GitHub Username: dylan-chu

SOTIP (State Of The IT Projects)

Description

Problem:
When I was doing research for my master's thesis, I found it difficult to gather statistics for the success and failure rates of IT projects that came from data that could be substantiated. Luckily, I stumbled upon the IT Dashboard (https://itdashboard.gov/drupal/), a US government website that tracks the performance of federal IT projects. To my delight, the data included potentially insightful bits of information such as each project's variances from its projected budget and schedule, the experience level of the project manager and the software development methodology used (if the project was a software development project).

I was especially interested in the success and failure rates of IT projects, which are usually determined by whether the projected completed on time, on budget and with the specified features. The IT Dashboard provided clear indicators for two of these variables, cost and schedule. Each project's variances from its estimated cost and schedule is labeled as either 'Green', 'Yellow' or 'Red'. For example, a project that exceeded its estimated cost by more than 30% is labeled 'Red'. However, a project that completed with its cost below its estimated cost by more than 30% is also labeled 'Red'. The same holds for true for a project's variance from its estimated schedule. This made it difficult to visualize how many projects were outperforming their estimates from the charts the IT Dashboard provided for each indicator.

I was also interested in factors that might influence the outcome of a project such as the experience level of the project manager or the adopted software development methodology. There were no graphs or tables that incorporated these factors.

To gather statistics that I wanted for my thesis, I had to download the data for all the projects, scrub the data, and import the cleaned data into a local database. After those steps were done, I had to manually query the data using SQL statements and finally calculate the ratios that I wanted. This was a time-consuming process and this process has to be repeated if I want to use the latest data, which is continually being updated.

Solution:
SOTIP is an app that allows researchers and teachers to have a peek at the state of the US government's IT projects. It provides a visualization of the data provided by the IT Dashboard from multiple perspectives and with a focus on the performance of the IT projects. Users can drill down on subsets of projects to gain more details for the IT projects that make up a subset.

Intended User

This app is made with researchers and teachers in mind. Researchers who are interested in why projects succeed or fail can examine the data provided by the IT Dashboard in summarized form and can drill down to look at individual projects. Teachers can show support for the validity of a few project management ideas/beliefs with the charts the app provides.

Features

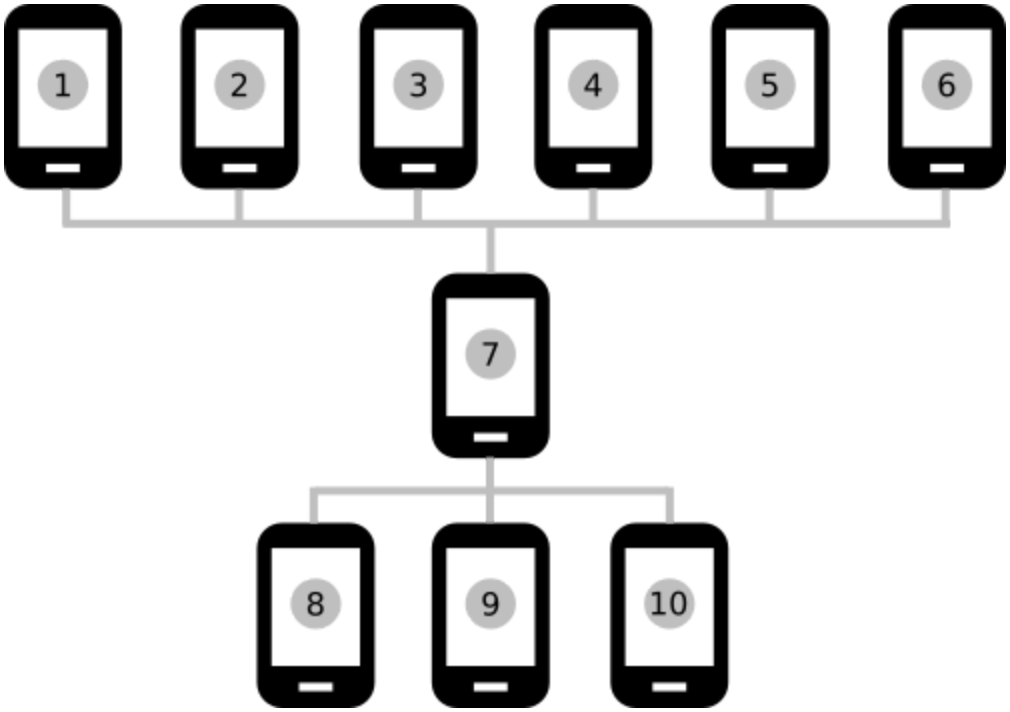
SOTIP allows users to:

- Visualize how US government IT projects have performed or are performing in relation to their estimated cost and/or schedule
- Visualize the performance of the projects when they are broken down by the project manager's experience level
- Visualize the performance of the software development projects when they are broken down by the development methodology
- View a list of IT projects that are similar in one aspect
- Get more details about an IT project, including information about the investment under which it belongs
- View the communication channels that could be used to ask for more information about a project or investment

User Interface Mocks

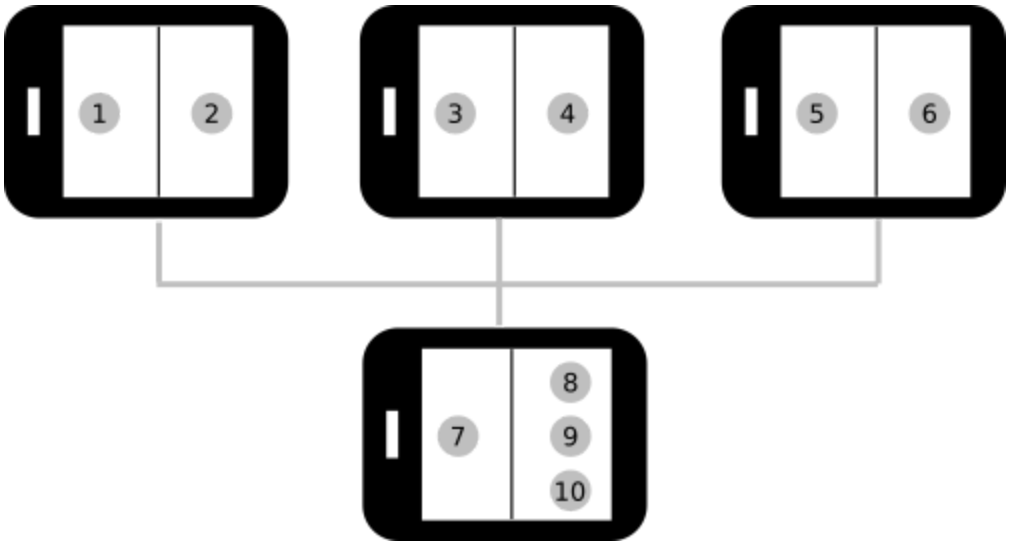
App flow for phones

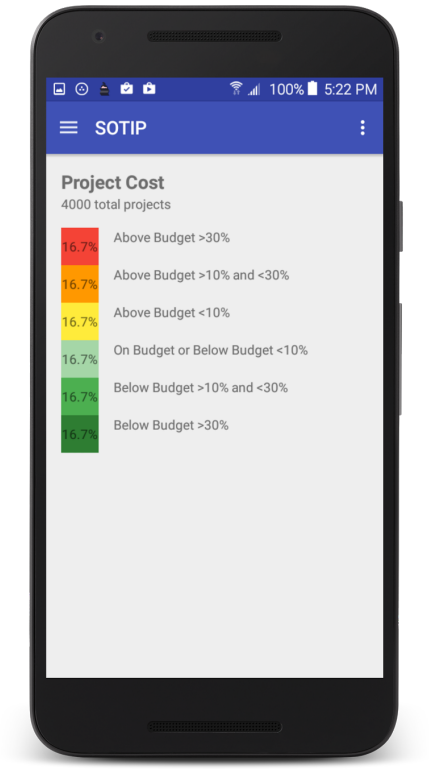
The diagram below shows the structure of the app when viewed on a mobile phone. A drawer navigation mechanism provides access to Screens 1 through 6. These screens can all lead to Screen 7, which can lead to a ViewPager containing Screens 8 through 10.



App flow for tablets

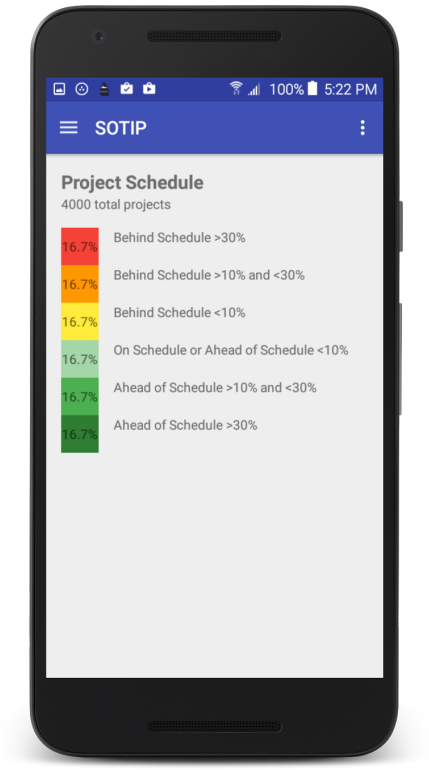
The diagram below shows how the screens from above are combined when displayed on a tablet. Similar to the UI for phones, a drawer navigation mechanism provides the ability to switch between the screens at the top of the diagram. Clicking on any of the charts or diagrams in these screens will open up another screen with a list of projects on the left side and a ViewPager on the right side.





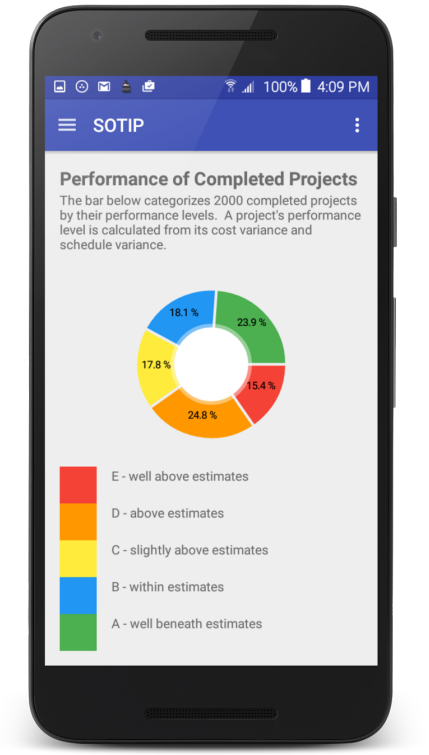
Screen 1: Cost Variance

This screen answers the question: How did the IT projects do or how are the IT projects doing in terms of their estimated cost? It shows 6 categories for the cost variance of a project and the percentage of projects that fit in each category. The categories, represented by 6 coloured squares, range from under budget by more than 30% to over budget by more than 30%. Clicking on a coloured square opens up Screen 7 with a list of the projects in that category.



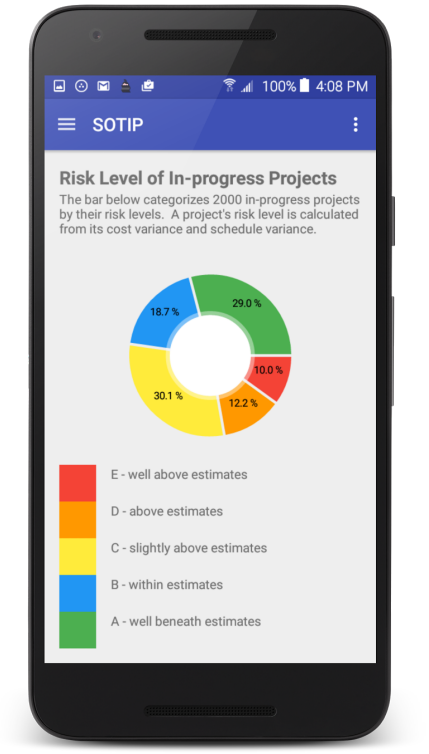
Screen 2: Schedule Variance

This screen answers the question: How did the IT projects do or how are the projects doing in terms of their estimated schedule? There are 6 coloured squares representing 6 categories of schedule variance. The categories are from ahead of schedule by more than 30% to behind schedule by more than 30%. Clicking on a coloured square opens up Screen 7 with a list of the projects in that category.



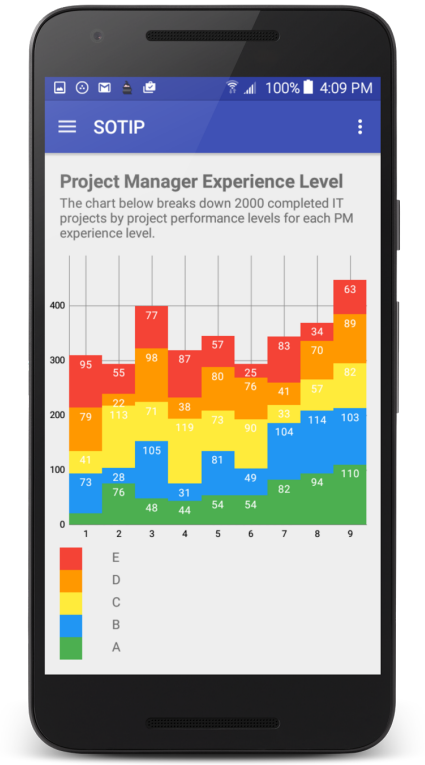
Screen 3: Performance of Completed Projects

This screen answers the question: How well did the completed projects perform in terms of their estimated cost and estimated schedule? The 6 categories for cost variance are given a value from 0 to 5. The 6 categories for schedule variance are assigned a similar range of values. These two values are added together to get a score from 0 to 10 for each project. The project scores are grouped into 5 bands and the percentage of projects that fit into each band is shown in the pie chart. Clicking on a pie segment opens Screen 7 with a list of the projects that are in that band.



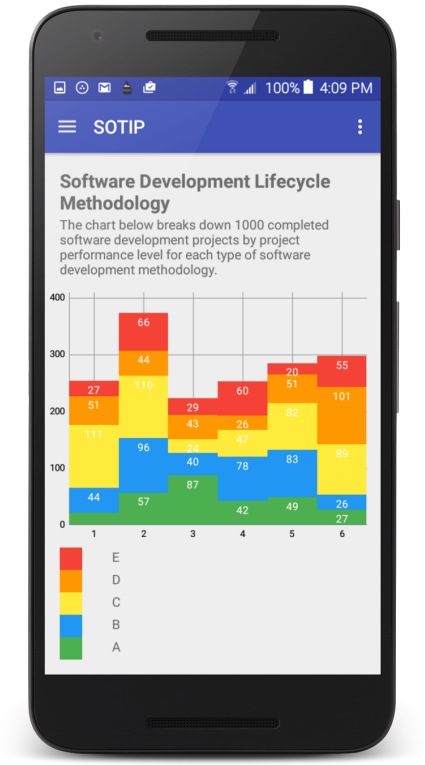
Screen 4: Risk Level of In-Progress Projects

This screen answers the question: How well are in-progress projects performing in terms of their estimated cost and estimated schedule? Each project that is still in progress is assigned a score in the same manner as the completed projects. Once again, clicking a pie segment opens Screen 7 with a list of the projects that are represented by that segment.



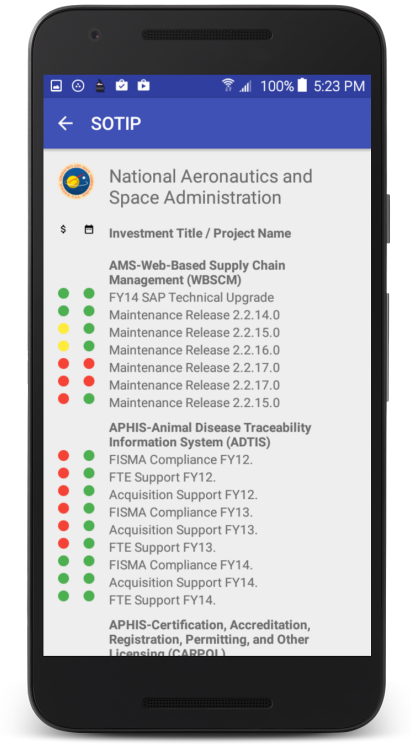
Screen 5: PM Experience Level

This screen provides insight into the question: Does the experience level of the project manager (PM) have an influence on the performance of the project? Each project is given a value in the range of 1 to 9 that correspond to one of 9 different levels of PM experience. This screen shows how many projects the PMs at each different level are managing and how the projects are performing. Clicking on a segment of the bar chart opens up Screen 7 with a list of projects that belong in that segment.



Screen 6: SDLC Methodology

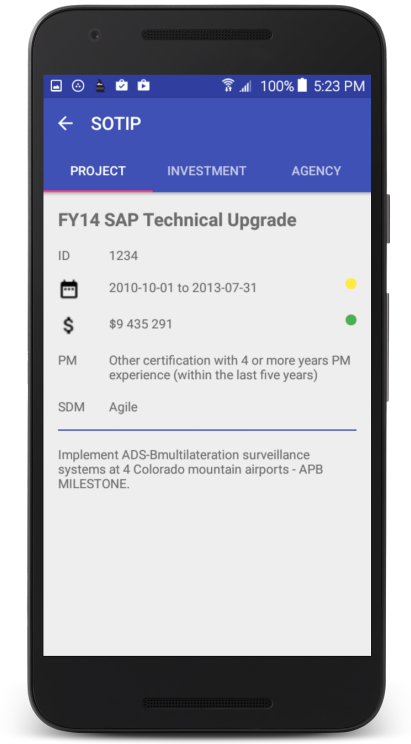
This screen provides insight into the question: Does the methodology used on a software development project have an influence on the outcome of the project? Each software development project has a value in the range of 1 to 6 that correspond to the type of software development lifecycle (SDLC) methodology used on the project. This screen shows how many projects use each methodology with a further breakdown on the number of projects performing at what level. Like the earlier screens, clicking on the chart opens up Screen 7.



Screen 7: Project List

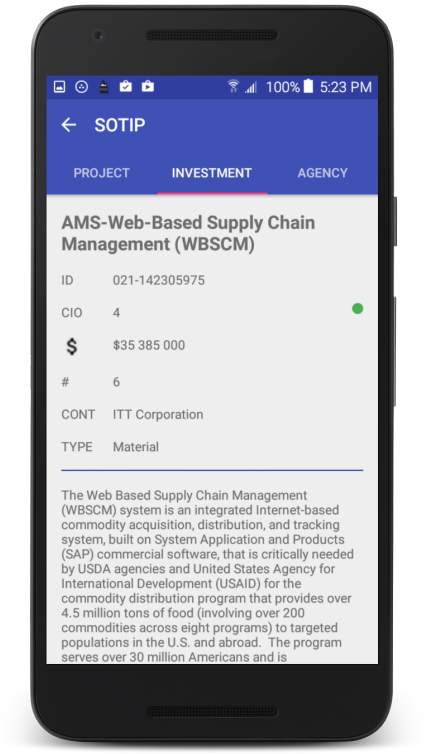
This screen shows a list of projects that belong to a subset of the data. Only the projects for one agency is shown in the list at any one time. The user can swipe left and right to navigate to the projects for another agency.

The projects are grouped together under the investment that they belong to and there are colour indicators for the cost and schedule variances for each project. Clicking on a project opens up Screen 8 with the details for the selected project.



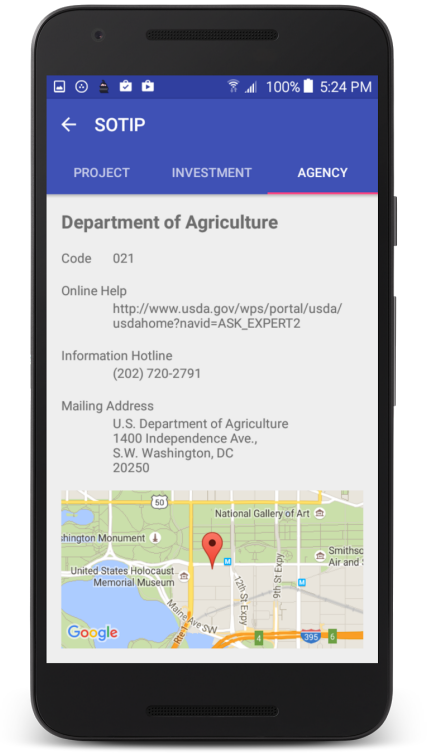
Screen 8: Project Details

The user may want to investigate projects that are poor performing or high performing to understand why. This screen, along with Screen 9 and Screen 10, give the user more details for a project. This screen show details that are specific to the project, including the start and completion dates, the budget and the objectives.



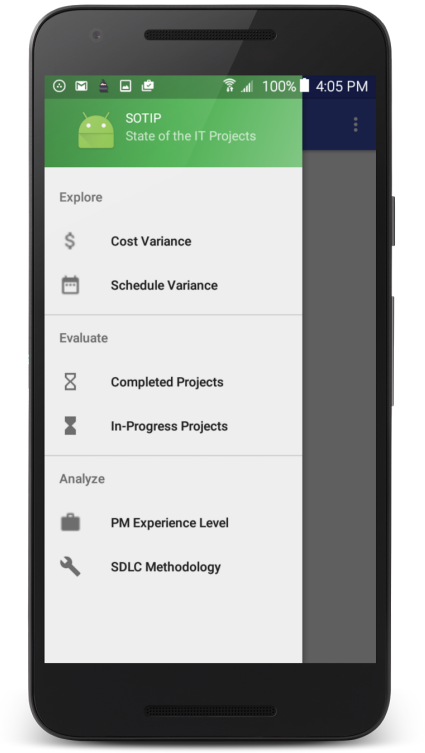
Screen 9: Investment Details

Most projects are not standalone projects but are undertaken as part of a greater strategy. These “investments” have more materials written for them, such as the business case for the investment, and reading these materials can provide a better understanding of the project. This screen displays some details for the investment under which the project belongs.



Screen 10: Agency Details

If the user would like more information about a project, one possible method for getting that information is to contact the agency responsible for the project. This screen lists the agency’s communication channels.



Drawer Navigation

The user can navigate to Screens 1 through 6 directly through this navigation mechanism. Each submenu item leads to a different screen. The 'Cost Variance' option leads to Screen 1, the 'Schedule Variance' option leads to Screen 2 and so forth.

Widget

The widget will show the same content as Screen 3, giving the user a peek at the performance of completed IT projects. Each band in the coloured bar represents a performance level. The width of a band is proportional to the percentage of the total projects that performed at that level. Clicking on the widget will open up the app showing Screen 3.

Key Considerations

How will your app handle data persistence?

The app will retrieve data from APIs from a remote server and store all retrieved data in a local database. It will use a Content Provider as the interface to the local database. As well, the app will sync with the remote server using a SyncAdapter. If a data sync is successful, the date of the sync is stored in SharedPreferences.

Note: The app will use data from 4 data feeds provided by the IT Dashboard but the app will not download from these data feeds directly. Instead, the app will get its data from API endpoints that I will create on another server using Amazon API Gateway and other Amazon Web Services (Lambda, DynamoDB, …).

The main reason for this design decision is that downloading from the data feed directly takes a long enough amount of time that can degrade the UX. Currently, there are over 4000 results for the data feed for the IT projects and each result has over 25 fields with rather long names.

A better approach is to download the data from the data feed, put it in a database on another server and then have that server expose APIs for accessing the data. Instead of returning the entire dataset, the APIs could return just updates to the data which contains just the fields the app needs and the fields have names that are abbreviated. Another benefit to this approach is that the APIs can return aggregated data so that the app does not need to derive the values for the charts.

Describe any corner cases in the UX.

The app is primarily a data visualization app and allows limited interaction so there does not seem to be any corner cases in the UX. Some of the screens will be implemented with more elegant designs than the mockups show but they will still remain relatively simple which makes it unlikely that the app will encounter any corner cases.

Describe any libraries you'll be using and share your reasoning for including them.

The app will use Retrofit to retrieve data from remote APIs. Retrofit simplifies the process of connecting to the remote server and extracting the returned JSON data.

The app will also use MPAndroidChart for creating the charts. There are features (such as labeling the entire X-axis and Y-axis) that are missing from this library and I will create a custom view if it is too difficult to implement a feature into the existing code for the library.

Next Steps: Required Tasks

Task 1: Project Setup

After creating a new Android project in Android Studio, edit the *build.gradle* file at the project level and at the *app* module level to be able to use two libraries: Retrofit and MPAndroidChart. The links below show how to update the files to have the right dependencies to use the libraries.

Retrofit: http://square.github.io/retrofit/
MPAndroidChart: https://github.com/PhilJay/MPAndroidChart

Task 2: Build the Server

Build the server. The server will download data from the following data feeds at a set time each day:

-https://itdashboard.gov/api/v1/ITDB2/dataFeeds/projects
-https://itdashboard.gov/api/v1/ITDB2/dataFeeds/businessCase
-https://itdashboard.gov/api/v1/ITDB2/dataFeeds/contracts
-https://itdashboard.gov/api/v1/ITDB2/dataFeeds/investmentRelatedURLs

Besides being saved in a database table, the data from the *projects* feed will be summarized and stored in another table called *charts*. The data from the other 3 feeds will be amalgamated and stored in a third table called *investments*. A fourth table, called *agencies*, with be manually populated with data for the agencies.

The following API endpoints will be exposed:

/projects/{date}
/charts/{date}
/investments/{date}
/agencies/{date}

(As this part is outside the scope of Android development, the implementation details will be excluded from this document.)

Task 3: Implement the Functionality for the Data Layer

Using the Retrofit library, implement the interfaces to describe the API endpoints and the container classes to hold the data that each endpoint returns. Each endpoint will return results

with fields that correspond to the columns (excluding the _ID column) for the respective database tables described below.

Implement the classes to implement a Content Provider. The local database should have 4 tables: *projects*, *charts*, *investments*, and *agencies*.

The schema for the *projects* table is as follows:

- _ID: int
- agency_code: text
- investment_id: text
- investment_title: text
- id: int
- name: text
- status: int
- start_date: text
- completion_date: text
- sch_var_indic: int
- sch_var_days: int
- lifecycle_cost: real
- cost_var_indic: int
- cost_var_amt: real
- pm_exp_lvl: int
- sdlc_method: int
- other_sdlc_method: text
- objectives: text
- updated_time: text

The schema for the *charts* table is as follows:

- _ID: int
- date: text
- type: int
- data: text

The schema for the *investments* table is as follows:

- _ID: int
- agency_code: text
- id: text
- title: text
- summary: text
- num_projects: int
- lifecycle_cost: real
- cio_rating: int
- contractor: text

contract_type: text
url1: text
url2: text
url3: text
updated_time: text

The schema for the *agencies* table is as follows:

_ID: int
agency_code: text
name: text
website, text
online_help_link: text
info_phone: text
mail_addr: text
latitude: real
longitude: real
updated_time: text

Implement the classes for a SyncAdapter. Schedule the SyncAdapter to connect to the API endpoints once a day and implement the connection and data retrieval functionality using the Retrofit classes. When setting query parameters, check in SharedPreferences for the last date that a sync was successful and pass in this date if one exists. Otherwise, pass in "1900-01-01". The APIs only return changes to the data that happened after the supplied date.

Using the Content Provider, modify the local database with the returned data. Except for the *charts* table, when modifying the other tables, check the rows in the table first to determine whether an update or an insert is needed.

Task 4: Implement the UI for Screens 1 through 6

Build the UI for Screens 1 through 6 from the UI mockups and populate the screens with fake data.

-Use a PieChart from the MPAndroidChart library for Screens 3 and 4.
-Use a BarChart for Screens 5 and 6.
-Use a Fragment for each screen.

Build the drawer navigation UI and implement the ability to switch between the 6 screens using this navigation mechanism.

Task 5: Implement the Visualization Features on Screens 1 through 6

In each Fragment class for Screens 1 through 6, implement the methods to use a loader. The loader will use the Content Provider to query the *charts* table for the data for each respective chart. For each row in the table, the value for the *type* column indicate the screen that the data is for. Parse the value returned for the *data* column into individual data items and populate the chart with the data items.

Flush out the click listener for each chart so that it passes forward a value that identifies the selected chart segment before it starts an Activity to display a list of projects on Screen 7.

Task 6: Implement the UI for Screen 7 and the List of Similar IT Projects Feature

Build the UI for Screen 7 from the UI mockup. Use a RecyclerView for the list and create the corresponding RecyclerView.Adapter class. Implement a loader that returns a cursor and populate the list using the cursor.

When obtaining the cursor, use the Content Provider to query the *projects* table with the passed in value (from any of Screens 1 to 6) to retrieve only a subset of the projects. As well, just retrieve the projects for the lowest numbered agency initially. If the cursor returns no results, query the Content Provider again for the next highest numbered agency. Any results returned should be ordered primarily by investment title and secondarily by project ID.

Each list item shows the following content:

- -investment title
- -project name
- -cost variance indicator
- -schedule variance indicator

The investment title is only displayed for the first project in the investment. The indicators should be converted into their visual counterparts.

Task 7: Implement Navigation Functionality for Screen 7

Implement gesture listeners that allows the user to swipe left and right to navigate through the agencies. In other words, since the list of projects displayed on Screen 7 is for only one agency, I want the user to be able to swipe left or right to get a corresponding list of projects for another agency. The agencies are identified by their agency code, which is a three-digit numeric code, so swiping right moves to the next highest agency code and swiping left moves to the next lowest agency code.

Implement click listeners for the items in the list. Pass along the project ID, investment ID and agency ID before starting an Activity showing the selected project's details with Screen 8.

Task 8: Implement the UI for Screens 8 through 10

Build the UI for Screens 8 through 10 by looking at the UI mockups. Use a Fragment for each screen.

Screen 8 shows the following content for a project:

- -project ID
- -start date
- -completion date
- -project lifecycle costs
- -PM experience level
- -SDLC methodology (if applicable)
- -objectives

Screen 9 shows the following content for the corresponding investment:

- -investment ID
- -CIO rating
- -number of IT projects
- -total costs of all its projects
- -who was awarded the contract
- -the type of contract awarded
- -summary of the investment
- -any related URLs

Screen 10 show the following content for the corresponding agency:

- -agency code
- -a link to online help repository
- -a phone number for an information hotline
- -a mailing address
- -a Google Map showing the location of the agency

Connect the screens together by using a TabLayout and a ViewPager.

Task 9: Implement the Project Details Feature on Screens 8 and 9

Screens 8 and 9 are part of a ViewPager. When the Activity containing the ViewPager starts, use the passed in project ID and investment ID to query the *projects* and *investments* table to

populate the respective screen. Use a loader and query the database through the Content Provider.

Task 10: Implement the Agency Communication Channels Feature on Screen 10

Screen 10 is also part of a ViewPager. Similar to how Screens 8 and 9 are populated, use the passed in agency code to query the *agencies* table to populate Screen 10. This screen has a map which also needs to be configured.

Sign up for a API key and enable Google Maps API for the app. Configure AndroidManifest.xml to be able to use Google Maps for the app.

When Screen 10 starts, use the latitude and longitude coordinates for the agency to show a map at zoom level 14 with a marker indicating the location of the agency.

Task 11: Implement Google Analytics

Sign up for an Google Analytics account and get a tracking ID. Configure AndroidManifest.xml to be able to use Google Analytics.

Extend the Application object and create an instance of Analytics in it. Disable auto tracking and send an hit when the user views any of the screens with a chart (Screens 1 through 6).

Task 12: Implement the UI for Tablets and Make It Work

Create layout files shown in the app flow for tablets diagram. Modify the code so that the app works with the new and existing layouts. Lock the orientation to landscape for tablets (sw600dp).

Task 13: Implement the Widget

Build the UI for the widget. Implement the classes for the widget and allow the user to click on the widget to launch the app displaying Screen 3.