**Kennesaw State University**

**College of Computer Science and Software Engineering**

Course: CS 4322 – Mobile Software Development

Fall Semester 2017

Instructor: Dr. Selena He

Due Date: December 5th, 2017

******

***The Weightlifting and Exercise Management Mobile Solution***

Richmond Mensah | Lead Developer & Backend Design | <rmensah4@students.kennesaw.edu>

Michael Bourgault | Frontend Design & Network Integration | [mbourgau@students.kennesaw.edu](file:///C:\Users\Michael\Desktop\A%20Fall%20Semester%202017\Distributed%20Computing\Project\mbourgau@students.kennesaw.edu)

Abstract

When we first proposed spotR, the goal was to create an app that would be easy to use, knowledgeable in workout information, not too user-reliant, but most of all, a useful tool for anyone who wants to plan their workouts in a fun, yet rewarding way. The concept came from a friend of Richmond’s who complained about all the non-satisfactory workout apps on the market today. “Every one of them does one thing right, but a million things wrong.” This is how spotR was born.

This app, when first proposed, was comprised of 4 basic functionalities, 2 expected functionalities, and 3 extra functionalities. The group was also comprised of three members as opposed to the present two. Our design was intended to be implemented in two separate phases. For the first phase, the focus was on implementing the base functionalities by the assigned deadline. The same goes for the second phase with the expected functionalities and extra functionalities only if it was judged that there was time to implement them by the final presentation.

While implementing this project, many challenges were faced, and many were overcome, but not all of them. Some challenges we faced came with designing a nice dynamic UI interface to house the workouts, sets, and reps that the user could add, modify, and delete. Over the course of the whole project, this proved to be one of the most difficult tasks because when we initially designed the UI, we did not set up the modules to modify the database or even to repopulate easily. Looking back, that is the biggest regret I have of designing this project.

Overall, the project was only a minor success. While many of our expected and base functionalities were completed, not all expected functionalities were achieved by the final deadline. On the other hand, while the app is still not at its full potential, there are plans to further the design of the app, like to fully integrate the database to display more than just the suggested workouts. We would also like to add in some of the extra functionalities like the running map and the progress report.

Despite a semester full of errors and long nights of coding, the final design did not live up to our initial expectations, but the experience taught us many things about designing in Android, as well as working in a group. If there is one thing that can be taken away from this experience, it is that designing a fully functional mobile app for production can be extremely difficult if you have never done it before. Without precise knowledge of many classes in android development, the work can be slow and more difficult than it needs to be. However, without this “trial by fire,” we might never have implemented certain functionalities in android that are now valuable skills for us as developers.

Introduction

A good workout is something everyone deserves, but sometimes it is just not that easy to plan. Sure, everyone wants to go to the gym, exercise, look great, and be happy about it, but sometimes it is a lot more complicated than it looks. These are the reasons we designed spotR. Whether users are experienced gym-goers who are looking for a way to mix up their workouts, or brand new to the gym and need help getting started, spotR’s functionalities should help guide users in the write direction.

Our first base functionality that we proposed was an automated weight calculation feature. With this feature, users are able to know how many reps and at what weight to lift at based on their current abilities (max bench, max squat, max deadlift) combined with their choice of lifting regiment (hypertrophy, strength, and power). Our second base functionality was a daily reminder push notification that would remind the user to workout based on their current “lifting regiment” choice. The third base functionality would be a calendar view that takes the user to different days of the year in which they can create and update workouts. Finally, the last base functionality, an in-depth user profile, would provide a login method and security for a user’s personal information, as well as give them exclusive rights to access our weight calculation feature and their own instance of the database.

Despite our full list of base functionalities, our only expected functionalities included integrating a YouTube video player so that the user can watch tutorial videos on exercises, and the ability for a user to take a video of themselves performing a workout to watch at a later time. At our mid-semester update, we made the decision to move the YouTube player to an extra functionality because of the unfortunate loss of our third member.

For the whole semester we used Android Studio as our IDE to implement the different functionalities. Some other technologies that spotR utilizes include Firebase Database, Messaging, Authentication, and User Profiles. Connecting the Firebase database to our project was a simple functionality that came integrated into Android Studio. For installing many of the dependencies, we did not need to understand much about Firebase itself, but when it came to actually implementing Firebase classes, a little more work came with it. For our presentations, we used a Galaxy S6 Active (SM-6890) with Android Nougat 7.1.1 running API 23. Because we only had one phone to test our app on, designing functionalities in the emulators would sometimes cause errors once they reached the phone. Towards the second half of the semester, to prevent this, we would test our code at the start of each week on the phone to make sure there are no flaws in the Gradle dependencies, the API, or some other unforeseen error.

From week one, we had a set plan for our design, the functionalities that we would implement, and what time we would spend to implement them. The following schedule, which we tried to follow to the letter, was presented at the proposal presentation:

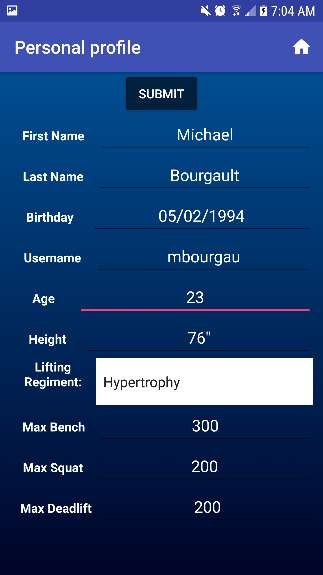
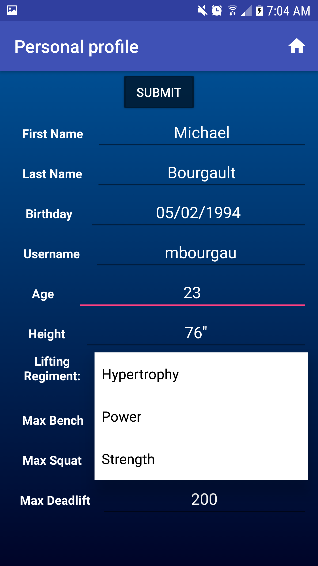
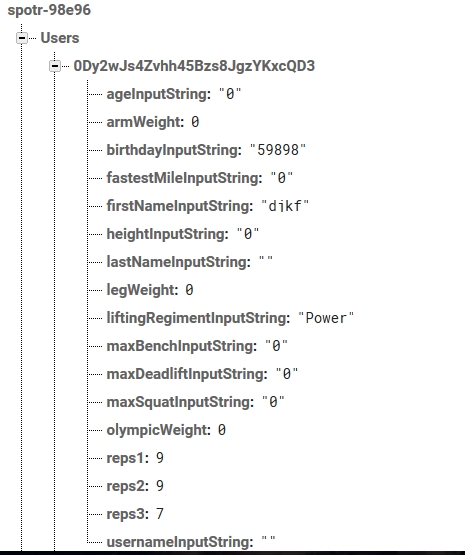
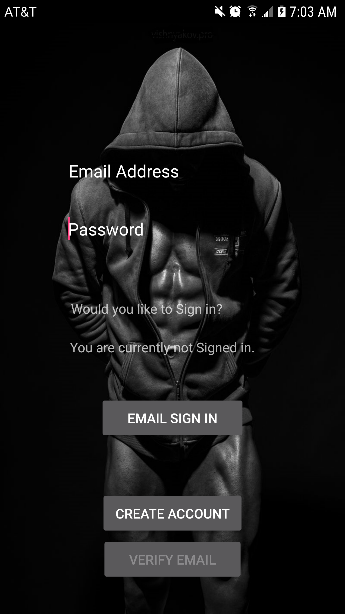
Week 1-5: Research, Week 6-8: User Interface Design / Testing

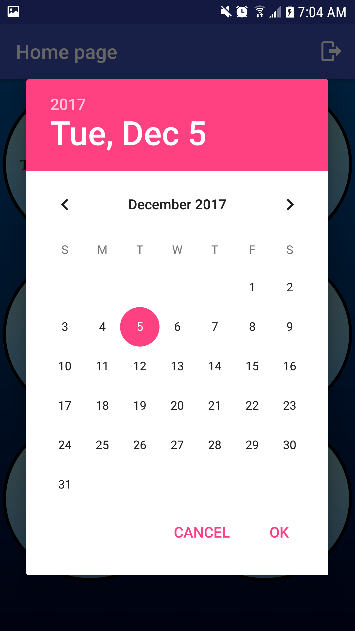
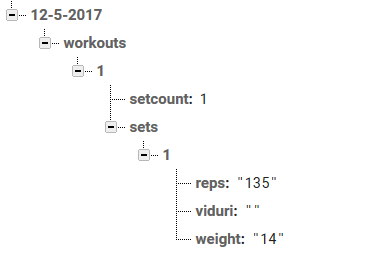
Week 9 - 12: Backend Functionality and Wiring / Testing, Week 13: Testing

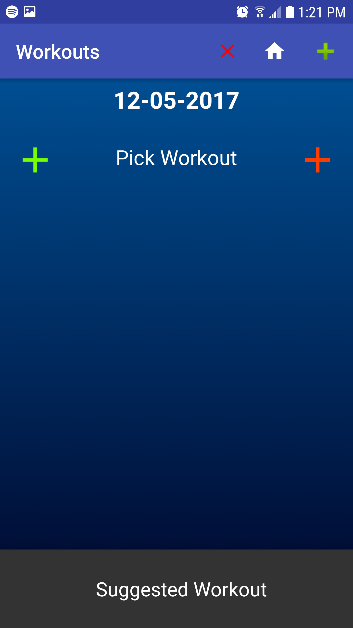
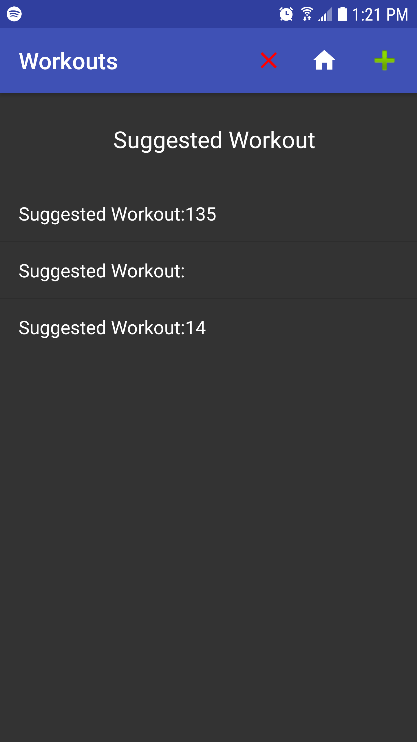
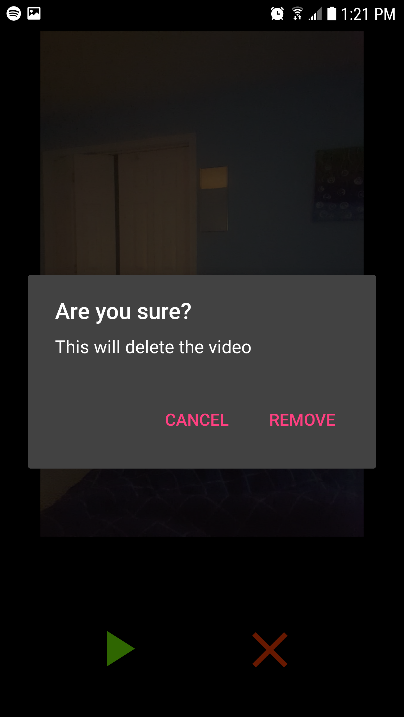
Week 14 - 15: Extra Functionality / Testing, Week 16: Final Presentation / Complete Product

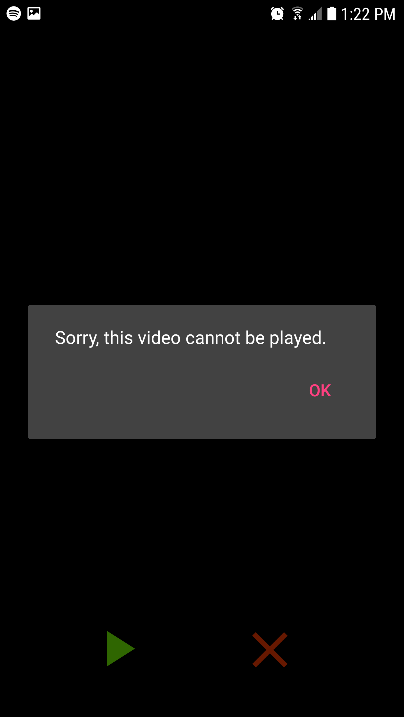
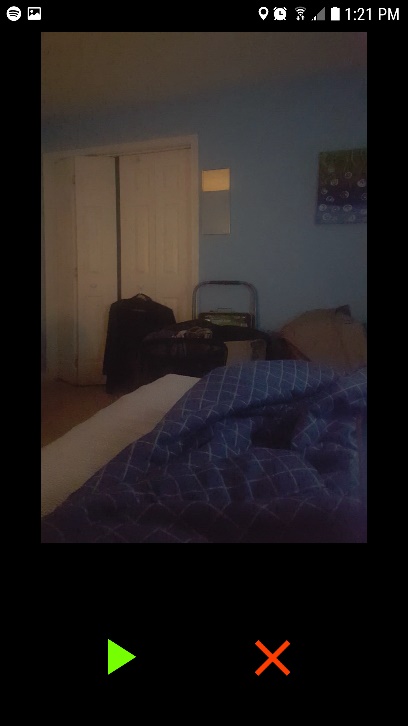
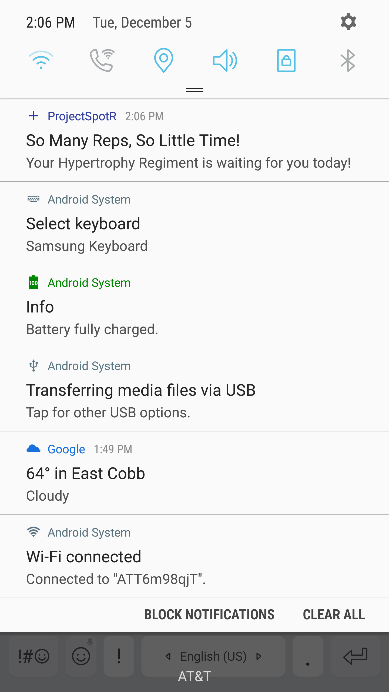
In our initial design schedule, we intended on splitting the tasks up in terms of workflow. Kyle, who was our UI designer until the mid-semester update, was tasked with getting us wireframes and building the layout files. Once he completed a layout, the plan was to start building the functionalities for that portion of the project. Up until week 9, we had limited communication from him, and it wasn’t until then that we were informed that he had dropped the class. This left us to gather up his tasks and divide them between ourselves to complete right before the mid-update presentation. This set us back in our own tasks, which led to a very sloppy presentation, and from then on had us behind schedule. For a time, we were able to get back on track and complete each of the base functionalities as they came up, but more and more tasks kept piling up which made the final implementation a much tougher goal for us.

Implementation

To implement the User Profile, first we had to get some sort of authentication system. The FirebaseAuth system allows us to create new profiles on a Firebase User Database that provides us direct authentication with just an internet connection. To create an account, we set up a button on the front end that takes the user input of Email Address and Password as parameters. That API call actually calls a POST on the Firebase Database which then creates a “unique” userID for each new user. While we are now capable of creating our own authentication system, it seemed logical at the time to use an off-device login. After the user creates an account, they must verify their email by clicking the now enabled verify email button at the bottom of the page. This button is furnished through firebase to send an email using the email that is in the Email Address TextView. Until the user receives this email and clicks the link in it, the user does not have access to login. When the user creates an account, the database that we created on Firebase is updated in the Users tree. The new user now has a blank profile to fill in. To sign in, we added implementation through Firebase Auth that gathers the user email address and password and matches them to the current email address and password in the respective TextViews. If these do not match, the TextViews show red and require that the correct password be input. This is simply an if case that determines whether the userID and password match each other. On the next page, the personal profile, we implemented a list of inputs that allow the user to change their personal profile information around. This is also where we implement the ability to change the user lifting regiment for the workout calculator. The submit button on this page is set to a function to update the Firebase Database using the “User” class that we made. By using the DataSnapshot method in a Firebase Database instance, we gather the current values of the TextViews, and implement and an onChange() method which helps update the values on the interface. This onChange() is called at the start of the activity and at any point in which data is changed in the database. The lifting regiment input required us to convert the value in the dropdown from an id to a String value, but other than that, this page did not provide any real issue for us.

Our Home Page is the next step in the workflow of the user. This page is relatively simple in that it is 6 clickable buttons with separate intents set to different pages, but some, starting with the Calendar View were apart of our base functionality, and required some work with intents and bundles. When the user of the application presses the calendar button, the application creates a dialog in which the user can select the desired date. After the user selects the date, the dialog sends the user to an activity where he/she will be able to select the desired workouts. The implementation of the dialog consists of the calendarView class. This class extends DialogFragment and implements DatePickerDialog.OnDateSetListener.  There are two methods of the DatePickerDialog.OnDateSetListener that need to be overridden: onCreateDialog and onDateSet. In onCreateDialog  I created an instance of Calendaar by invoking the Calendar.getInstance() method. The I use c.get(Calendar.YEAR), c.get(Calendar.MONTH) and c.get(Calendar.DAY\_OF\_MONTH) to specify integer values for the year, month and day of the month. The method returns a new DatePickerDialog() pith the context and date variables as parameters. The onDateSet() method  has a string named date created by appending the values the user selected. Next, there’s an intent named resultIntent2 that takes in the current context and the workoutPage.class as a parameter. The previously created string (date) is passed as a parameter to getContext().startActivity() along with extraDate. extraDate is the name of the extra data. getContext().startActivity(resultIntent2)  is responsible for launching the next activity with the extra data. This same implementation goes for the Today’s Date button that sits above the calendar button on the Home Page except that we gather the date from the Date class in the Java SDK.

Once on the workout page, our user is presented with a toolbar view including a Clear Workouts button and Add Workout Button. Some standard variables on this page are the date that we simply take out of the bundle passed in from the intent on the previous page, and the userID which is taken from the Firebase Auth class and set in Application Shared Preferences. With this information, we began to connect to the user database. By calling a reference to the database that creates a branch of data in the “Workouts” section of our database using the “DatesByWorkouts” class, we are able to use this to update and read from the information in the database. When we get to this point, we now have a date under the userID of the user in order to segment our database. With this information, we implemented our front end code. When the user presses add workout, a layout is inflated that includes the button that appears on the left and the right. These buttons call a similar function to call a set layout inflater that appends a set a list predefined under each “Pick Workout.” Within these classes we set up the database addition and subtraction. To do this, we simply called a reference to the database, and in our classes “SetsByWorkout” and “WorkoutByDate,” we added in a write to database functionality. When the user clicks on the “Pick Workout” this sends them to an activity that allows them to choose exercises. We implemented this page creating some arrays that contain the values of exercises that we wanted to include. To pick a workout, the user must simply click the list item that they would like to include. This is possible by using a list adapter to populate a list, and then creating an intent that sends a bundle back to the original screen. The search functionality on this page is simply a string match design that is set up inside of an onChange() function.

This portion of our app is still not fully functional. There are a collection of reasons we believe that this might be the case. The foremost of these ideas is discussed in the Prognosis section of this paper.On the bottom of the workouts page, we included a section known as suggested workout. The panel is implemented by using a library called “panel” that is created by an openSource user. This In our implementation, the purpose of this is to show the user the weights and reps that they should do based on their lifting regiment. We implemented this by showing getting the reps and weights that we stored by default in the “sets” portion of each workout in the database. By referring the the workout number and set number to the function fillData(), we should have been able to add the default workout data into the “suggested workout” panel. Unfortunately, this portion of our app was also not functional and the reasons for this are further discussed in our Prognosis section. In order to be able to record videos there needed to be permission set in the manifest. <uses-permission android:name="android.hardware.camera2" /> allows the application to access the camera hardware in order to be able to take videos. In the class workoutPage there are several buttons created, two of them are used for the video functionality. The first button (recorVideo) allows the application to record the video. The second (viewVideo) allows the user to view the video created.  In the recordVideo setOnClickListener there is a HashMap that saves the current view to be utilized later in order to actually view the video. An intent is first created with no parameters (callVideoAppIntent)  and then the action for the intent is set programmatically using allVideoAppIntent.setAction(MediaStore.ACTION\_VIDEO\_CAPTURE). Using startActivityForResult(callVideoAppIntent, ACTIVATE\_START\_CAMERA\_APP) starts the camera app and returns the video. After the video is captured, the onActivityResult() method is invoked and it checks for the request code and the result code. If they both match, it will proceed to save the  Uri of the video in the user defined array list that helps with managing all the videos created. The viewVideoButton allows the user to actually see the video. When pressed, it activates the setOnClickListener. First the Uri of the video is retrieved from the arraylist using the current view as an argument to the Hashtable toRecognizeView. After the value is retrieved, an intent by the name of  resultVideoIntent is created using workoutPage.this anf watchVideo.class as parameters. The putExtra allows the Uri string to be passed to the next activity where the functionality to watch the video is stored. startActivity() starts the next activity with the proper values. The watchVideo class sums up to the class receiving the values for the uri and setting a video view (watchVideo) using that same uri as a parameter. Watch.start() allows the user to watch the video.

The final functionality that we implemented was the Push Notifications. This is a service that is provided through the Firebase Messaging service and required little input on our part. To configure this, we set up the Postman messaging console to send push notifications based on a subscribe/write method. On the app itself, we had to subscribe based on the lifting regiment that someone would input on the user profile page. Most of this design happens in the Firebase Messaging Service class. Once the app is configured to receive push notifications through the Android Manifest we simply had to design the messages that we would send the app. In our case, we also set up the notification so that when the user clicks on it, the user is taken to the activity associated with the notification. To do this we set the title, body, image, and intent in the messaging service, off of the app itself.

Prognosis

Overall, the functionalities that we intended on having for the app were split between partially completed and fully completed. In the partially completed category lies the calendar view and video integration. In terms of the calendar view, our biggest downfall comes from the inability for the activities to fully repopulate. While we had this functionality working rather well before the final update presentation, in trying to solve another issue with the database, the feature malfunctioned and the database is no longer queried correctly. In terms of video integration, the incompleteness of this functionality also stems from the link to the database being incorrect. As stated above, the video integration allows a user to save videos, but without the database correctly storing the video URI and being able to retrieve it, our app cannot keep videos after repopulating just yet. The same issue goes for choices in workouts, reps, and weight. As of right now, the only repopulation that the workout pages are capable of is adding workouts and sets based on the amount from the previous session on that date.

Behind every setback in our project, there was a clearly impeding issue in our own implementation that could be considered a cause. In the case of our repopulation issue, the major problem arose with the design of the page itself. As stated above for the workout page, we chose to implement it using dynamic layout inflator files. Until a few weeks ago, our understanding of this concept was less than ideal. In our implementation, as I tried to pull from the database and populate the dynamic views with values, it did not initially occur to us that an Adapter class would solve this problem almost instantly. After many other futile attempts to fill the set text boxes with the current weight and reps, we finally resorted to using the adapter and had very little issues with it until the week before the final presentation.

The issue that we were trying to solve when we altered the code from being able to repopulate on its own was a database design flaw. The structure of the database, as pictured above, was going to be a major problem in the grand scheme of the app, but more importantly, we were not able to gather important data like the videoURI easily with the original structure. In attempting to debug the new issue with the layouts not properly repopulating, we can speculate that the major issue lies in how we are calling the database somewhere in our code. In testing, it seems that the program fails to detect information in the branch we are attempting to search completely, and will choose not to try to repopulate because it thinks the date has no workouts stored in it. The problem is that the date does have workouts in it, but they are not being detected by a query. In our experience with Firebase, it is likely that the problem is on our end and not on the actual database, but we are not ruling this possibility out. While the repopulation of the workouts was not an explicit functionality of the app, the lack of this functionality detriments some of the other functionalities further. If we can get the pages to repopulate again, the app should be able to provide full functionality for any user.

Conclusion and Future Work

Were we asked to start this app again from the beginning, there are two major design points that we would change. First, we would design spotR with a local LSQLite database as opposed to having a complicated cloud database. There are many benefits that come with a local SQLite database including the ability to workout offline, the chance to have faster asynchronous tasks, and the chance to have local user profiles which simplifies things for the user immensely. For our implementation, it did not make sense to use an online database, and it ended up being a more complicated endeavor than we could handle at the time. Second, we would look further into the best design for a User Interface for this type of app. While our design was alright, it could really use some “tightening up.” Testing was limited on our final product, so we aren’t exactly sure how optimal the design was.

Going forward, the app needs a lot of work. Aside from the two changes mentioned above, we might consider overhauling the app and trying to implement it given our new skills as opposed to those that we started the semester with. Our future goal is for the app to make the Google Play Store and eventually garner at least 1000 downloads. To get there, we will have to work very hard on implementing a better system the second time around.