

222145_Assignment2_Combine _Assginment

by Keyur Pandya

Submission date: 11-Dec-2022 11:30PM (UTC+0000)

Submission ID: 191916367

File name: 42739_Keyur_Pandya_222145_Assignment2_Combine_Assginment_472250_1752431927.docx
(1.47M)

Word count: 2671

Character count: 12697

Individual Reflection Report

COMBINED ASSIGNMENT FOR SOFTWARE DEV AND
TESTING/AGILE SEP/DATABASE SYSTEMS

KEYUR PANDYA (2222145)

Table of Contents

Introduction: 2

Evaluation 2

Contribution 5

Database Task..... 8

Peer Evaluation 8

Conclusion 8

Demonstration Error! Bookmark not defined.

Introduction:

Our task was to develop a console application for a car rental company using Java. The application allows users register, login, browse cars, and book. I was part of Team 1 and in this report, I will discuss our approach and my interpretation of it. We had five days (four and a half) to finish this project, so time was scarce. We decided to focus on delivering a working MVP rather than trying to meet all requirements. The MVP should deliver a working model that lets users create accounts, log in, and book cars, and we planned to focus on administration in later releases. This report covers agile practises, database design, and team application development.

Evaluation

Following the receipt of the assignment brief on midday of day 1, our group discussed the database design planning by the requirements provided for this assignment. Everyone presented what they had accomplished up to this point, and we all had something ready to show before we even began. In the table I created, I identified the entities, after which I eliminated redundant standard fields and functional dependencies. Next, I eliminated past dependencies to transform the table into a second normalisation form. Finally, I removed transitive dependencies and corrected the relationships between the entities.

The tables created by everyone were formatted using the 3NF process, and they were nearly identical for all of us. I had not made audit tables as I was occupied with the DVLA project. However, Arthur on our team did, and he also populated data using Mockaroo, which put him a step ahead of the rest of the group. As a result, everyone on the team agreed that Arthur's work should be utilised, which would help us save time. However, it did need alterations. We all looked at the database and then discussed the necessary changes and updates we needed to make to his existing work.

First, we all got together and decided on some naming conventions to use. The format we used had a "_" in between each pair of letters. A good example would be "first_name" or "pickup_location." After we had finished correctly naming the fields, we had a group discussion about correctly limiting the length of fields, such as valid UK registration plate length is a maximum of 7 characters. Since our application was intended for customers based in the UK, we limited the maximum size of the country field to 16 characters, with Northern Ireland serving as the most extended option. We considered the longest model names, such as "Lamborghini Aventador LP 750-4 Superveloce Roadster" and the longest manufacturer name "Mercedes-Benz Aktiengesellschaft", to appropriately limit the maximum length of fields.

In addition to this, we made sure that we were using the appropriate data types, such as making the daily fees data type numeric rather than int. In the daily fee field value, we included constraints such as only allowing two digits to show after the decimal point. Other restrictions, such as the minimum length of the username having to be greater than or equal to six and the password having to be greater than or equal to eight, were also discussed and

added. In addition to this, we added a constraint to ensure that the date of the car's return should occur after the date on which it was picked up.

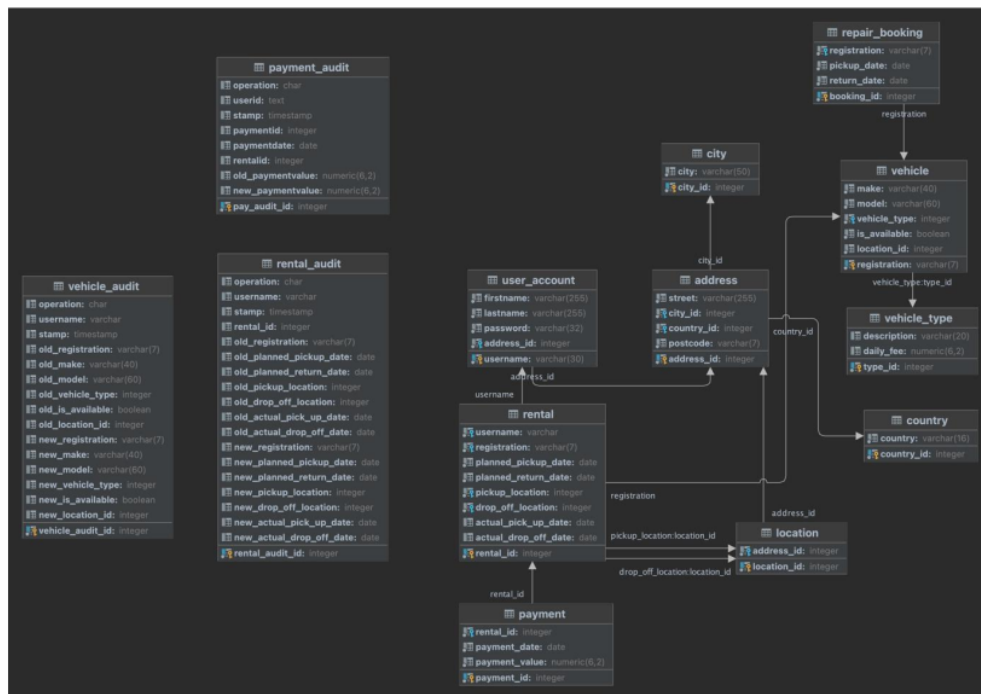


Figure 1 Database Design for Car Rental Company

After that, we made sequences for our primary keys while ensuring they did not deviate from their default values. We then populated the from Mockaroo, where Arthur had already created functions to mock appropriate values. Mockaroo is a software which helps us settle mock data.

On the second day, we began our session of sprint refinement. In this session, we created user activities based on the requirements. Using the user activities, we could break down the tasks the user would perform during interaction with the application. We then started jotting tasks; each team member participated and created tasks and user stories. One of the peers started arranging a job based on their choice without much discussion because the peer felt it was right. We only had a little time, so everyone ignored it and started working on user stories instead. Everyone contributed to writing user stories. We then discussed and planned out which stories to include in the first release for MVP, and the rest can be moved to future releases.

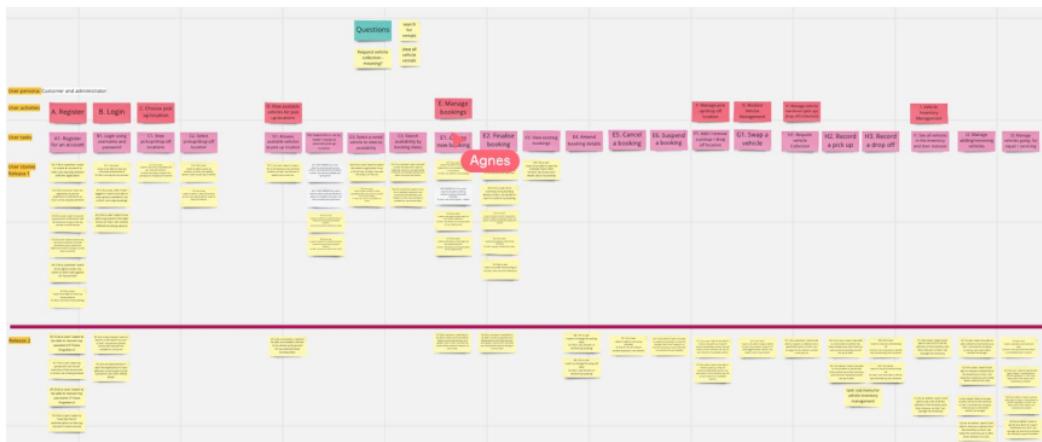


Figure 2 Our user story board on Miro. Link to board - <https://miro.com/app/board/uXjVP8q41Qk=/>

After we had completed and finalised the user stories for release one, we assigned indexes to each of them to quickly identify them. After that, we utilised the planning poker tool on Miro to give complexity points to each of our stories. The user stories were added to the Jira board, a tool for agile project management, by Agnes and me at the end of the day after the story points had been assigned. We were populating the Jira board as Elanwy created a GitHub repository to host our code, and Arthur was making amendments to push his SQL script on GitHub. By the end of day 2, we had the Jira board, the GitHub repository, and the SQL scripts to create the database ready.

After the meeting, Elanwy asked if she could be given the responsibility of completing the A and B parts of the user activities (refer to user storyboard, and its indexes), which are registering and logging in, because she had previously spent some of her free time working on the user login page. As a result, everyone felt a sense of urgency to pre-take the user stories to ensure that we do not miss out on anything and ultimately have fewer complexity points. Agnes oversaw the user activities section C, Arthur oversaw D, and I oversaw E. I am aware that a lack of time and the assignment criteria of a feeling that sufficient work must be done contributed to the decision.

In my opinion, it somewhat contradicts agile, the purpose of agile and using Kanban board is that we could see who is working on which stories and people can take more user stories based on the completion of their report on which they are currently working. In this instance, some tasks were more straightforward than expected, and as a result, some individuals had sufficient time to write detailed comments and perform some testing.

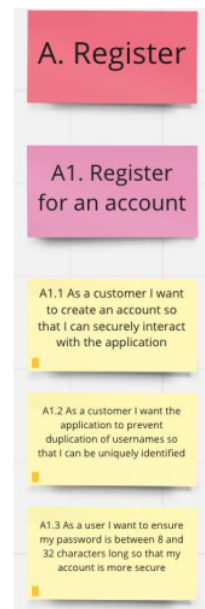


Figure 3 Including indexes in user stories so we can easily track them.

Contribution

I was responsible for completing the release one of E section (manage bookings) where E2.2 was going to be done by Agnes as she felt like she might need more complexity points. The user story E3.1 was not a part of first release, but I was able to pick it up as it was like showcasing the booking reference ID after checkout.

After posting all the stories to the Jira board, it indicated that Sprint 1's total complexity points were 202. I was tasked with completing 52 of those points, but I added one more feature from the backlog later in that sprint, bringing the total velocity up to 207 points (Jira oddly reports 217 points), of which I completed 57 [story points](#). The java code is on GitHub repo under the folder KP_Finalise_Booking and the sql code can be found in Posgresql folder insider KP_Finalise_Booking. I will now talk about some of the stories I have worked on in next section.

One of my user stories, [E2.0](#), demanded that the user see a summary of payment for the dates they chose in E1.4 to E1.5, which include the dates for picking up and returning the vehicle. Then, it was my responsibility to use them to calculate the total time the user wanted to reserve the vehicle. I could calculate the total booking fee and display that summary to the user once I knew the total duration. As shown in Figure 5, I divided the story into smaller tasks on Jira.

Once I have the total duration and the vehicle the user has chosen to reserve from user story E1.1, I needed to get the value of that registration ID by creating a global variable in that story that I need to use to find the chosen vehicle type of that vehicle in user story E2.0. This property can be used to determine the daily fee for the chosen vehicle after the vehicle type has been accessed from the database. I used the daily rate and total duration to calculate the user's total booking fee based on the chosen vehicle and booking dates.

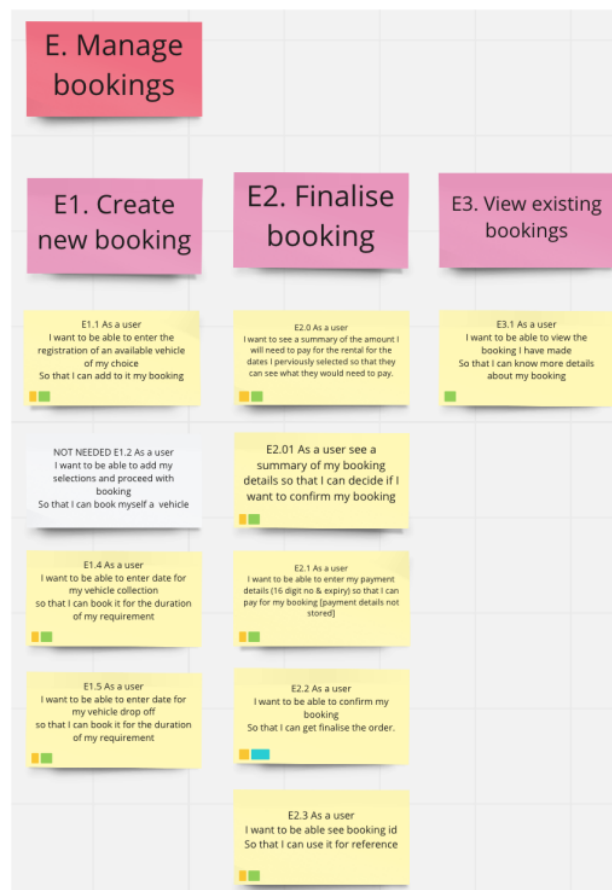


Figure 4 Section E, the part assigned to me (all the stories in image above were my responsibility apart from E2.2 with light blue colour which was done by Agnes).

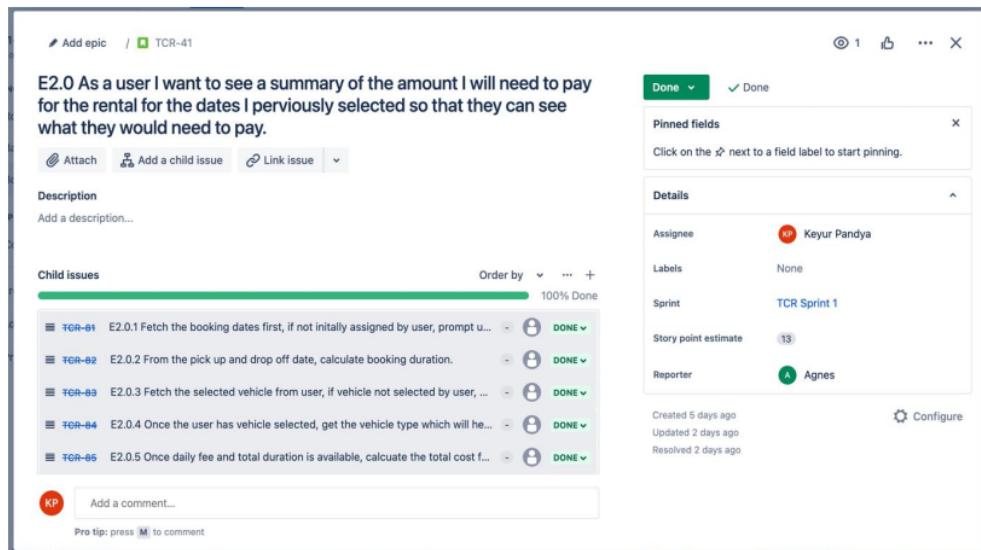


Figure 5 Story E2.0 on Jira board broken into sub tasks.

However, the story became more complicated because one of my peers decided to alter the flow that users should be able to use the application's search functionality or menu to move on to booking. Because of this, I had to change every line of code for user stories E1.1–1.5. However, E1.1 wasn't too bad because I modified Arthur's code to validate the licence plate.

However, I had already developed a programme that validates dates for user stories E1.4–E.15 by making sure they are not entered earlier than the current date. The user had the option to re-enter the dates rather than having the programme crash when the user entered the dates incorrectly. My code also validated for checking that the pickup date is earlier than the drop off date. My entire code where I spent good chunk of my time had to be wiped out when I was forced to use the date variables from the search functionality. I then called C3.1 story instead, which does validate but causes the programme to crash entirely if the user types a typo. The difficulty was that my approach required the user to provide dates, which would complicate things if they hadn't done so already since there would be two global variables for pickup and drop-off dates, each with a different pick-up date and drop-off date. These dates would then be used for booking summaries, payment summaries, and sending data to the database's rental table. which would lead to numerous errors because we are unsure of the appropriate variable to use. I was able to resolve this issue by using a simple if statement, which redirects to that method if the user has not entered any details before the booking stage and retrieves the global startDate and endDate variables from that method for my calculation summary. This issue was raised as a big deal, but I am very proud that I was able to solve it using simple solution.

In story E2.0.1, my challenge was to bring all the summaries of the data that the user had previously selected. The issue was that there were two ways to choose a vehicle and booking dates: one was through the search menu, and the other was through the booking creation process. In stories E1.1, E1.4, and E1.5, I subtly modified and changed the code so

that even if a user rents a separate vehicle or dates, it will only fetch the updated date and vehicle.

In user story [E2.1](#), my responsibility was to take card details from the user, although we planned not to store the card details, I had to validate the length of the card number must be exactly 16 digits long, excluding negative numbers and alphabets. The expiry date on the card is just the expiry month and year, so I had to convert that and make an expiry date out of it. Then, I used that to compare with the user's current date and time of making payment, which would throw an error if the card expiry date were earlier than the user's current date of entering details. This was harder to implement than I thought it would be. The month field also needed to be between months 1 and 12, only accept two digits, and not be negative or contain any alphabets. The same holds true for the year of expiration. In the end, my programme had to request a three-digit CVV, which was not as challenging as validating a card's expiration.

Initially, I was assigned to work on story E2.2, which entails database data entry and user confirmation of the reservation. However, since Agnes lacked the necessary points to finish, I gave this story to her because I had to write other stories on which mine depended. This story was also in charge of adding the rental ID, payment date, and payment amount to the database's payment table whenever a booking was confirmed. Agnes did create the functionality to add the code to the rental table and confirm bookings, but she missed to include the ability to populate the payments table as soon as the user confirmed the booking, which would have generated the rental ID needed in the payments table to identify which booking the payment is for. This omission is probably the result of misunderstanding. When we got to know it was too late to amend any of changes in the code, however, I have created an SQL procedure which achieves the same outcome and could be used within code, it is saved inside
postgres>KP_Finalise_Booking>insert_records_into_payment_table.sql.

In story [E2.3](#), I was tasked to show the booking reference ID once the user has made confirmation in user story E2.2. There were several ways to accomplish this; I could have either written a Java programme or created a SQL function and used it to retrieve the booking reference ID. I chose to use a sql query within Java code because I thought it was much easier. But just to not lose my grades, I also developed a SQL function that could have been used to bring the same outcome, but I like to keep my code in a one file because I find the Java side to be simpler.

I was also able to add another into this sprint from backlog E3.1 which shows user their rental history. It was achieved with simple SQL query within Java, but I also developed an SQL function that could have been used to bring the same outcome, it can be found in postgres>KP_Finalise_Booking folder.

Database Task

We all had our own database versions before the project, as I mentioned in the evaluation, since Arthur had already made a lot of progress, we simply decided to take his, and we started working from there. On a five-hour team call, we made the necessary adjustments and used them for our project. Elanwy performed the database testing and used Mockito for that.

Peer Evaluation

We had a good run as a team. Everyone made a significant contribution to this project. In addition to his 46 story points, Arthur's database work efforts stand out. I contributed to 57 story points in total. Elanwy also did excellent work. She contributed 68 story points in total, which was significant. Speaking of Agnes, she was good too and helped team by contributing to 26 story.

Conclusion

Despite any flaws, I am proud of the knowledge I learned from this assignment and the things I contributed. Learning, in my opinion, is more meaningful than anything. Since everyone was preoccupied with their own tasks, there was limited teamwork. We initially needed a flow chart to decide on the application's design, which was one thing we were missing and led to some sudden alteration. I was disappointed because after putting a lot of effort into getting my code to work, one of my peers persuaded me to change the flow at the last day which made things difficult for me. Agile, in my opinion, is also very important because it gives people the flexibility to modify requirements and the outcome as needed. Agile enabled me to deliver E3.1 in this sprint, which allowed me to add it to the sprint.

Links

Here is link to my video demonstration - <https://youtu.be/hGf94vpINSw>

GitHub repo link - <https://github.com/elanwyevans/car-rental-company>

Miro board link - <https://miro.com/app/board/uXjVP8q41Qk=/>

222145_Assignment2_Combine_Assginment

GRADEMARK REPORT

FINAL GRADE

70/100

GENERAL COMMENTS

Instructor

Sw Des Eval (30): 24
Database Eval (30): 22
Code & Test (30): 20
Reflection (10): 4
Feedback:
=====

Good team work and software design. Good evaluation of the process.

The database design looks a little strange and parts of it are not in 3NF.
Evidence of teamwork in design, but more evaluation of the design choices could have been provided.

Code has been completed to a good degree, some task might have been a little more challenging.

Not very much reflection provided, should have provided more discussion of the challenges you faced.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

