracted from Church Community Builder and the messages page is working with LiveStream.

III. Winter 2017

Most of the implementation of the iOS implementation was done during Winter term. The first couple weeks were spent continuing to learn how Xcode and Android Studio worked and getting our base applications up and running. Once we grasped the basic development concepts, we began working on our individual pieces. Originally, the project was divided the following way:

- Bulletin Page and iOS Interface: Courtney

- Events Page and Android Interface: Kevin
- Messages and Donations: Max

However, assignments were shifted when individual pieces were more difficult than we originally thought. Throughout the development process, the project shifted in the following way:

- Bulletin Page, Events Page (Android), Donation Page (Android): Courtney
- Events Page (iOS): Kevin
- Donation Page (iOS), Messages Page: Max

We discovered that the Events page was more involved than we planned for and ended up needing to split the responsibilities in order to have this page finished on time. The specifics of each page will be described below.

By the end of Winter term, we had two almost completed applications. We focused on the iOS implementation because that was the one we began first and spent the most time on. The iOS app had the home page, bulletin page, donation page, and most of the messages page finished at this point. The events page was still being worked on but it was in progress.

The Android implementation was slower to begin and was about halfway done at the end of this term. Only the bulletin and the donation pages were finished by this point.

IV. Current Status

Beginning Spring term, we still had quite a bit of work to finish before the apps were completed. Though the goal was to finish development by Winter term, this did not happen. We focused on finishing the Android app as it had the furthest to go before being done. At the same time, we did continue to work on the iOS app to try and get them both complete as soon as possible.

As of now, both apps are finished, though neither are available for download in their respective stores. The client has decided to continue working on the app and adding additional information, which is discussed below, and they want to hold off on publishing the applications. However, our original requirements have been met with the apps as they are.

On the iOS app, an XML Parser was used to request and parse the public calendar from Church Community Builder. The parsed information was then presented in a UITableView. When a user clicks on each event, they can see an additional view with more details, like time, location, group leader, and contact information. Here is a code snippet from the iOS version of the EventViewController:

```
func createDatePicker() {
    // format the picker
    datePicker.datePickerMode = UIDatePickerMode.date
    //toolbar
    let toolbar = UIToolbar()
    toolbar.sizeToFit()
    // bar button item
    let doneButton = UIBarButtonItem(barButtonSystemItem:
                        .done, target: nil, action: #selector(donePressed))
    toolbar.setItems([doneButton], animated: false)
    changeDate.inputAccessoryView = toolbar
    changeDate.inputView = datePicker
}
```

This code creates the datePicker which allows the user to select the month, day and year. While the day function does not work (per our client's request), it does allow the user to select the month and year. This sets up the date picker using the UiDatePicker, and adds a UIToolbar to the top which allows the user to cancel picking the date, or pass the selected date to be set as the end URL.

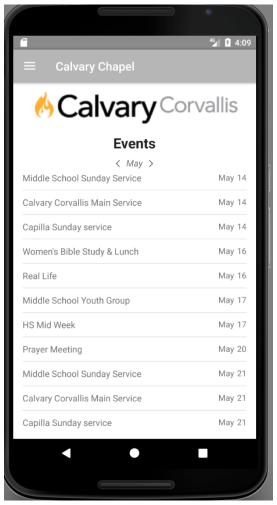
```
func donePressed() {
    // format date
    let dateFormatter = DateFormatter()
    dateFormatter.dateFormat = "yyyy-MM-dd"
    pickerTracker = true
    startDate = dateFormatter.string(from: datePicker.date)
    updateTable()
    self.view.endEditing(true)
}
```

This code is the selected action for the DatePicker. When the user pressed done, the selector is set and will call the donePressed function which sets the new start date to the one the user selects.

The Android app is very similar, though with the language difference, a new XML Parser was needed. Again the information is requested and parsed from the public calendar, and then presented in a List View. Users are also able to click on each event and view additional details. A code snippet from the XML Parser is listed below:

```
 public String getXmlFromUrl(String url) {
        OkHttpClient client = new OkHttpClient();
            final String basic = "Basic " + Base64.encodeToString(CREDENTIALS.getBytes(),
            Base64.NO_WRAP);
            String str = null;
            Request request = new Request.Builder()
                    .url(url)
                    .header("Authorization", basic)
                    .build();

            try {
                Response response = client.newCall(request).execute();
                str = response.body().string();
            } catch (IOException e) {
                e.printStackTrace();
            }
        return str;
    }
```

We used OkHttpClient in order to request the XML response from the URL. Once we received the response, we parsed it using a Document Builder Factory. The end results on how this response is displayed on the app can be seen in Figure 1.

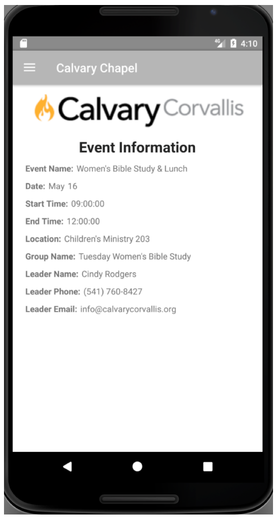


Events

May

Middle School Sunday Service	May 14
Calvary Corvallis Main Service	May 14
Capilla Sunday service	May 14
Women's Bible Study & Lunch	May 16
Real Life	May 16
Middle School Youth Group	May 17
HS Mid Week	May 17
Prayer Meeting	May 20
Middle School Sunday Service	May 21
Calvary Corvallis Main Service	May 21
Capilla Sunday service	May 21

(a) A List of Events



Event Information

Event Name: Women's Bible Study & Lunch
Date: May 16
Start Time: 09:00:00
End Time: 12:00:00
Location: Children's Ministry 203
Group Name: Tuesday Women's Bible Study
Leader Name: Cindy Rodgers
Leader Phone: (541) 760-8427
Leader Email: info@calvarycorvallis.org

(b) An Event Details

Fig. 1: Android Event Page

The Bulletin page is an exact replica of the corresponding page on Calvary's website on both the iOS and Android app. The church was in the process of creating a new website that was powered by Wordpress. Wordpress sites can be parsed using JSON and REST API. The response from the request is a JSON Object that can be parsed and information can be pulled from the response. A JSON Parser was used to request the bulletin page on the website and the response we received was the content of the page. A sample response from the JSON request was:

```
{
  "id": 1038,
  "date": "2016-10-27T19:22:53",
  "date_gmt": "2016-10-27T19:22:53",
  "guid": {
    "rendered": "http://www.calvarycorvallis.org/?page_id=1038"
  },
  "modified": "2017-03-18T11:48:50",
  "modified_gmt": "2017-03-18T18:48:50",
  "slug": "bulletin",
  "status": "publish",
  "type": "page",
  "link": "https://www.calvarycorvallis.org/bulletin/",
  "title": {
    "rendered": "This Week&#8217;s Bulletin"
  },
  "content": {
    "rendered": "<p>all bulletin content would be here...
    ...
    },
    Additional, unrelated JSON returned below here...
 }
```

From this response, the "content": { "rendered": .. } was the part that contained that actual bulletin content that would need to be displayed on the application. The parsing for iOS is listed below:

```
 do {
        guard let bulletin = try JSONSerialization.jsonObject(with: responseData,
        options: []) as? [String: AnyObject] else {
                 print("error trying to convert data to JSON")
                 return
         }


                guard let bulletinContent = bulletin["content"]?["rendered"] as?
        String else {
                  print("Could not get bulletin content from JSON")
                  return
         }
         let actualContent = bulletinContent.replacingOccurrences(of: "<[^>]*.", with:
         "", options: .regularExpression, range: nil)

         DispatchQueue.main.async{
                  self.jsontext.text = actualContent
         }
} catch  {
         print("error trying to convert data to JSON")
         return
}
```

The parsing for Android, which was simpler to extract the needed information is below:

```
if (response != null) {
        try {
                JSONObject jsonResponse = response.getJSONObject(TAG_CONTENT);
                String jsonData = jsonResponse.getString(TAG_RENDERED);
                textView.setText(jsonData);
                 Log.e("App", "Success: " + response.getString("yourJsonElement"));
        } catch (JSONException ex) {
                  Log.e("App", "Failure", ex);
        }
}
```

In order to style the content similarly to the way the website is styled, the response was wrapped in HTML and CSS and loaded into a Web View. This was done slightly differently but very similarly within each app.

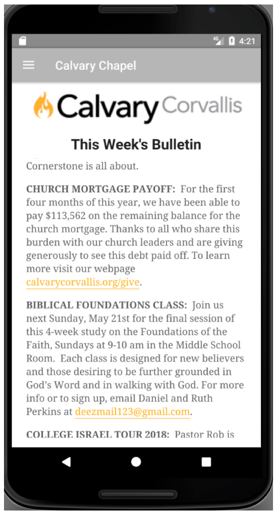
For iOS, loading into a UIWebView was as follows:

```
let htmlCode = "<!DOCTYPE HTML><html><head><style> body {color: #5b5e5e; font-family:
'Lora', Palatino;} a { border-bottom: 1px solid #fbaf17; color: #fbaf17;
text-decoration: none; }
.staff a { border-bottom: 0px none; } a:focus, a:hover { border-bottom: 1px solid #fbaf17;
color: #b17b0e; }</style></head><body>" + bulletinContent + "</body></html>"

self.bulletinWeb.loadHTMLString(htmlCode, baseURL: nil)
```

For Android, loading into a Web View is listed below:

```
String bulletinContent = "<!DOCTYPE HTML><html><head><style> body {color: #5b5e5e;
font-family: 'Lora', Palatino;} a { border-bottom: 1px solid #fbaf17; color: #fbaf17;
text-decoration: none; } .staff a { border-bottom: 0px none; } a:focus, a:hover {
border-bottom: 1px solid #fbaf17; color: #b17b0e; }</style></head><body>" +
jsonData + "</body></html>";

myWebView.loadDataWithBaseURL("file:///android_asset/", bulletinContent,
"text/html", "utf-8", null);
myWebView.getSettings().setAllowFileAccess(true);
```

With the above listed code, the bulletin page was rendered on the app and styled to look almost exactly as it does on the website. The resulting bulletin page on both devices can be seen in Figure 2.



This Week's Bulletin

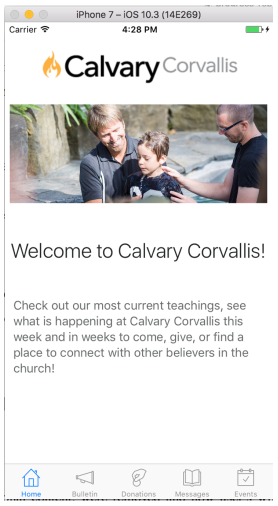
Cornerstone is all about.

CHURCH MORTGAGE PAYOFF: For the first four months of this year, we have been able to pay $113,562 on the remaining balance for the church mortgage. Thanks to all who share this burden with our church leaders and are giving generously to see this debt paid off. To learn more visit our webpage calvarycorvallis.org/give.

BIBLICAL FOUNDATIONS CLASS: Join us next Sunday, May 21st for the final session of this 4-week study on the Foundations of the Faith, Sundays at 9-10 am in the Middle School Room. Each class is designed for new believers and those desiring to be further grounded in God's Word and in walking with God. For more info or to sign up, email Daniel and Ruth Perkins at deezmail123@gmail.com.

COLLEGE ISRAEL TOUR 2018: Pastor Rob is

(a) Android



Welcome to Calvary Corvallis!

Check out our most current teachings, see what is happening at Calvary Corvallis this week and in weeks to come, give, or find a place to connect with other believers in the church!

(b) iOS

Fig. 2: Bulletin Page

Because the Donation page handles sensitive information, we decided with our client not to have the application handle the private information. Instead, the entire donation page was loaded into a UIWebView from the Calvary website. The header, footer, and additional content, were removed and now user's will have the ability to donate quickly while still protecting their credit card and personal information.

On the iOS app, the code that was used to load this page is listed below:

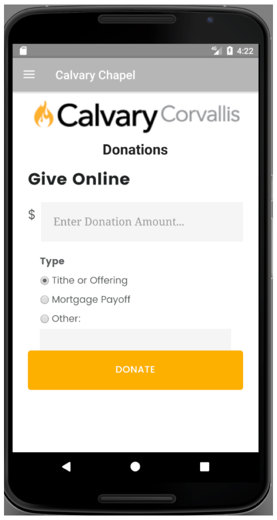
```
let donateURL = URL (string: "https://www.calvarycorvallis.org/give/")
let requestObj = URLRequest(url: donateURL!)
donateView.loadRequest(requestObj)
donateView.delegate = self
donateView.scrollView.delegate = self
donateView.scrollView.isScrollEnabled = false
```

The code for handling the loading of this page for Android is listed below:

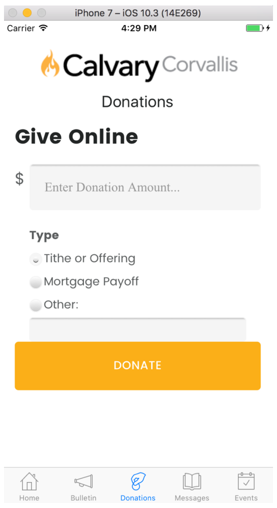
```
@Override
    public void onPageFinished(WebView view, String url) {
            myWebView.loadUrl("javascript:(function() { " +
            "document.getElementsByClassName('site-header')[0].style.display='none'; " +
            "document.getElementsByClassName('footer-widgets')[0].style.display='none'; " +
            "document.getElementsByClassName('content')[0].style.display='none'; " + "})()");
            myWebView.setVisibility(View.VISIBLE);
            myWebView.getSettings().setLoadWithOverviewMode(true);
            myWebView.getSettings().setUseWideViewPort(true);
        }
  });
```

```
        myWebView.setVisibility(View.GONE);
        myWebView.loadUrl("https://www.calvarycorvallis.org/give/");
```

The donation page on both devices can be seen in Figure 3.



(a) Android



(b) iOS

Fig. 3: Donation Page

The messages page is a hard coded video from Calvary's LiveStream account, along with a button to access previous videos. With the system our client is using we had two options to when it came to implementing the video viewer, either to use livestreams API and stream key, or to embed the video and use livestreams live link to keep the video current. The issue we ran into when going the livestream API route, is that we need the calvary corvalis unique key that would give us access to the developer tools. When asked about this our client looked but was unable to provide the key for us. The second option is what we were left with. We explained that in order to stream their live video we would need one of two things. The could either upgrade their livestream account which would open up the live embed link, or host their sermons on multiple platforms such as youtube which allow for free live streaming on different devices. Our client was not able to provide us with either of the above options so we were left with a hard coded link that just displays a past sermon. If they are able to do either of the above steps, changing a single line of code can easily set up livestreaming in the future.

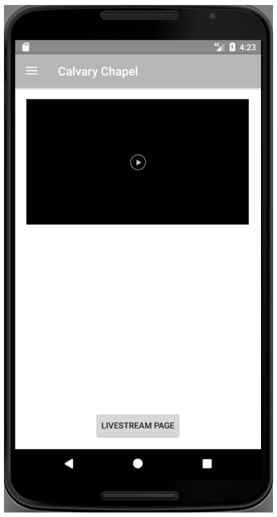
Below is the code used on the fourth fragment which is the messages page. This is from the android version which is almost identical to the IOS version save a few changes due to the differences between java and swift.

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
        myView = inflater.inflate(R.layout.fourth_layout, container, false);

        String videoLink = "<html><iframe id=\"ls_embed_1493363421\" src=\"
        https://livestream.com/accounts/18343788/events/7279945/videos/154327352/player
        ?width=960&height=540&enableInfo=false&defaultDrawer=&autoPlay=true&
        mute=false\" width=\"960\" height=\"540\" frameborder=\"0\" scrolling=\"no\"
        allowfullscreen> </iframe></html>";

        myWebView = (WebView) myView.findViewById(R.id.messagesView);

        WebSettings webSettings = myWebView.getSettings();
        webSettings.setJavaScriptEnabled(true);
        myWebView.getSettings().setLoadWithOverviewMode(true);
        myWebView.getSettings().setUseWideViewPort(true);
        myWebView.getSettings().setBuiltInZoomControls(true);
        myWebView.loadData(videoLink, "text/html", "utf-8");

        return myView;
```

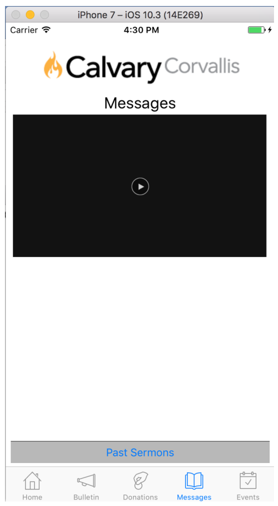
The button on the botton opens up a link to the livestream page in the users default browser. This gives the user the ability to search for past sermons so long as they still exist on the livestream page. Below is a function that I made that is tied to the button on the messages page, this links to the livestream page on their default browser.

```
public void browser1(View view) {
        Intent browserIntent=new Intent(Intent.ACTION_VIEW, Uri.parse("
        https://livestream.com/calvarycorvallis"));
        startActivity(browserIntent);
    }
```

The messages page on both devices can be seen in Figure 4.



(a) Android



(b) iOS

Fig. 4: Messages Page

V. Problems Encountered

We encountered quite a few stumbling blocks throughout the project that took a lot of our time. There were many small issues that arose throughout the project but that is to be expected when you are working on a type of project you never have before. The first issue we ran into was a problem linking the main storyboard to the view controllers. The problem arose when we tried to link visual components to the view controllers and then write the code. X-Code did not like this when they were not linked and threw a couple errors and mainly would not work. Now, this sounds like a pretty simple issue, but when you have not worked with x-code before, it takes a while to figure out. Once the issue was diagnosed it only took a few hours to fix, it was understanding the issue that took us a long time to figure out. The issue was hard to figure out because most tutorials online were single page apps and did not need to link additional view controllers. However when we fixed the problem we had the sermons page functional within a day or two.

Another issue that we had was with the calendar page on both iOS and Android. We were working on it for a long time using the CCB API. The issue was that the login cannot have the ”@” symbol within the username. The usernames issued to us by our client had that symbol within them and created a big issue for us. The error that we received was not clear enough for us to diagnose the problem quickly and ended up taking us a few weeks to resolve.

A third issue we ran into was the donations page. We decided to use the existing web donation platform that our client was already using except displayed through the app. The reasoning behind this was so that we would not have to handle any of the churches private banking information. The other reason being that the users would already be familiar and comfortable with this method. We displayed the donations page within a WebView in the app, however this created an unforeseen issue. Because of the nature of HTML pages and how they display, we had the whole website being displayed on the WebView, navigation, header, and footer included. We wanted just to have the donation link and nothing else from that page displayed. It was a head-scratcher but eventually we were able to figure out a javascript command that was able to strip all the undesired content away so that just the donation fields were displayed.

VI. Future Work

Within the last few weeks, the client has realized there are additional features they would like to be present on the applications. Because of time constraints, we were not able to complete these features for the client. However, the client does have a web development team that we will be turning the project over to and they will continue to develop and add to what we have developed.

The additional features include push notifications, multiple languages, the ability to fill out registration forms for events, and possible additional pages.

For the push notifications, the client wants to be able to alert their members when there is an event cancelled, such as a church service, or other emergency alerts. At this time, we are unable to implement push notifications. On the iOS app, notifications require a developer account to be able to implement and test. Our client does have a developer account and will have the ability to add notifications. On the Android app, notifications are sent out through Google Cloud Messaging which would add an additional service the church would need to keep track of. Our client has not decided whether or not they would want to add this service, so notifications will not be implemented on the Android app right now.

About halfway through development, the client wondered about the possibility of making the app available in different languages. Because this was not part of the original requirements, we set it as a stretch goal to complete if we had finished all other requirements and still had time to spare. As of this time, we were not able to implement this for the client as the original requirements took longer to complete.

Recently, the client wanted to add the ability to fill out registration forms for particular events. The theory would be that a person could click on an event, see a registration link, and fill out the form directly on the app and submit it to the church. Currently, we are trying to test this function with test events and are waiting for the client to add a test form. More than likely we will not be able to finish this implementation before we are finished with the project.

The client's senior staff has ideas of future pages or content that could be added to the app. Adding pages to the apps is a relatively easy task and will be explained to the development team when we are finished with the project. They are not sure what kind of content may be added, but they want to keep the possibility of more pages open.