C868 – Software Capstone Project Summary

Task 2 – Section C

Capstone Proposal Project Name: Android App for Fuel and Oil Change Entry

Student Name: Leonard Lutz

Table of Contents

Application Design and Testing ***1***

Design Document ***1***

- ***Class Design*** ***1***
- ***UI Design*** ***2***
- ***Database Design*** ***6***

Unit Test Plan ***8***

- ***Introduction*** ***8***
  - ***Purpose*** ***8***
  - ***Overview*** ***8***
- ***Test Plan*** ***9***
  - ***Items*** ***9***
  - ***10***
  - ***Features*** ***12***
  - ***Deliverables*** ***12***
  - ***Tasks*** ***12***
  - ***Needs*** ***12***
  - ***Pass/Fail Criteria*** ***12***
- ***Specifications*** ***12***
- ***Procedures*** ***12***
- ***Results*** ***12***

C4. Source Code ***12***

C5. Link to Live Version ***13***

User Guide 13

Appendix A – Tests Performed 14

Duplicate office database on the phone 14

Test API Server Programs 14

Test Adding Fuel Fills 16

Test Adding Oil Changes 17

Appendix B – Sample Test Data 19

Fuel Fills for Vehicle V003 19

Fuel Fills for Vehicle V004 19

Oil Change for Vehicle V004 19

Application Design and Testing

Design Document

Class Design

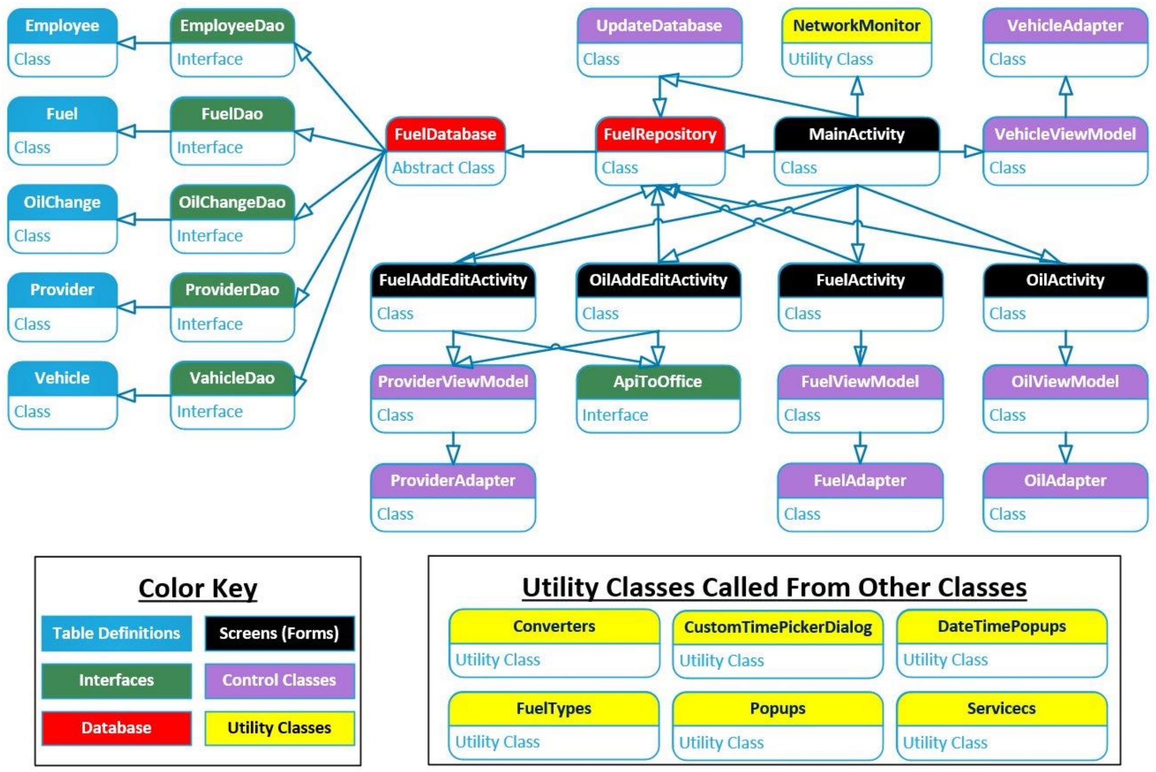


Figure 1: Classes and Their Relationships

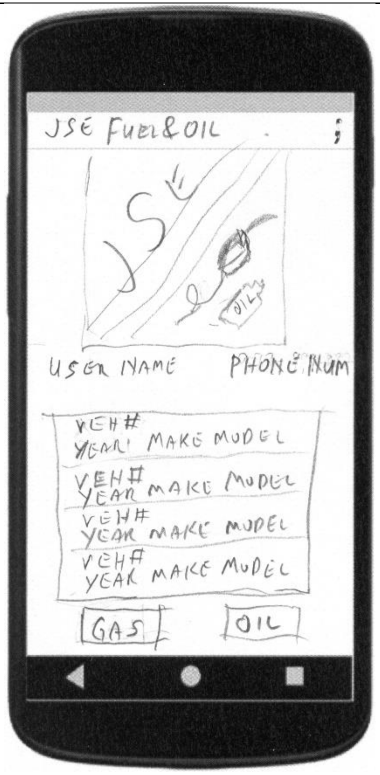
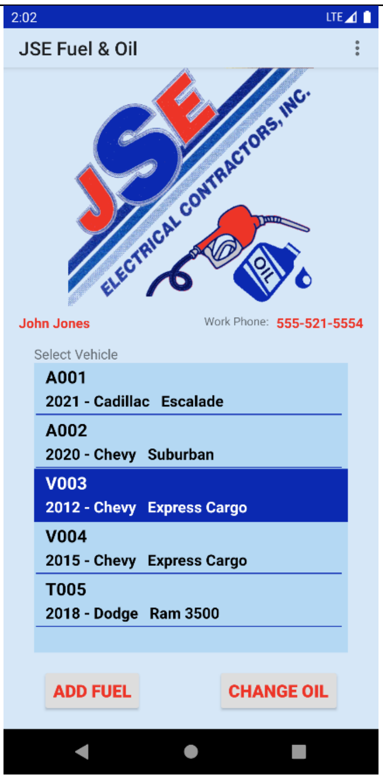
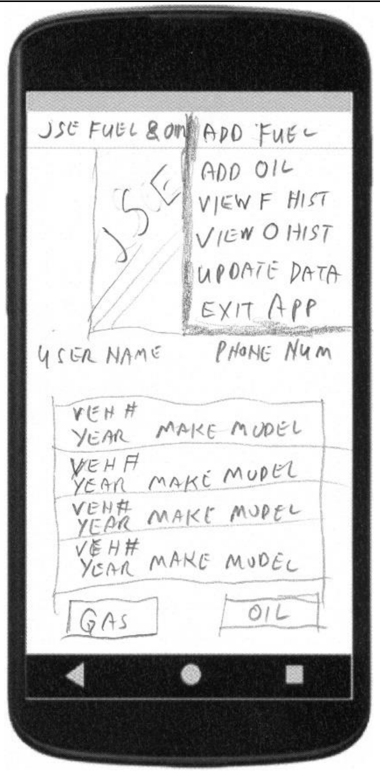
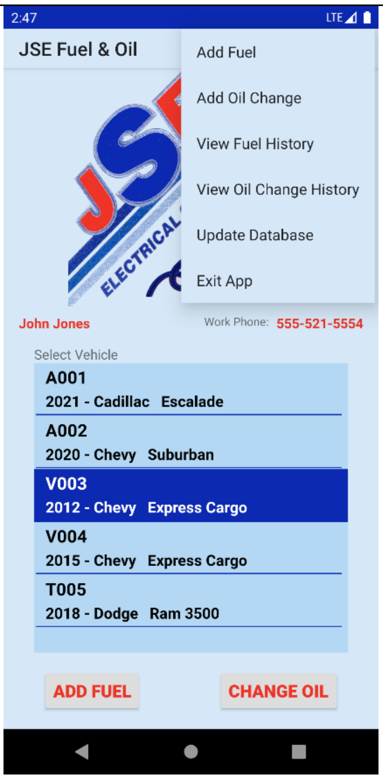
Figure 1 (shown above) indicates that, even though there are only five screens for this application (indicated with black headers), there are many additional classes required to perform all of the tasks that happen behind the scenes; 29 other classes, to be precise! Each of the *screens* require “ViewModels”, which each require “Adapters” (all indicated with purple headers) to display lists on their screens. In addition, each screen requires a connection to the “Repository” (red) which accesses the database (also red) through interfaces (green) which, in turn, require specific “Table Definition” classes (blue) that define how the information is recorded in the database stored on the phone. Screens that are used for inputting fuel fill-up and oil change information also require an interface to send/receive data over the internet to update the main database stored on the server at the company office.

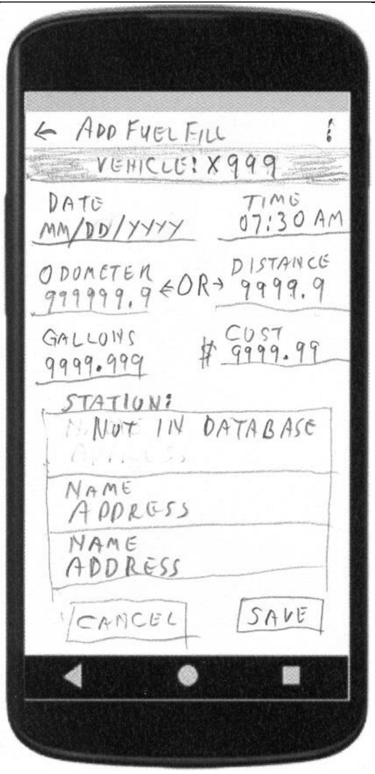
The “MainActivity” screen also uses the “UpdateDatabase” class to connect (through the repository) to the local database, as well as create a direct connection over the company’s WiFi to synchronize the phone’s database with the database on the company server.

The “Utility Classes” (yellow) store information and routines used to convert data from the way the application holds information (such as dates in memory getting stored as long integers in the database) and routines that standardize the way “pop-ups” are called for data manipulation or display. Other utility classes offer “type” definitions, such as “Services” and “FuelTypes” which provide the ability to determine which gas stations offer specific types of fuel, or which providers offer specific types of services.

UI Design

From the user’s perspective, this is a fairly simple application that allows the user to enter information that, in the past, was submitted with receipts and entered by office personnel.   The project could have been completed with only three screens, but two more were added to provide the user with reports that offer useful information for vehicle preventative maintenance. Descriptions for each screen, and their low fidelity (Concept) and high fidelity (Final Design) images, are included below.

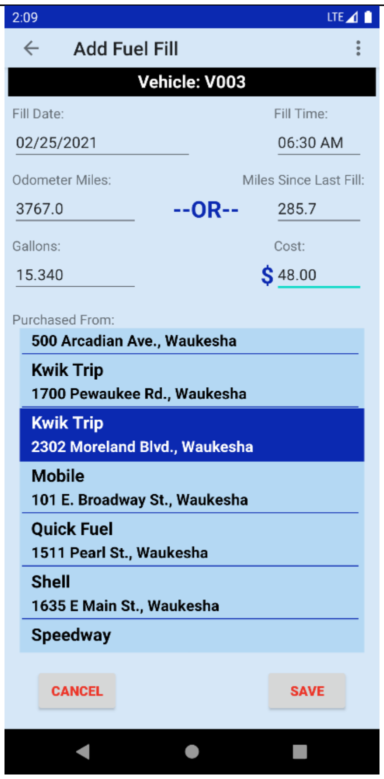
 JSE Fuel & OIL OIL USER NAME PHONE NUM VEH # YEAR MAKE MODEL VEH # YEAR MAKE MODEL VEH # YEAR MAKE MODEL VEH # YEAR MAKE MODEL GAS OIL Concept	**Main Screen** The main screen of the app will display the JSE Logo, the user's name, the phone number of the phone the app is running on, and the list of JSE owned vehicles. The user can select a vehicle from the list and choose to add fuel or record an oil change.	 2:02 JSE Fuel & Oil John Jones Work Phone: 555-521-5554 Select Vehicle A001 2021 - Cadillac Escalade A002 2020 - Chevy Suburban V003 2012 - Chevy Express Cargo V004 2015 - Chevy Express Cargo T005 2018 - Dodge Ram 3500 ADD FUEL CHANGE OIL Final Design
 JSE FUEL & OIL ADD FUEL ADD OIL VIEW F HIST VIEW O HIST UPDATE DATA EXIT APP USER NAME PHONE NUM VEH # YEAR MAKE MODEL VEH # YEAR MAKE MODEL VEH # YEAR MAKE MODEL VEH # YEAR MAKE MODEL GAS OIL Concept	**Main Screen (Showing drop-down Menu)** The main screen's drop-down menu offers an alternate method to add fuel or change oil, and also provides additional choices to view fuel fill and oil change histories, update the database, or exit the application.	 2:47 JSE Fuel & Oil Add Fuel Add Oil Change View Fuel History View Oil Change History Update Database Exit App John Jones Work Phone: 555-521-5554 Select Vehicle A001 2021 - Cadillac Escalade A002 2020 - Chevy Suburban V003 2012 - Chevy Express Cargo V004 2015 - Chevy Express Cargo T005 2018 - Dodge Ram 3500 ADD FUEL CHANGE OIL Final Design



Concept

Add Fuel Screen

This screen allows the user to enter all required data for a fuel fill-up, including choosing the gas station from a list. Information in the database restricts the list to show only those providers that actually sell that type of fuel. If the vehicle uses diesel or bio fuel, the list will not show providers that do not offer that type of fuel.



2:09 LTE

Add Fuel Fill

Vehicle: V003

Fill Date: 02/25/2021 Fill Time: 06:30 AM

Odometer Miles: 3767.0 --OR-- Miles Since Last Fill: 285.7

Gallons: 15.340 Cost: $ 48.00

Purchased From:

500 Arcadian Ave., Waukesha

Kwik Trip
1700 Pewaukee Rd., Waukesha

Kwik Trip
2302 Moreland Blvd., Waukesha

Mobile
101 E. Broadway St., Waukesha

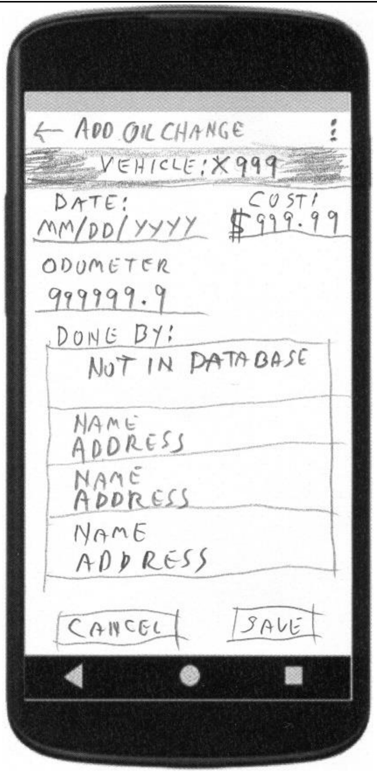
Quick Fuel
1511 Pearl St., Waukesha

Shell
1635 E Main St., Waukesha

Speedway

CANCEL SAVE

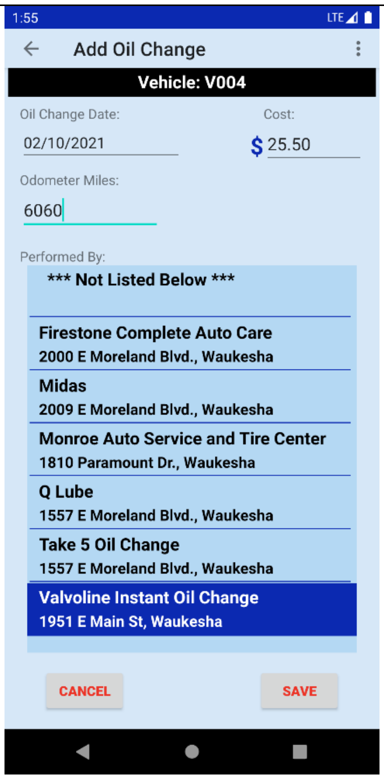
Final Design



Concept

Add Oil Change Screen

This screen allows the user to enter all required information for an oil change, including choosing the provider from a list. Again, this list will only include providers that offer oil changes as one of their services.



1:55 LTE

Add Oil Change

Vehicle: V004

Oil Change Date: 02/10/2021 Cost: $ 25.50

Odometer Miles: 6060

Performed By:

*** Not Listed Below ***

Firestone Complete Auto Care
2000 E Moreland Blvd., Waukesha

Midas
2009 E Moreland Blvd., Waukesha

Monroe Auto Service and Tire Center
1810 Paramount Dr., Waukesha

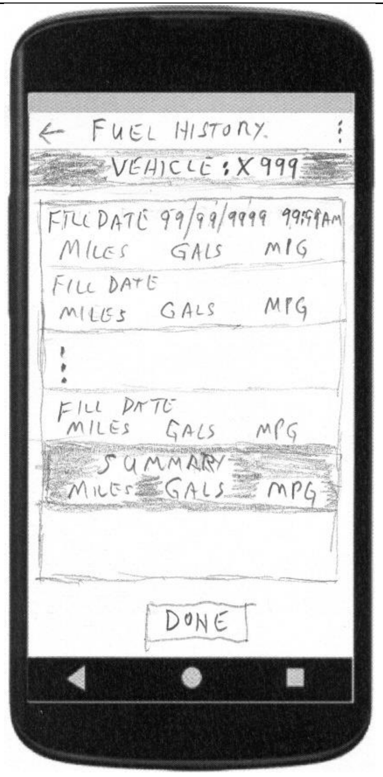
Q Lube
1557 E Moreland Blvd., Waukesha

Take 5 Oil Change
1557 E Moreland Blvd., Waukesha

Valvoline Instant Oil Change
1951 E Main St, Waukesha

CANCEL SAVE

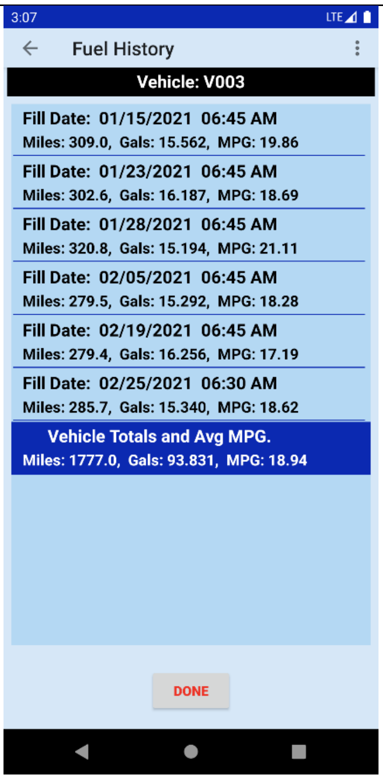
Final Design



Concept

Fuel History Screen

This screen provides a report of all recorded fill-ups for the selected vehicle including a summary of miles travelled, fuel consumed, and the average MPG for this vehicle.


3:07

Fuel History

Vehicle: V003

Fill Date: 01/15/2021 06:45 AM
Miles: 309.0, Gals: 15.562, MPG: 19.86

Fill Date: 01/23/2021 06:45 AM
Miles: 302.6, Gals: 16.187, MPG: 18.69

Fill Date: 01/28/2021 06:45 AM
Miles: 320.8, Gals: 15.194, MPG: 21.11

Fill Date: 02/05/2021 06:45 AM
Miles: 279.5, Gals: 15.292, MPG: 18.28

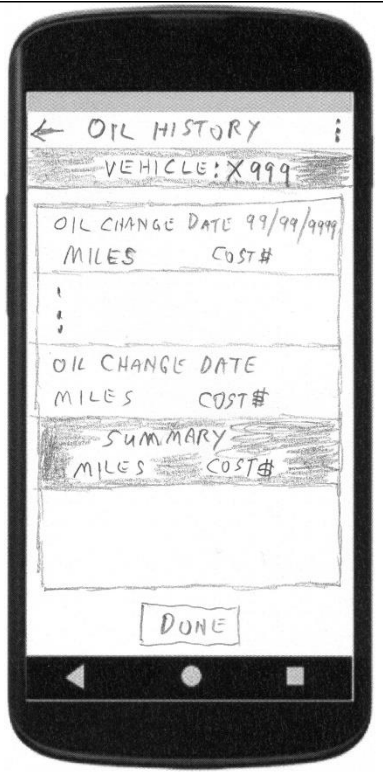
Fill Date: 02/19/2021 06:45 AM
Miles: 279.4, Gals: 16.256, MPG: 17.19

Fill Date: 02/25/2021 06:30 AM
Miles: 285.7, Gals: 15.340, MPG: 18.62

Vehicle Totals and Avg MPG.
Miles: 1777.0, Gals: 93.831, MPG: 18.94

DONE

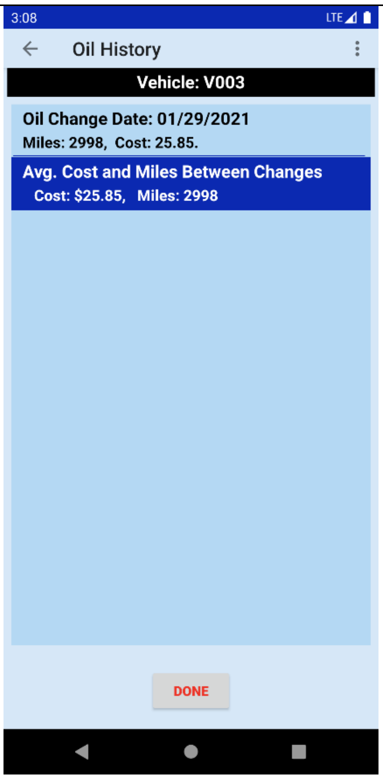
Final Design



Concept

Oil History Screen

This screen provides a report of all recorded oil changes for the selected vehicle and a summary showing the average cost for all of its oil changes and average miles travelled between oil changes.


3:08

Oil History

Vehicle: V003

Oil Change Date: 01/29/2021
Miles: 2998, Cost: 25.85.

Avg. Cost and Miles Between Changes
Cost: $25.85, Miles: 2998

DONE

Final Design

Database Design

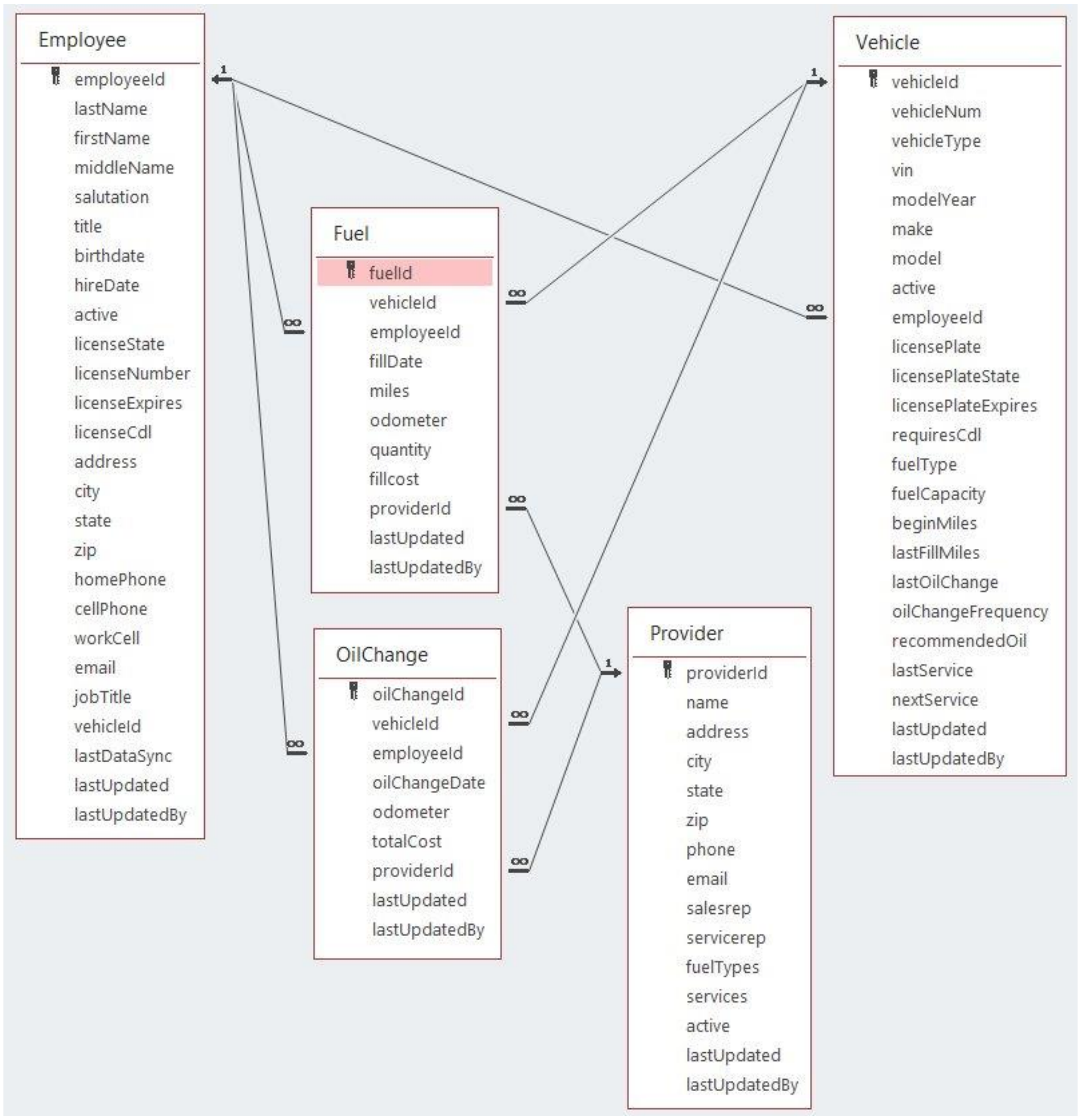


Figure 2: Entity Relationship Diagram for the Company's MySQL Database

Figure 2 (above) shows the related database tables as stored on the company server. For security, and to save limited storage space on the phone, certain fields in the Employee, Vehicle, and Provider tables do not transfer out of the office. Figure 3 (below) shows the information that actually gets stored on the phone.

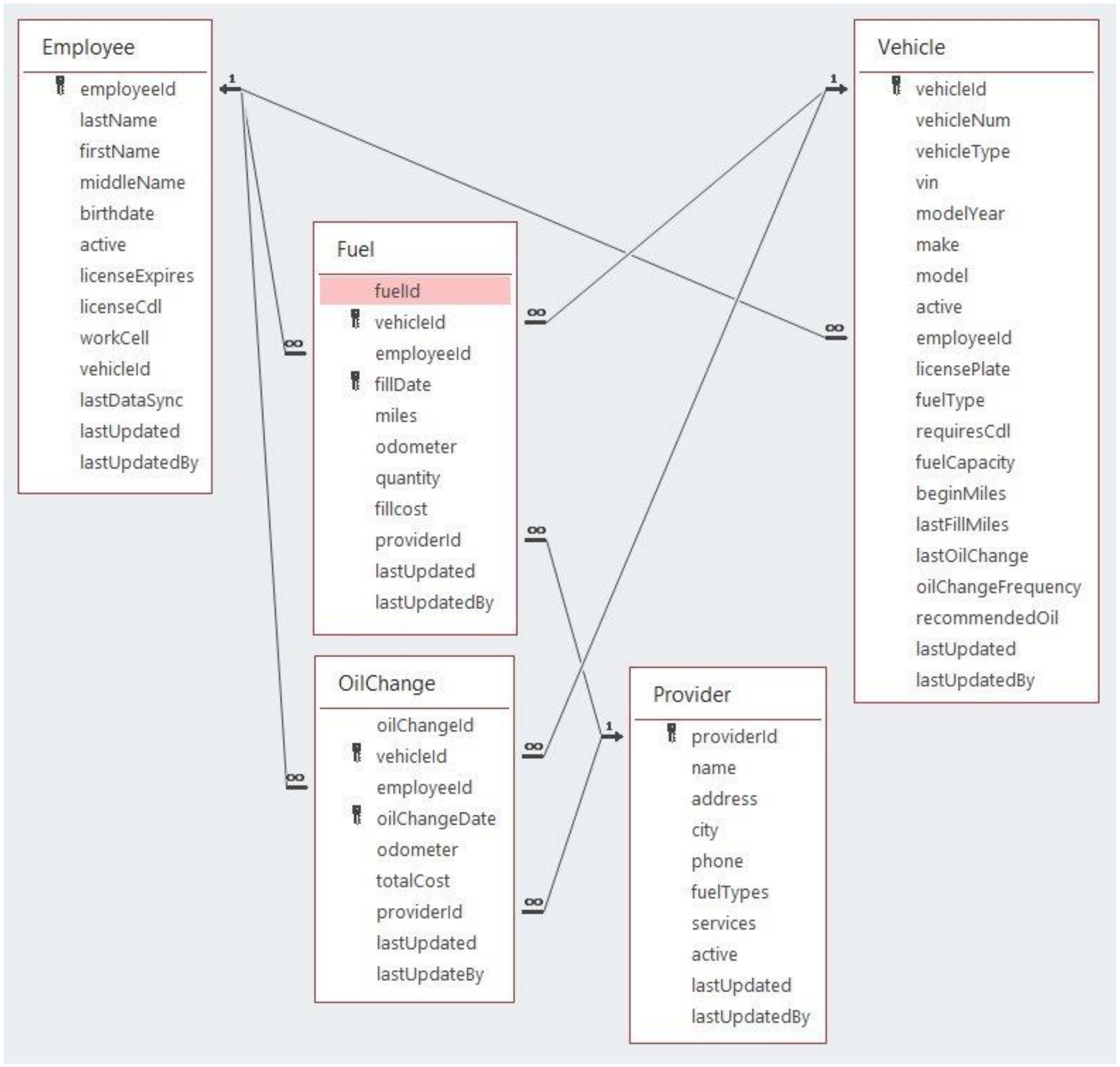


Figure 3: Entity Relationship Diagram for the Phone's SQLite Database

The other differences between the two databases involve the Fuel and OilChange tables. First, these two files are duplicated in their entirety. Second, in the MySQL database in the office, the primary keys are the fuelId and oilChangeId,, while on the phones' SQLite database, the primary key is the combination of the vehicleId and fillDate/oilChangDate fields. This offers the ability to maintain data integrity by preventing duplicate entries, while making sure that only the office database can assign database table Id's. It also allows the phone to store a zero in the

Id field which is used to indicate which entries have not yet been sent to the office. Once the entry is sent to the office, the office assigns the Id to the entry and transmits it back to the phone where the application can update it in the phone's database. This is required because many Fuel and OilChange entries can be updated by many different users every day, and all entries from all vehicles are stored in the same tables in the databases.

Unit Test Plan

Introduction

Purpose

This application requires access to two distinct databases. A secure MySQL database located on the company's office server, which is only accessible when the user is connected to the office's Local Area Network[1], and a limited SQLite database stored locally on the user's phone to allow access to information when outside the office. The main obstacles to this solution are the methods required to access each database, the differences in the way data are stored in each of them, and the way information is transferred from one to the other. This requires complex manual testing, using multiple external utilities, to validate that the application is transferring, synchronizing, and adding data correctly to both of the databases.

Overview

Differences in the information that is stored in each database, as shown in Figures 2 and 3 above, as well as the way data is stored in each (see Table 1 below) requires extensive testing, using multiple methods, to be sure that data is being sent, received, and stored correctly.

[1] Because JSE no longer exists, and because people testing this application require access to a working database, a sample database has been installed on a website available to the author and has been made accessible over the internet.

Data Type	**Storage Format**		**Differences**	
	Office MySQL	**Phone SQLite**	**MySQL**	**SQLite**
Date/Time	String	Long Integer	String is easy to read when viewing database with MySQL Workbench	Date/Time stored as number of seconds since 1970-01-01 00:00:00 UTC
Numeric with Decimals	Decimal	Floating Point	Precise number with specific number of digits and decimal places.	Imprecise number which must be rounded to specific decimal place for proper display..

Table 1: Database differences

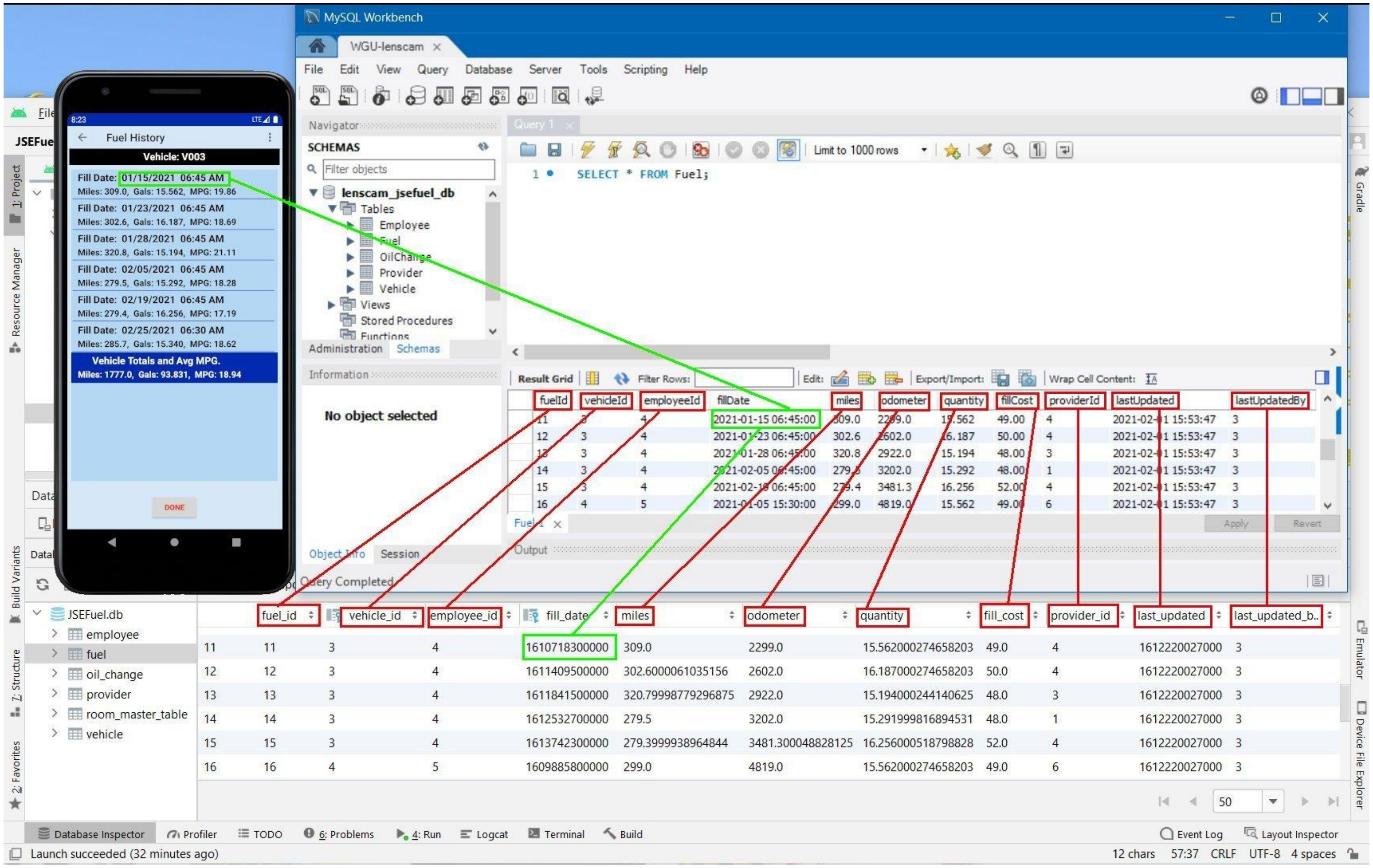
Additionally, the methods used to transfer data differ, depending on whether the data is being retrieved from the office database to update the phone, or the data is being added on the phone and transmitted to the office. Updating the phone is done through a secure connection using JDBC over the office LAN. Updating the office is accomplished by sending requests over the internet to the company's web server via HTTP and using utilities written in PHP to securely update the remote database.

Test Plan

Items

There are 4 different items required to verify the information in both databases:

1. Android Studio – not only for creating the application, but to use its "Database Inspector" to view the contents of the phone's database.
2. A phone to run the application (an emulator will suffice).
3. "MySQL Workbench" to view the contents of the office database.
4. A utility able to send sample database entries and receive responses to test the API that is used to send Fuel and Oil Change entries to the office (I used Postman, available from https://www.postman.com/downloads/)



Fuel History

Vehicle: V003

Fill Date: 01/15/2021 06:45 AM
Miles: 309.0, Gals: 15.562, MPG: 19.86

Fill Date: 01/23/2021 06:45 AM
Miles: 302.6, Gals: 16.187, MPG: 18.69

Fill Date: 01/28/2021 06:45 AM
Miles: 320.8, Gals: 15.194, MPG: 21.11

Fill Date: 02/05/2021 06:45 AM
Miles: 279.5, Gals: 15.292, MPG: 18.28

Fill Date: 02/19/2021 06:45 AM
Miles: 279.4, Gals: 16.256, MPG: 17.19

Fill Date: 02/25/2021 06:30 AM
Miles: 285.7, Gals: 15.340, MPG: 18.62

Vehicle Totals and Avg MPG.
Miles: 1777.0, Gals: 93.831, MPG: 18.94

```
1 ● SELECT * FROM Fuel;
```

fuelId	vehicleId	employeeId	fillDate	miles	odometer	quantity	fillCost	providerId	lastUpdated	lastUpdatedBy
11	3	4	2021-01-15 06:45:00	309.0	2299.0	15.562	49.00	4	2021-02-01 15:53:47	3
12	3	4	2021-01-23 06:45:00	302.6	2602.0	16.187	50.00	4	2021-02-01 15:53:47	3
13	3	4	2021-01-28 06:45:00	320.8	2922.0	15.194	48.00	3	2021-02-01 15:53:47	3
14	3	4	2021-02-05 06:45:00	279.5	3202.0	15.292	48.00	1	2021-02-01 15:53:47	3
15	3	4	2021-02-19 06:45:00	279.4	3481.3	16.256	52.00	4	2021-02-01 15:53:47	3
16	4	5	2021-01-05 15:30:00	299.0	4819.0	15.562	49.00	6	2021-02-01 15:53:47	3

No object selected

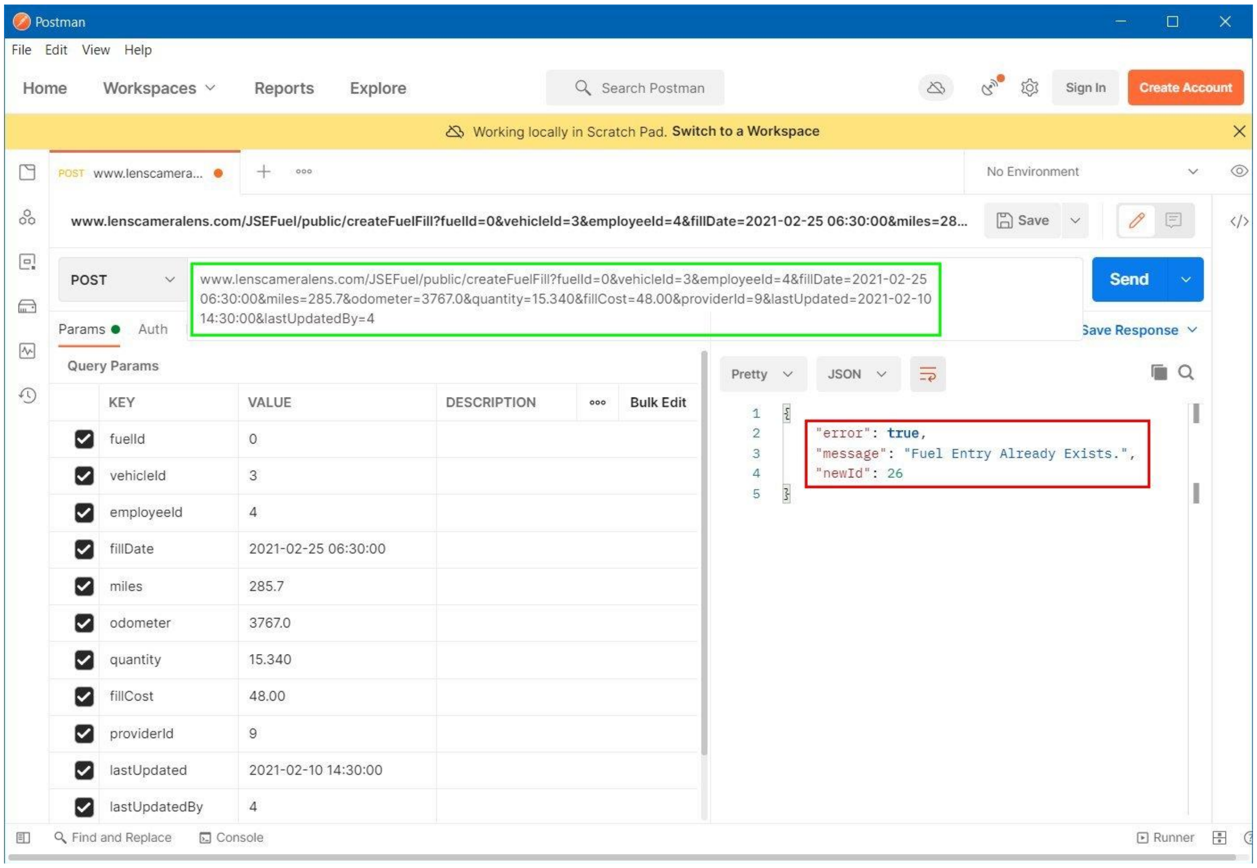
Query Completed

	fuel_id	vehicle_id	employee_id	fill_date	miles	odometer	quantity	fill_cost	provider_id	last_updated	last_updated_b..
11	11	3	4	1610718300000	309.0	2299.0	15.562000274658203	49.0	4	1612220027000	3
12	12	3	4	1611409500000	302.6000061035156	2602.0	16.187000274658203	50.0	4	1612220027000	3
13	13	3	4	1611841500000	320.79998779296875	2922.0	15.194000244140625	48.0	3	1612220027000	3
14	14	3	4	1612532700000	279.5	3202.0	15.291999816894531	48.0	1	1612220027000	3
15	15	3	4	1613742300000	279.3999938964844	3481.300048828125	16.256000518798828	52.0	4	1612220027000	3
16	16	4	5	1609885800000	299.0	4819.0	15.562000274658203	49.0	6	1612220027000	3

Launch succeeded (32 minutes ago)

Figure 4: Screen Capture showing 3 of the 4 required utilities:

(upper left) Phone Emulator, (upper right) MySQL Workbench, (bottom) Android Studio showing Database Inspector



www.lenscameralens.com/JSEFuel/public/createFuelFill?fuelId=0&vehicleId=3&employeeId=4&fillDate=2021-02-25 06:30:00&miles=28...

POST

```
www.lenscameralens.com/JSEFuel/public/createFuelFill?fuelId=0&vehicleId=3&employeeId=4&fillDate=2021-02-25
06:30:00&miles=285.7&odometer=3767.0&quantity=15.340&fillCost=48.00&providerId=9&lastUpdated=2021-02-10
14:30:00&lastUpdatedBy=4
```

Query Params

	KEY	VALUE	DESCRIPTION
✓	fuelId	0	
✓	vehicleId	3	
✓	employeeId	4	
✓	fillDate	2021-02-25 06:30:00	
✓	miles	285.7	
✓	odometer	3767.0	
✓	quantity	15.340	
✓	fillCost	48.00	
✓	providerId	9	
✓	lastUpdated	2021-02-10 14:30:00	
✓	lastUpdatedBy	4	

```
{
    "error": true,
    "message": "Fuel Entry Already Exists.",
    "newId": 26
}
```

Figure 5: Screen Capture showing the 4th required utility, Postman

In Figure 4, the red boxes and the lines between them indicate the fields that need to be compared to verify that the data matches. The green links show the date fields. The date fields are stored as readable strings in the MySQL database in the office, while the phone stores them as long integers in SQLite. Bringing up the “Fuel History” screen on the phone allows us to view the information on the phone as a readable date and time. However, MySQL displays the time in a 24-hour format while the phone app will display the time in 12-hour AM/PM style.

Figure 5, above, shows the “Postman” utility with entries for each field to be sent to the office. In the screen capture provided, the request to add a new fuel fill has already been sent. The text in the green box shows how the information is transmitted to the server, and the information in the red box shows the response received. In this case, the information would have created a duplicate entry (which is not allowed) and provides us with the “fuelId” of the matching record. All responses are checked programmatically to ensure data integrity is maintained.

Features

Please see Appendix A for features of each test performed.

Deliverables

A document describing the test plan and the results of each test completed.

Tasks

Please see Appendix A for tasks required for each test performed.

Needs

The tests require the four utilities described previously in “Items”, as well as a connection to the office database server.

Pass/Fail Criteria

Please see Appendix A for criteria for each test performed.

Specifications

Please see Figures 4 and 5 above for sample screen captures of tests performed.

Procedures

Please see Appendix A for procedures to follow for each test.

Results

Please see Appendix A for expected and actual results for each test, and Figures 4 and 5 for sample screen captures of results.

C4. Source Code

All source code for this application can be found in a compressed file containing the entire Android Studio project. The source code for the PHP files that make up the API used on the server can be found in the “Resources” section of the Android Studio project.in the “API Files” folder. The files used to build and populate the sample MySQL database are also in the “Resources” section in the folder named “SQL Build Files”.

The compressed file’s name is: **JSEFuel.zip**.

C5. Link to Live Version

This project is not a “Website”, so there is not a “Live Version” to be linked to. However, the application does contain a link to “Live Data”, in the form of a database connection to a working, sample, MySQL database. Also, the installation file that can be compiled with the supplied source code has been uploaded to a Website owned by the author, and can be downloaded to a phone (or a phone emulator) at:

http://lenscameralens.com/download/JSEFuel.apk

User Guide

The user guide for this application can be found in a separate Adobe Acrobat file.

The file’s name is: **JSEFuel-UserGuide.pdf**

Appendix A – Tests Performed

Duplicate office database on the phone

Use the application to copy required information from the office database to the phone (completed when user presses the button to “fetch database” during first run of application, or anytime the user presses “Update Database” from main screen’s drop-down menu). For testing, the office database can be changed by using another phone or emulator to save fuel fillups or oil changes. Use the application, MySQL Workbench, and Android Studio’s Database Inspector to compare results. Fix any issues found and repeat testing until all tests successfully pass.

Test Case	Description	Method	Expected Results	Actual Results	Pass/Fail
1	Compare Employee Table	Manual	All fields in all records match	All fields in all records match	Pass
2	Compare Vehicle Table	Manual	All fields in all records match	All fields in all records match	Pass
3	Compare Provider Table	Manual	All fields in all records match	All fields in all records match	Pass
4	Compare Fuel Table	Manual	All fields in all records match	All fields in all records match	Pass
5	Compare OilChange Table	Manual	All fields in all records match	All fields in all records match	Pass

Test API Server Programs

Use Postman to send requests to the server and view its responses. The application will validate the information before attempting to send requests and all fields will be included in the request, so we should run these Postman tests with valid data. However, if changes are ever made to the database structure, additional or different fields may be required, so we should test with missing fields to make sure the API recognizes if any are not sent. Fix any issues found and repeat testing until all tests successfully pass.

Test Case	Description	Method	Expected Results	Actual Results	Pass / Fail
1	Try adding a fuel fill with missing parameter(s).	Manual	error = true message = list of missing parameters	error = true message = list of missing parameters	Pass
2	Try adding a fuel fill with all parameters	Manual	error = false message = “Fuel Entry Created Successfully.”	error = false message = “Fuel Entry Created Successfully.”	Pass
3	Try adding the same fuel fill again	Manual	error = true message = “Fuel Entry Already Exists” newId = fuelId of existing entry	error = true message = “Fuel Entry Already Exists” newId = fuelId of existing entry	Pass
4	Try adding a fuel fill with all parameters	Manual	error = false message = “Fuel Entry Created Successfully.”	error = true message = “Unspecified Error Occurred.” This can occur if there is a connection or unknown database error. Test should be retried until the error no longer occurs.	Fail
5	Try adding an oil change with missing parameter(s).	Manual	error = true Message = (list of missing parameters)	error = true message = (list of missing parameters)	Pass
6	Try adding an oil change with all parameters	Manual	error = false message = “Oil Change Entry Created Successfully.”	error = false message = “Oil Change Entry Created Successfully.”	Pass
7	Try adding the same oil change again	Manual	error = true message = “Oil Change Entry Already Exists” newId = oilChangeId of existing entry	error = true message = “Oil Change Entry Already Exists” newId = oilChangeId of existing entry	Pass

Test Case	Description	Method	Expected Results	Actual Results	Pass / Fail
8	Try adding an oil change with all parameters	Manual	error = false message = "Oil Change Entry Created Successfully."	error = true message = "Unspecified Error Occurred." This can occur if there is a connection or unknown database error. Test should be retried until the error no longer occurs.	Fail

Test Adding Fuel Fills

Use the application's "Add Fuel" screen to send fuel fill data to server. Use the application, MySQL Workbench, and Android Studio's Database Inspector to compare results. Fix any issues found and repeat testing until all tests pass.

Test Case	Description	Method	Expected Results	Actual Results	Pass / Fail
1	Fill in the fuel fill screen with some entries blank and press "SAVE" button	Manual	User notified of missing information and requested to fill in missing info. User returned to fuel fill screen until all fields have been entered	User notified of missing information and requested to fill in missing info. User returned to fuel fill screen until all fields have been entered	Pass
2	Fill in all entries on the fuel fill screen and press "SAVE" button while connected to the internet	Manual	New matching entries in both databases, user returned to main screen	New matching entries in both databases, user returned to main screen	Pass

Test Case	Description	Method	Expected Results	Actual Results	Pass / Fail
3	Fill in all entries on fuel fill screen and press "SAVE" button while NOT connected to the internet (turn on airplane mode)	Manual	New entry in phone database with "0" in fuelId field, nothing changed in office database, user returned to main screen	New entry in phone database with "0" in fuelId field, nothing changed in office database, user returned to main screen	Pass
4	After successful entry while NOT connected to the internet, check databases after the internet connection has been re-established (turn off airplane mode)	Manual	All fields in all fuel records match in both databases, there are no entries in phone database with fuelId of "0"	All fields in all fuel records match in both databases, there are no entries in phone database with fuelId of "0"	Pass

Test Adding Oil Changes

Use the application's "Add Oil Change" screen to send Oil Changes to the server. Use the application, MySQL Workbench, and Android Studio's Database Inspector to compare results. Fix any issues found and repeat testing until all tests successfully pass.

Test Case	Description	Method	Expected Results	Actual Results	Pass / Fail
1	Fill in the oil change screen with some entries blank and press "SAVE" button (use airplane mode)	Manual	User notified of missing information and requested to fill in missing info. User returned to oil change screen until all fields have been entered	User notified of missing information and requested to fill in missing info. User returned to oil change screen until all fields have been entered	Pass
2	Fill in all entries on the oil change screen and press "SAVE" button while connected to the internet	Manual	New matching entries in both databases, user returned to main screen	New matching entries in both databases, user returned to main screen	Pass

Test Case	Description	Method	Expected Results	Actual Results	Pass / Fail
3	Fill in all entries on oil change screen and press "SAVE" button while NOT connected to the internet (turn on airplane mode)	Manual	New entry in phone database with "0" in oilChangeId field, nothing changed in office database, user returned to main screen	New entry in phone database with "0" in oilChangeId field, nothing changed in office database, user returned to main screen	Pass
4	After successful entry while NOT connected to the internet, check databases after the internet connection has been re-established (turn off airplane mode)	Manual	All fields in all fuel records match in both databases, there are no entries in phone database with oilChangeId of "0"	All fields in all fuel records match in both databases, there are no entries in phone database with oilChangeId of "0"	Pass

Appendix B – Sample Test Data

The tables below offer additional sample data for testing the application. Please don't forget to pick a provider from the list on the appropriate screen.

Fuel Fills for Vehicle V003

Date	Time	Odometer	Miles	Gallons	Cost
Feb 25, 2021	6:30 AM	3767.0	285.7	15.340	48.00
Mar 3, 2021	6:45 AM	4081.7	314.7	15.966	49.00
Mar 10, 2021	6:15 AM	4386.3	304.6	16.159	51.01
Mar 15, 2021	6:30 AM	4729.7	343.4	16.568	54.00
Mar 19,2021	6:30 AM	5065.9	336.2	16.964	56.00

Fuel Fills for Vehicle V004

Date	Time	Odometer	Miles	Gallons	Cost
Feb 16, 2021	4:15 PM	6284.2	286.9	16.780	62.00
Feb 24, 2021	3:45 PM	6588.8	304.6	16.159	51.00
Mar 3, 2021	4:15 PM	6886.9	298.1	15.005	57.00
Mar 8, 2021	4:30 PM	7189.9	303.0	16.845	65.00

Oil Change for Vehicle V004

Date	Cost	Odometer
Feb 10, 2021	25.50	6060