Assignment 1

Thi Van Anh DUONG Student ID: 90023112

Diploma of Information Technology, Curtin College

ISEN1000: Introduction to Software Engineering

Coordinator: Khurram Hameed

18 April 2022

Student Declaration of Originality

☒	This assignment is my own original work, and no part has been copied from another student's work or source, except where it is clearly attributed.
☒	All facts/ideas/claims are from academic sources are paraphrased and cited/referenced correctly.
☒	I have not previously submitted this work in any form for THIS or for any other unit; or published it elsewhere before.
☒	No part of this work has been written for me by another person.
☒	I recognise that should this declaration be found to be false, disciplinary action could be taken and penalties imposed in accordance with Curtin College policy.

Electronic Submission:

☒	I accept that it is my responsibility to check that the submitted file is the correct file, is readable and has not been corrupted.
☒	I acknowledge that a submitted file that is unable to be read cannot be marked and will be treated as a non-submission.
☒	I hold a copy of this work if the original is damaged, and will retain a copy of the Turnitin receipt as evidence of submission.

A. Requirments and Planing

I. Functional requirements:

1. Actor:

A software system uses to control multiple buses for: **TRANSPERTH**

- Administrators (Human)
- Passengers (Human)
- Operators (Human)
- Drivers (Human)
- Developers (Human)
- Database (Non-Human)
- Sensors (Non-Human)
- GPS (Non-Human)

2. User stories:

- Administrators:
  - As an administrator, I want to login to **Transperth** fanpage so that I can response to user's feedback.
  - As an administrator, I want to add routes so that all passengers can catch their bus in proper time.
  - As an administrator, I want to update routes from 6.am to 10p.m so that passengers can plan their traveling.
  - As an administrator, I want to remove some routes that having less passengers riding.
  - As an administrator, I want to add, change or remove Bus Stations.
  - As an administrator, I want to display real time bus location or any delay in their routes.
  - As an administration, I want to extract information regarding passenger density in a day/week/month so that it can greatly facilitate for further planning and development of the whole system.
  - As an administration, I want to add, edit or delete the drivers from drivers' list in the system so that I can ensure the system's quality.

- Passengers:
    - As a service user, I want plan my journey in **Transperth** system so that I know when I can catch my bus and know how long does it take.
    - As a passenger, I want to save my favorite journey so that I can travel faster.
    - As a passenger, I want to pay my ticket via mobile app or travelling card with an automatic account management.
    - As a passenger, I want to create an account, login in order to use the bus service.
    - As a student (passenger), I want to get discount (student fare) for my ticket so that I can save money.
    - As a passenger, I want to give **Transperth** my feedback or complains so that the whole system can improve their quality better.
    - As a passenger, I want to check route information, bus stops, departure time, estimated time of arrival of stops so that I will not miss my bus.

- Operators:
    - ⇨ As an operator, I want to appoint a new driver or more driver so that avoiding the probability of being late in special circumstances (traffic congestion, accidents, bus broken, etc…)

- Drivers:
    - ⇨ As a driver, I want to view my passenger's feedback regarding to driving skills, attitude so that I can improve it.

- Developers:
    - ⇨ As a developer, I want to design more attributes for the system so that the whole bus system can run smoothly.

3. Use cases

Use case 1: Log in

- *Goal*: To log in to the system.
  Primary actor: member (**Administrator, Passenger)**
  Secondary actor: Database
  Trigger: Administrator click "log in" button to login to the system.
  Precondition: member that haven't login to the system.

  Flow of event:
  1. Member selects 'Log in'
  2. Login form appears.
  3. Type user name, password to the login form
  4. The system will double check user name, password of the member.
  5. If the member types the wrong username and password the use case will move to extension 1A. Otherwise, continuing performing other features.
  6. Use case end.

  Extension:

  1A. Member login unsuccessfully:

  I. The system announces that login process unsuccessfully.

  II. Select "Sign in" or re-type user name and password. If selecting "Sign in", continue use case 2: Sign in

  III. The system asks member re-type user name and password.

  IV. If the member agrees, the use case resumes at step 2 (FOE). Otherwise, the use case ends.

Use case 2: Sign in

- *Goal*: To sign in as the member of the system.
  Primary actor: member (**Administrator, Passenger)**
  Secondary actor: Database
  Trigger: Administrator click "sign in" button to sign in to the system.
  Precondition: member that haven't used to the system before.

  Flow of event:
  1. Member selects 'Sign in'
  2. Sign in form appears.
  3. Type necessary personal information to the form.
  4. Click 'Submit' button.
  5. The system announces the result of entering personal information. If member's information is not true then the use case will move to extension 1A. Otherwise, the use case will continue at step 6.
  6. The system updates member's information to member list.
  7. Use case ends.

  Extension:

  1A. Inaccurate information:

  I. The system announces that the information is inaccurate.

  II. The system asks user check their information again

  III. If user agrees, the use case resume at step 2. If not the use case ends.

Use case 3: Manage drivers

- *Goal*: To add, edit information or delete drivers from the drivers' list.
  Primary actor: **Administrator**
  Secondary actor: Database
  Trigger: Administrator click "confirm" to perform the operation.
  Precondition: log in successfully to the system.

  Flow of event:

  1. The administrator chooses option to do with drivers: add, update or delete drivers from the list.

     A. Add drivers:
        1.1. The system appears form to key the driver's information.
        1.2. The Administrator types in the driver's information
        1.3. Click "Save" button
        1.4. If typing successfully, the use case will continue at step 5. Otherwise, the use case will enter extension 1A
        1.5. Save driver's information.

     B. Edit driver's information:
        1.1. The system appears form to edit driver's information.
        1.2. The administrator keys all information that needs to update.
        1.3. Click "Save" button
        1.4. If save successfully, the use case will move to step 5. Otherwise, the use case will enter extension 1A.
        1.5. Save driver's information.

     C. Delete driver:
        1.1. The administrator chooses drivers want to delete.
        1.2. Click "Delete" to delete driver.
        1.3. The system ask administrator to confirm deleting. If the administrator agrees, the use case will continue at step 4. Otherwise, the use case will move to step 5.
        1.4. Announcing that driver has been deleted.
        1.5. The system displays the updated driver's list.
  2. The use case end.

Extension:

1A. Inaccurate information:

1. The system announces that the information is inaccurate.
2. The administrator types information again
3. The use case resumes at step 1.3.

Use case 4: Remove bus route

- *Goal*: To remove bus route that have less passengers riding.
  Primary actor: **Administrator**
  Secondary actor: Database
  Trigger: Administrator click "confirm" button to remove the class.
  Precondition: computer-based knowledge, log in successfully to the system.

  Flow of event:

  1. Administrator gather lists of regions, times, buses that usually have less passenger taking part in.
  2. Administrator selects routes he/she wants to delete in the app.
  3. Administrator click "remove".
  4. The system ask password for confirm deleting.

  Extension:

  1A. Administrator select the wrong route:

  I. The administrator goes back to the old working session.

  II. The administrator click "recover" button to put the route being deleted wrong.

  III. The use case resumes at step 2.

  1B. Administrator forgets his/her password.

  I. The system asks administrator select "Forget password" option.

  II. The system asks administrator reset password via email or phone number.

  III. The use case resumes at step 4.

Use case 5: Plan a journey

- *Goal*: To plan a journey.
  Primary actor: **Passenger**
  Secondary actor: Database, GPS
  Trigger: click "OK" button to plan a journey.
  Precondition: login or create a new account to start using bus services

  Flow of event:
  1. Passenger login or create a new account to begin using the service
  2. Passenger key their current position, their desired position, date and time in searching bar.
  3. Click "OK" to start find proper journeys.
  4. Passenger choose the suitable journey.

  Extension:

  1A. No bus is suitable to passenger's requirement:
  - I. The system will appear the message "Oh no, your bus is not available".
  - II. The system will recommend passenger the latest route with reasonable time.
  - III. The passenger chooses their route.
  - IV. The use case end.

  1B. The bus was unexpectedly canceled, the departure time was delayed.
  - I. The system will send passenger a message "Sorry! <content>"
  - II. Passenger checks bus information.
  - III. Passenger chooses another bus.
  - IV. The use case resumes at step 2.

Use case 6: Distribute drivers

- *Goal*: To distribute drivers.
  Primary actor: **Operator**
  Secondary actor: Database, GPS, Drivers
  Trigger: click "Confirm" button to appoint appropriate drivers.
  Precondition: knowing driver's schedule.

  Flow of event:
  1. Administrator gather lists of drivers who are free in the time they are going to be appointed.
  IV. Administrator selects drives he/she thinks suitable.
  V. Administrator click "confirm" to start appointing drivers.
  VI. The system sends a message to drivers about the route information.

  Extension:
  1A. The driver refuses:
  I. The system will announce to the operator.
  II. The operator will find another driver.
  III. If cannot the use case resumes at step 2.
  IV. The operator will delete this route.

Use case 7: Manage Bus Station

- *Goal*: To add, change or remove Bus Station.
  Primary actor: **Administrator**
  Secondary actor: Database
  Trigger: click "Confirm" button to perform the operation.
  Precondition: already log in successfully to the system.

Flow of event:

1. The administrator chooses option to do with Bus Station: add, change or remove.
   A. Add Bus Station:
      1.1. The system appears form to key the Bus Station information.
      1.2. The Administrator types in the Bus Station information
      1.3. Click "Save" button
      1.4. If typing successfully, the use case will continue at step 5. Otherwise, the use case will enter extension 1A.
      1.5. Save place information.

   B. Change Bus Station location:
      1.1. The system appears form to edit place information.
      1.2. The administrator keys all information that needs to update.
      1.3. Click "Save" button
      1.4. If save successfully, the use case will move to step 5. Otherwise, the use case will enter extension 1A.
      1.5. Save place information.

   C. Remove Bus Station:
      1.1. The administrator chooses place want to delete.
      1.2. Click "Confirm" to remove the place.
      1.3. The system ask administrator to confirm removing. If the administrator agrees, the use case will continue at step 4. Otherwise, the use case will move to step 5.
      1.4. Announcing that place is no longer active
      1.5. The system displays the updated Bus Station list.
2. The use case end.

Extension:

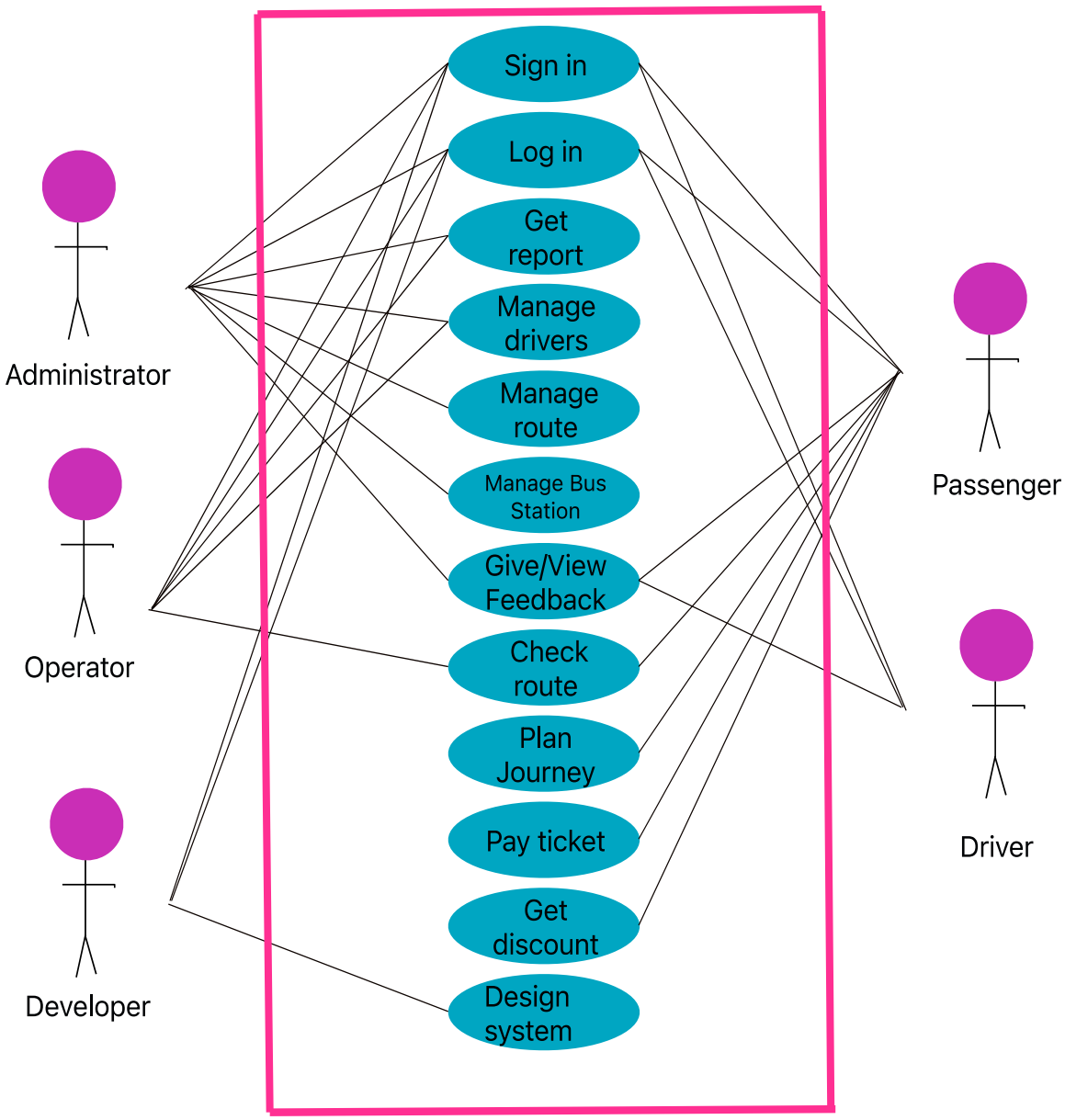
1A. Inaccurate information:

1. The system announces that the information is inaccurate.

2. The administrator types information again.
3. The use case resumes at step 1.3.

4. Use case diagram

A software system uses to control multiple buses for Transperth: **TRANSPERTH**



II. Non-Functional requirements:

1. Usability Requirement

- The software system must show all journey options, estimated departed times, prices for customers on the screen.
- The software system must allow passengers locate their position with the accuracy at least 99% of the time.
- The software system must allow administrator to track the bus.
- The software system must allow drivers to know about the information of their route which they are in charge.
- The software system must allow central control staff export passenger density as PDF.

2. Performance Requirement

- Response time: response to the arrival time of the latter bus and departure time of the previous one (to avoid traffic congestion at bus stop).
- Response time: response to the accidental occasion that delay the bus route.
- Response time: keep passenger updated with the changes (late or soon) and update route (additional bus at restricted hours or school holiday, public holiday, etc…)
- Data processing: update up to 100 new bus route per 15 minutes.
- How efficient its use of resources is? Internet connection (Wireless, 3G, 4G, …), GPS, Electrical Blackboard (Showing bus time), bus station.

3. Reliability Requirement

- Mean Time To/Between Failure (MTTF/MTBF): The system must have a MTTF at least 1 year
- Availability (AVAIL): The system should be available 99,999% of the time.
- Probability of Failure on Demand (POFOD): The system's control feature should have a POFOD of 0.1% or less.
- Rate of Occurrence of Failure (ROCOF): The system must not have more than 1 bus is late in every 1000 hours.

III. Planning

1. Work Breakdown

a. *Investigate **Transperth** bus management model*
b. *Analyze and design system*
c. *Recruit Employees:*
   c.1. Administrators
   c.2. Operators
   c.3. Drivers
   c.4. Developers
d. *Facilities:*
   d.1. Buses
   d.2. Bus stations
      d.2.1. Locations
      d.2.2. Parking policies (special events, rush hours, etc…)
e. *Create Database:*
   e.1. Driver information
   e.2. Route information
   e.3. Bus station information
f. *Create sign in and login feature*
g. *Create functional features:*
   g.1. Manage drivers
      g.1.1. Add drivers' information
      g.1.2. Edit drivers' information
      g.1.3. Remove drivers' information
   g.2. Manage routes
      g.2.1. Add route
      g.2.2. Edit route
      g.2.3. Remove route
   g.3. Manage bus stations
      g.3.1. Add bus station
      g.3.2. Change bus station
      g.3.3. Remove bus station
   e.4. Get report
h. *Ticket:*
   h.1. Payment
   h.2. Discount
i. *Journey:*
   i.1. Plan a journey
   i.2. Check journey
j. *Feedback*
k. *Testing the system*
l. *Integration and maintenance*

2. Estimation (WEEKS)

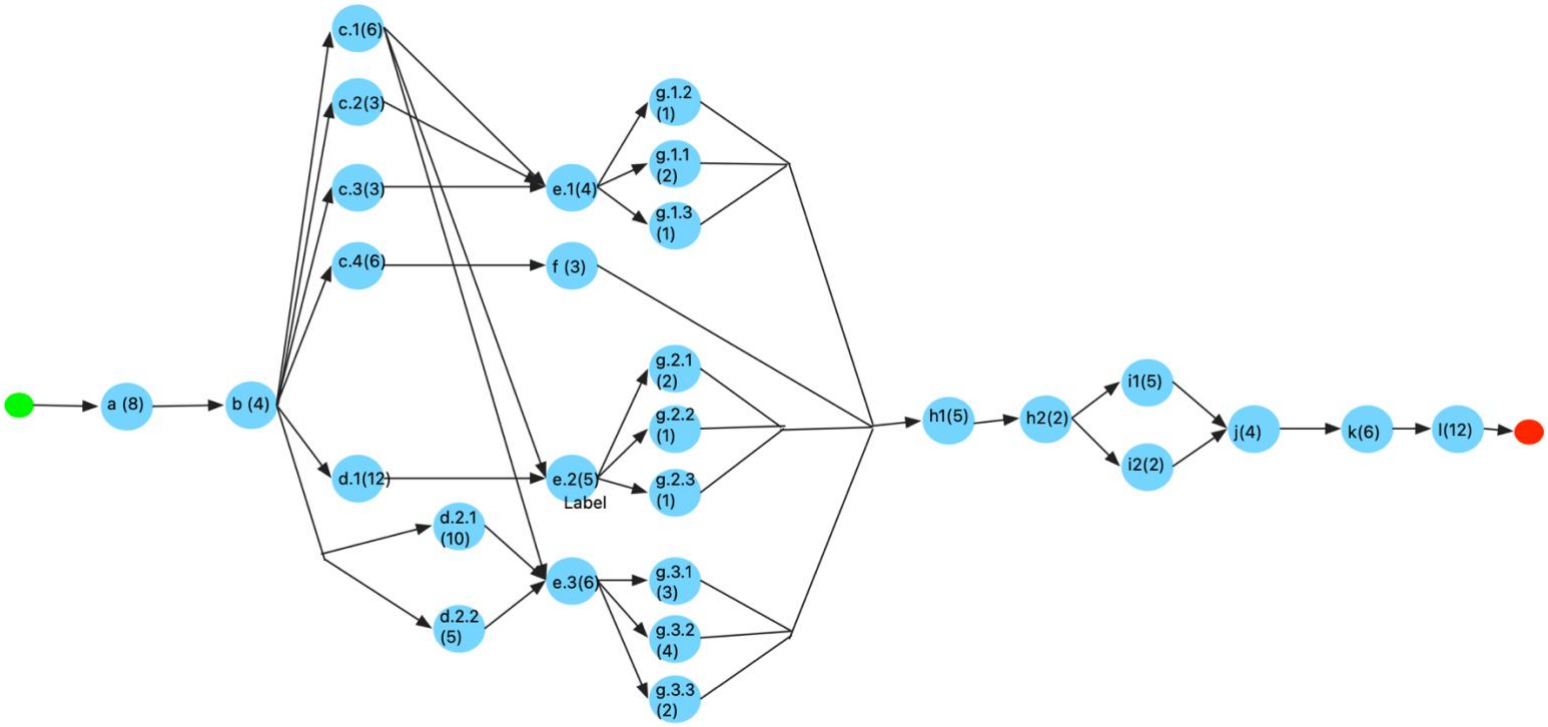
a.	*Investigate **Transperth** bus management model*	8W
b.	*Analyze and design system*	4W
c.	*Recruit Employees:*	
	c.1. Administrators	6W
	c.2. Operators	3W
	c.3. Drivers	3W
	c.4. Developers	6W
d.	*Facilities:*	
	d.1. Buses	12W
	d.2. Bus stations	
	d.2.1. Locations	10W
	d.2.2. Parking policies (special events, rush hours, etc…)	5W
e.	*Create Database:*	
	e.1. Driver information	4W
	e.2. Route information	5W
	e.3. Bus station information	6W
f.	*Create sign in and login feature*	3W
g.	*Create functional features:*	
	g.1. Manage drivers	
	g.1.1. Add drivers' information	2W
	g.1.2. Edit drivers' information	1W
	g.1.3. Remove drivers' information	1W
	g.2. Manage routes	
	g.2.1. Add route	2W
	g.2.2. Edit route	1W
	g.2.3. Remove route	1W
	g.3. Manage bus stations	
	g.3.1. Add bus station	3W
	g.3.2. Change bus station	4W
	g.3.3. Remove bus station	2W
	e.4. Get report	2W
h.	*Ticket:*	
	h.1. Payment	5W
	h.2. Discount	2W
i.	*Journey:*	
	i.1. Plan a journey	1W
	i.2. Check journey	1W
j.	*Feedback*	4W
k.	*Testing the system*	6W
l.	*Integration and maintenance*	12W

3. Prerequisite Activities

Activity	Dependencies	Duration
a	-	8W
b	a	4W
c.1	b	6W
c.2	b	3W
c.3	b	3W
c.4	b	6W
d.1	b	12W
d.2.1	b	10W
d.2.2	b	5W
e.1	c.1, c.3	4W
e.2	c.1, d.1	5W
e.3	c.1, d.2	6W
f	c.4	3W
g.1.1	c.2, e.1	2W
g.1.2	c.2, e.1	1W

g.1.3	c.2, e.1	1W
g.2.1	e.2	2W
g.2.2	e.2	1W
g.2.3	e.2	1W
g.3.1	e.3	3W
g.3.2	e.3	4W
g.3.3	e.3	2W
h.1	f, g.1, g.2, g.3	5W
h.2	f, g.1, g.2, g.3	2W
i.1	h	5W
i.2	h	2W
J	i	4W
k	j	6W
l	k	12W

Activity-on-Node (AON) Graph With Timing



4. Critical Path and Slash Time

Find the earliest start (ES) and earliest finish (EF) for each activity.

- ⇨ Start with the activities that don't depend on anything (activity a)
- ⇨ Work forwards until get to the final activity.
- ⇨ Early Start Time = 0
- ⇨ EF = ES + Duration

I have the table as below:

Activity	Duration	ES	EF
a	8	0	8
b	4	8	12
c.1	6	12	18
c.2	3	12	15
c.3	3	12	15
c.4	6	12	18
d.1	12	12	24
d.2.1	10	12	22
d.2.2	5	12	17
e.1	4	18	22
e.2	5	24	29
e.3	6	22	28
f	3	18	21
g.1.1	2	22	24
g.1.2	1	22	23
g.1.3	1	22	23
g.2.1	2	29	31
g.2.2	1	29	30
g.2.3	1	29	30
g.3.1	3	28	31
g.3.2	4	28	32
g.3.3	2	28	30
h.1	5	32	37
h.2	2	37	39
i.1	5	39	44
i.2	2	39	41
J	4	44	48

k	6	48	54
l	12	54	66

Find the latest start (LS) and latest finish (LF) for each activity.

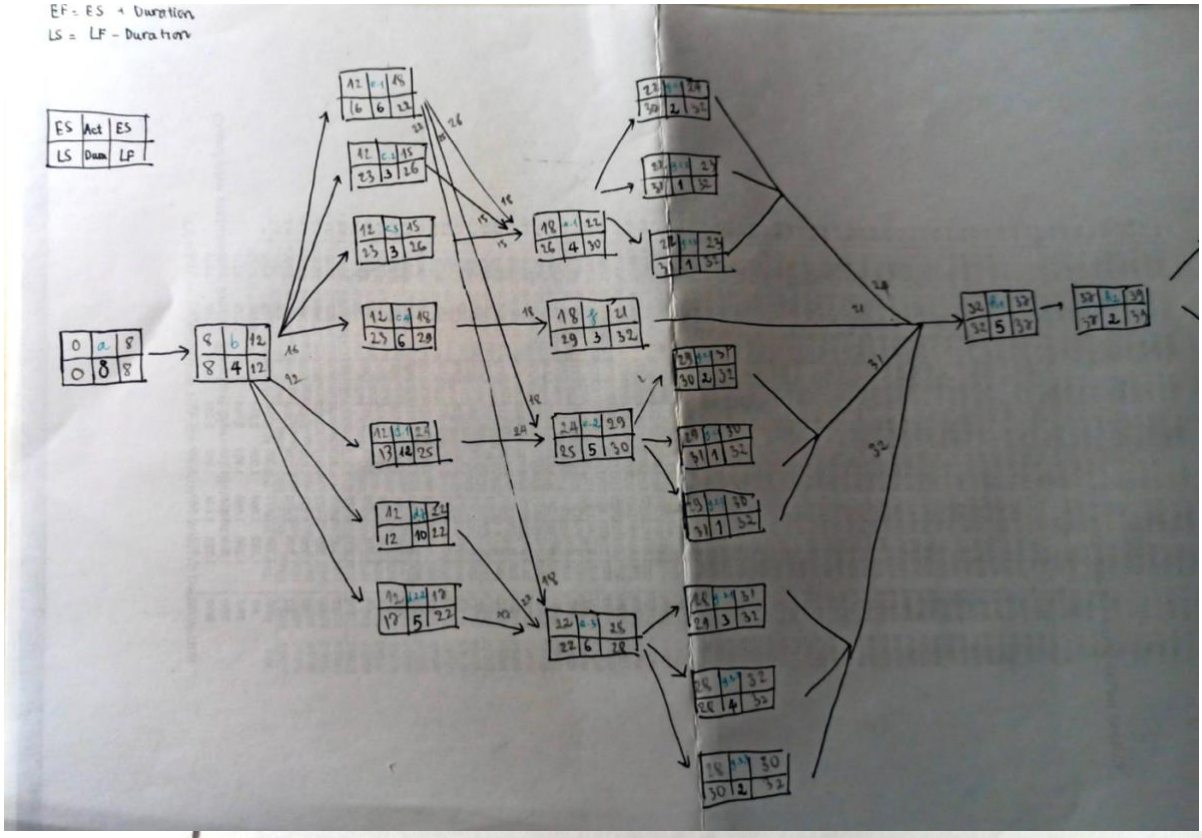
- ⇨ Start with the activities at the end of the project
- ⇨ Latest Finish of the activity l = Earliest Finish of the activity l
- ⇨ Work backward until get to the first activity
- ⇨ LS = LF – Duration

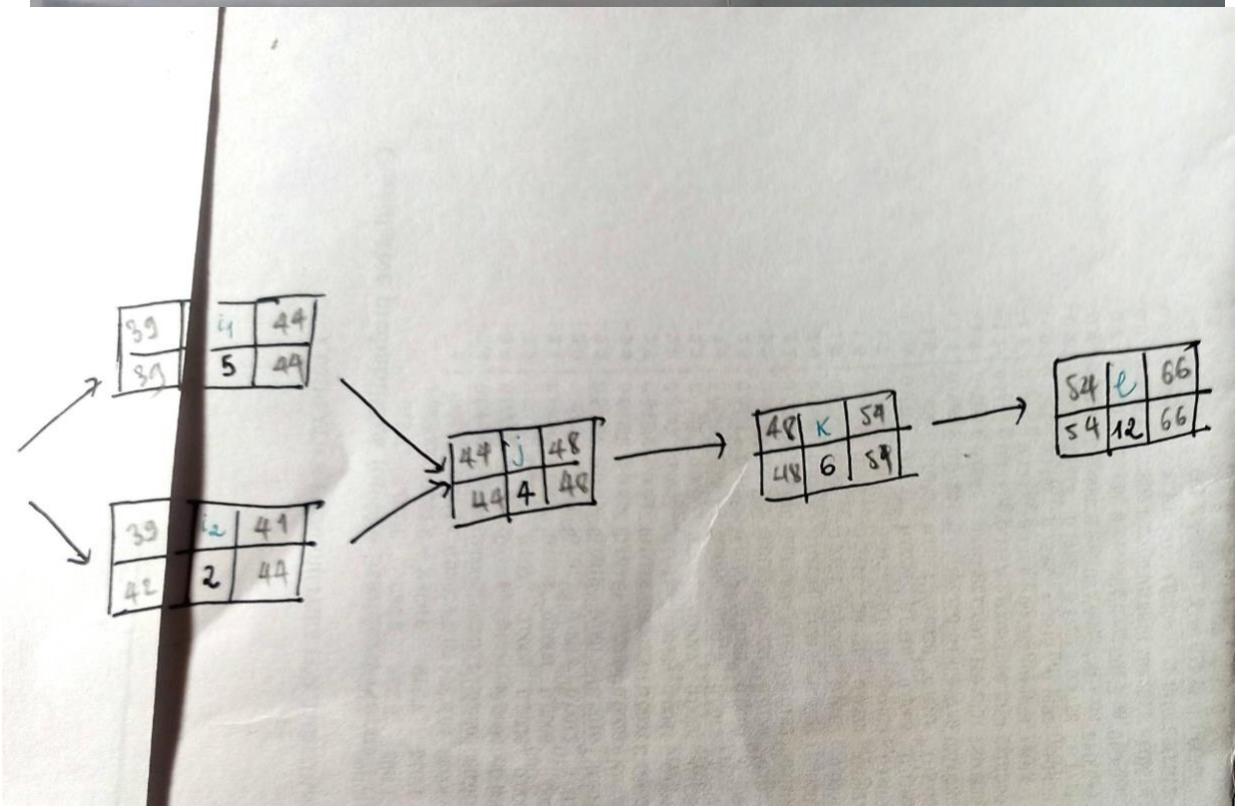
I have the table as below:

Activity	**Duration**	**LF**	**LS**
l	12	66	54
k	6	54	48
J	4	48	44
i.2	2	44	42
i.1	5	44	39
h.2	2	39	37
h.1	5	37	32
g.3.3	2	32	30
g.3.2	4	32	28
g.3.1	3	32	29
g.2.3	1	32	31
g.2.2	1	32	31
g.2.1	2	32	30
g.1.3	1	32	31
g.1.2	1	32	31
g.1.1	2	32	30
f	3	32	29
e.3	6	28	22
e.2	5	30	25
e.1	4	30	26
d.2.2	5	22	17
d.2.1	10	22	12
d.1	12	25	13
c.4	6	29	23
c.3	3	26	23

c.2	3	26	23
c.1	6	22	16
b	4	12	8
a	8	8	0 (equal to project duration)

Here is the diagram that I use to calculate ES, EF, LS, LF.





Slash Time = LF – EF = LS – ES

Activity	Duration	ES	EF	LF	LS	Slash Time
a	8	0	8	8	0	0
b	4	8	12	12	8	0
c.1	6	12	18	22	16	4
c.2	3	12	15	26	23	11
c.3	3	12	15	26	23	11
c.4	6	12	18	29	23	11
d.1	12	12	24	25	13	1
d.2.1	10	12	22	22	12	0
d.2.2	5	12	17	22	17	5
e.1	4	18	22	30	26	8
e.2	5	24	29	30	25	1
e.3	6	22	28	28	22	0
f	3	18	21	32	29	11
g.1.1	2	22	24	32	30	8
g.1.2	1	22	23	32	31	9
g.1.3	1	22	23	32	31	9
g.2.1	2	29	31	32	30	1
g.2.2	1	29	30	32	31	2
g.2.3	1	29	30	32	31	2
g.3.1	3	28	31	32	29	1
g.3.2	4	28	32	32	28	0
g.3.3	2	28	30	32	30	2
h.1	5	32	37	37	32	0
h.2	2	37	39	39	37	0
i.1	5	39	44	44	39	0
i.2	2	39	41	44	42	3
J	4	44	48	48	44	0
k	6	48	54	54	48	0
l	12	54	66	66	54	0

Critical Path: **a → b → d.2.1 → e.3 → g.3.2 → h1 → h2 → i1 → j → k → l.**

In this critical path, each activity is a critical activity. If I delay any of them, I will delay the whole project.

Therefore, in order to ensure the progress of the bus management software system, I have to always monitor the progress of critical activities (because it will tell me whether the project is on track or not).

5. A Plan for the use of version control

According to Work Break Down Structure, I have pointed out 5 tasks for the development team to complete during the project: Create Database, Create Sign in and Log in feature, Create Functional feature, Ticket Feature and Journey feature and appoint these 5 tasks for 5 developers. Following the estimated project duration table above, I will create each branch at Earliest Start time (ES) equivalent to each task.

- I intend to create 5 Branches:
  - ⇨ Branch 1: Developer 1 – for creating database (e). This branch will be created at Week 18 after Investigating **Transperth** bus management model, Analyzing and designing system, Recruiting Employees, Facilities steps. Although, activity e has e.1, e.2 and e.3 start at week 18, 24 and 22 respectively, I will create the branch at week 18 (the earliest time) so that the developer when completing this task will switch to another task immediately without waiting for a long time.
    Merge: it will be merged to master branch after finishing latest at week 30 (Latest Finish Time – LF)

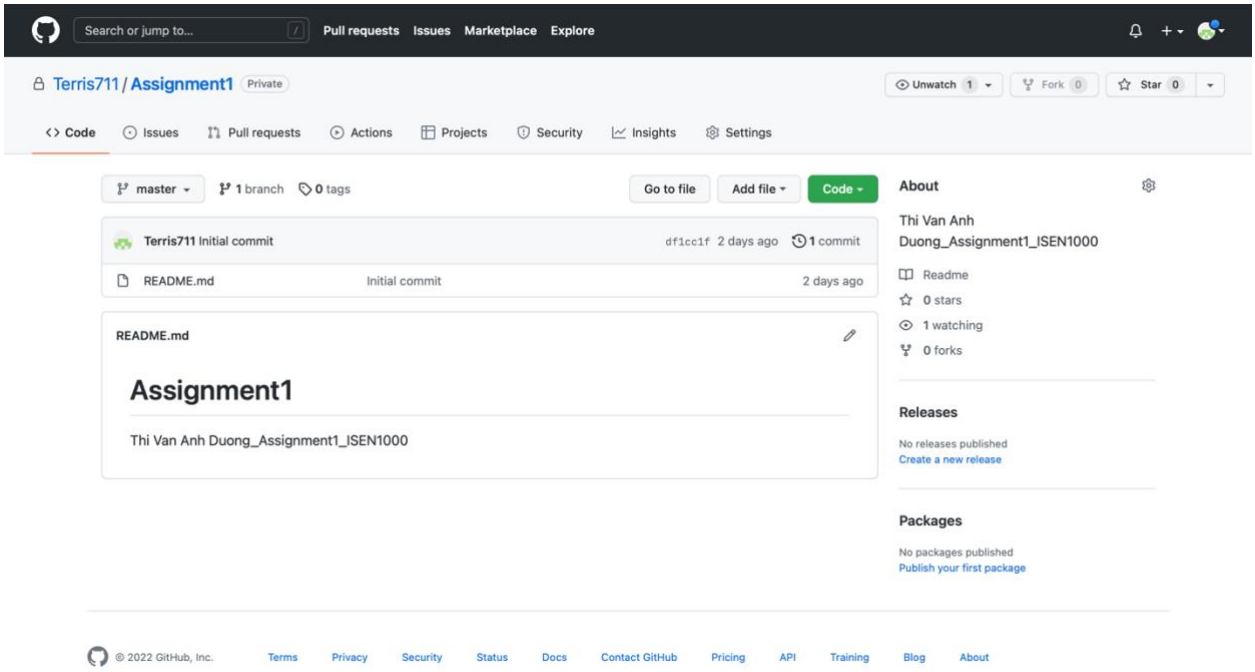
    Similarly:
  - ⇨ Branch 2: Developer 2 – for creating sign in and log in interface (f). This branch will be created at the same time as Branch 1, at Week 18.
    Merge: it will be merged to master branch after finishing latest at week 32 (Latest Finish Time – LF)
  - ⇨ Branch 3: Developer 3 – for creating functional feature (g). This branch will be created at week 22 when the code in branch Developer01 finishes.
    Merge: it will be merged to master branch after finishing latest at week 32 (Latest Finish Time – LF)

- ⇨ Branch 4: Developer 4 – for ticket feature (h). This branch will be created at week 32 when the code in branch Developer01, Developer02, Developer03 finish.
  Merge: it will be merged to master branch after finishing latest at week 39 (Latest Finish Time – LF)
- ⇨ Branch 5: Developer 5 – for journey feature (i). This branch will be created at week 39 when the code in branch Developer 04 finish.
  Merge: it will be merged to master branch after finishing latest at week 44 (Latest Finish Time – LF)

Finally, when 5 branches are completed, I will push my local repository to my GitHub account.

IV. Version Control

Create a project in Github:



Set up identity:

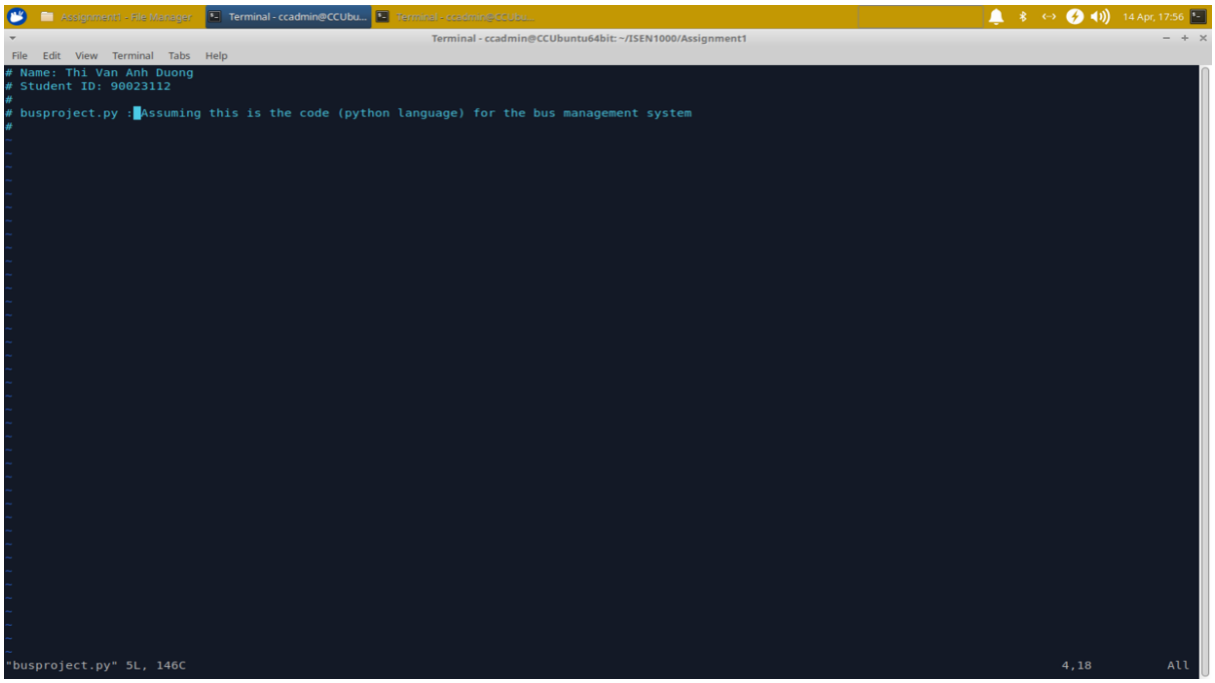
```
ccadmin@CCUbuntu64bit:~/ISEN1000$ cd Assignment1
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git config --global user.name "Thi Van Anh Duong"
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git config --global user.email "90023112@study.curtincollege.edu.au"
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git config user.name
Thi Van Anh Duong
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git config user.email
90023112@study.curtincollege.edu.au
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$
```

Make an initial commit:

```
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .~lock.Assignment1_Thi Van Anh Duong_90023112.docx#
        Assignment1.docx

nothing added to commit but untracked files present (use "git add" to track)
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git add .
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   .~lock.Assignment1_Thi Van Anh Duong_90023112.docx#
        new file:   Assignment1.docx

ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git commit -m "Initial Commit"
[master (root-commit) 1d03a76] Initial Commit
 2 files changed, 1 insertion(+)
 create mode 100644 .~lock.Assignment1_Thi Van Anh Duong_90023112.docx#
 create mode 100644 Assignment1.docx
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$
```

CREATING BRANCHES:

Create busproject.py in master branch where stores all source code of the project:



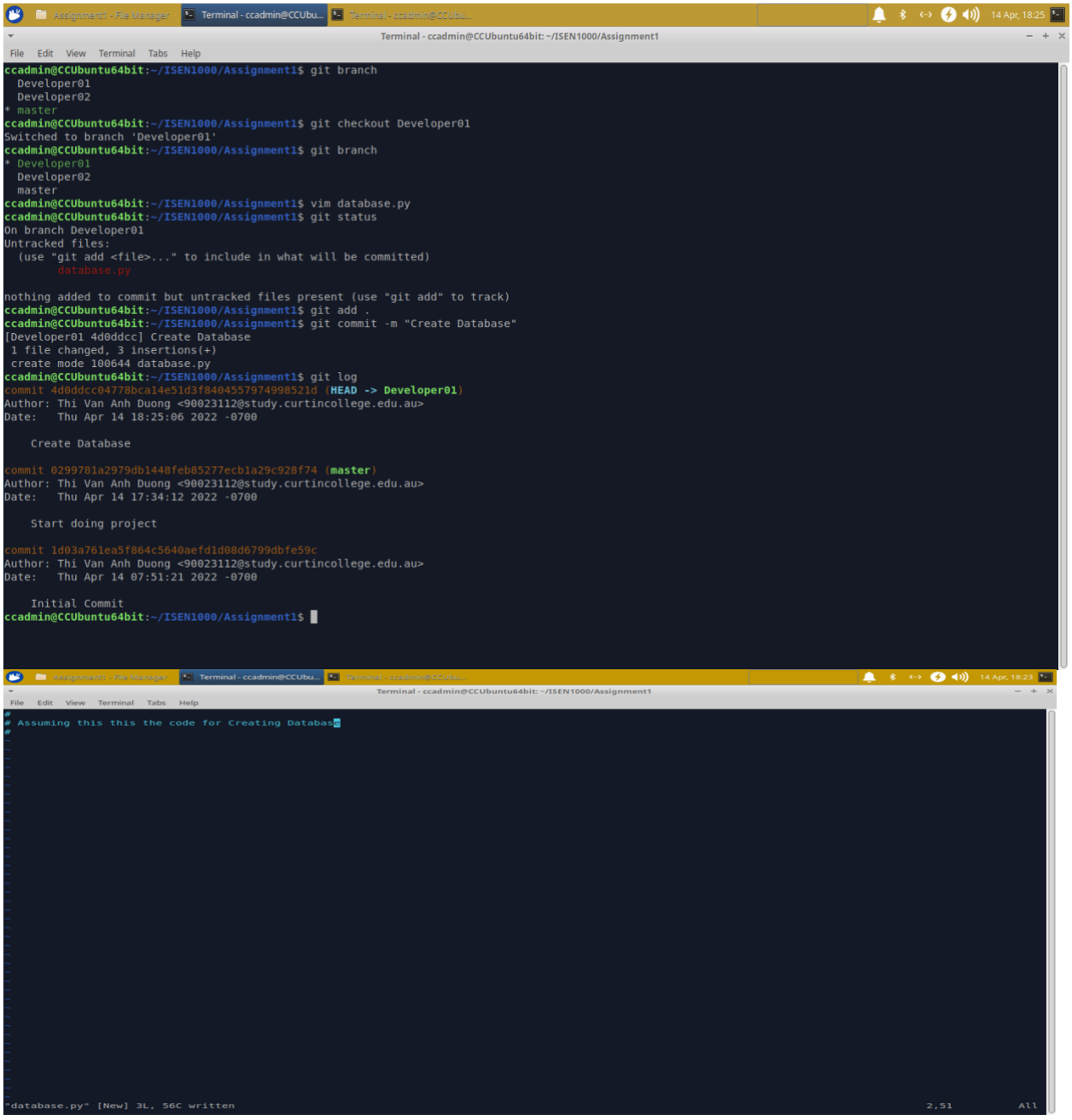
Stage and commit busproject.py

```
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    .busproject.py.swp
        new file:   .project.py.swp
        modified:   busproject.py

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    .project.py.swp
        modified:   busproject.py

ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git add .
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git commit -m "Second Commit"
[master 0ba4342] Second Commit
 2 files changed, 11 insertions(+)
 delete mode 100644 .busproject.py.swp
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git log
commit 0ba4342f0f0c664d29e0728fe025dc35edf73dbe (HEAD -> master)
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Thu Apr 14 21:31:25 2022 -0700

    Second Commit

commit 0299781a2979db1448feb85277ecb1a29c928f74
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Thu Apr 14 17:34:12 2022 -0700

    Start doing project

commit 1d03a761ea5f864c5640aefd1d08d6799dbfe59c
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Thu Apr 14 07:51:21 2022 -0700

    Initial Commit
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$
```

Branch 1:
Create Developer01 at week 18.

```
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git checkout -b Developer01
Switched to a new branch 'Developer01'
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git branch
* Developer01
  master
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$
```

Create database.py where stores the code and stag and commit it.



Let's check. You can see, database.py just exist in branch: Developer01

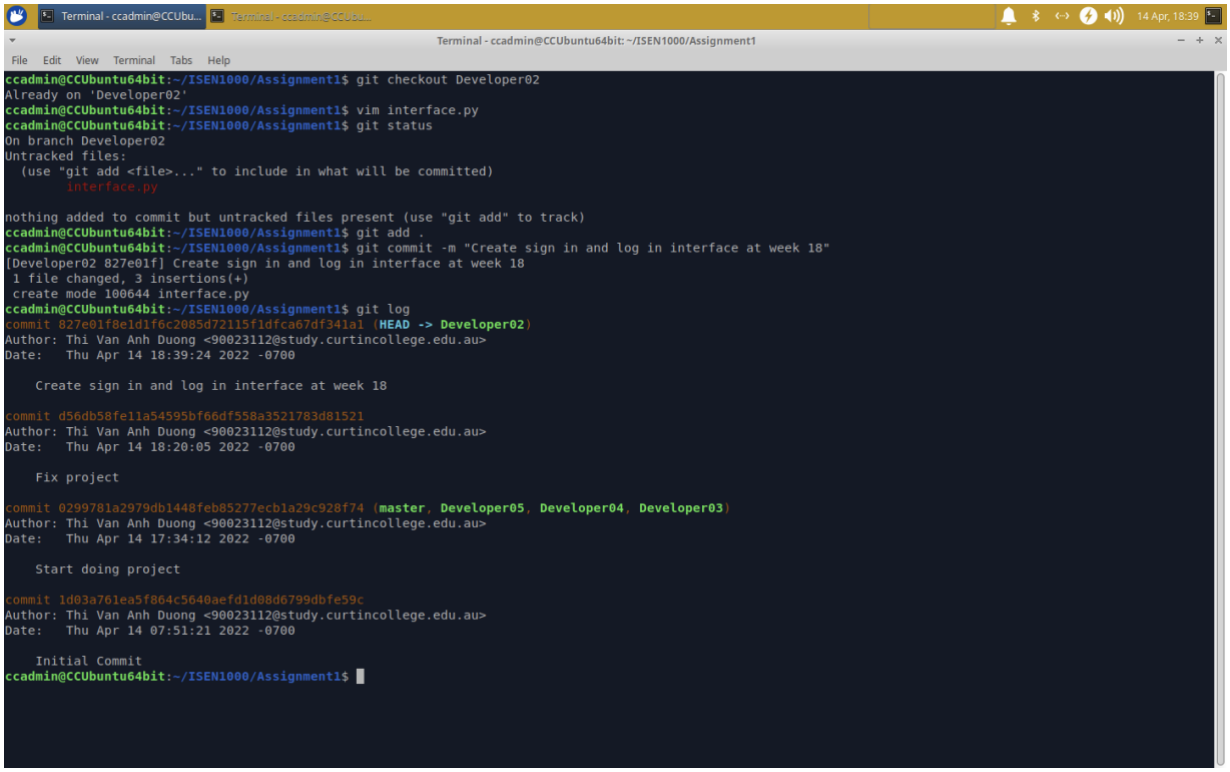
```
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git checkout
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git branch
* Developer01
  Developer02
  master
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ ls
Assignment1.docx  busproject.py  database.py
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git checkout master
Switched to branch 'master'
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ ls
Assignment1.docx  busproject.py
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$
```

When the code is finished by developer 1, we will merge it to master branch at week 30.

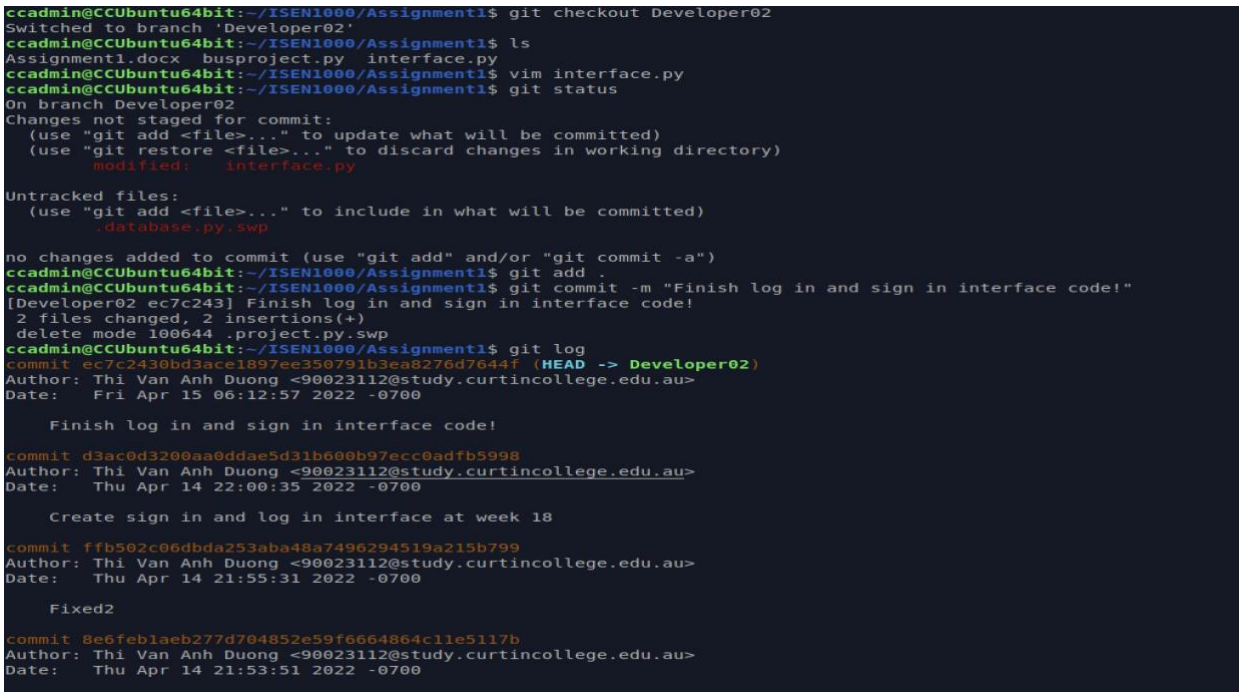
```
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git checkout master
Switched to branch 'master'
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ ls
Assignment1.docx  busproject.py
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git merge Developer01
Updating ffb502c..263934d
Fast-forward
 database.py | 5 +++++
 1 file changed, 5 insertions(+)
 create mode 100644 database.py
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .database.py.swp

nothing added to commit but untracked files present (use "git add" to track)
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ ls
Assignment1.docx  busproject.py  database.py
```

Branch 2:

Similarly, creating branch Developer02 at week 18, at the same time as branch Developer01



When the code is finished, stage and commit.



Merge the finished code to the master branch at week 32.

```
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git branch
  Developer01
  Developer02
* master
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git merge Developer02
Already up to date.
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ ls
Assignment1.docx  busproject.py  database.py  interface.py
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git status
On branch master
nothing to commit, working tree clean
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$
```

Branch 3:

We start create branch at week 22 slightly before the latest finish time of branch Developer01(week 30) and Developer02 (week32)

Create functional.py which contains the code for creating functional features of the system

```
#
# functional.py - Assuming this is the code for creating functional features of the bus management system
#
```

Stage and commit the file

```
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git checkout -b Developer03
Switched to a new branch 'Developer03'
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git branch
  Developer01
  Developer02
* Developer03
  master
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ vim functional.py
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git status
On branch Developer03
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        functional.py

nothing added to commit but untracked files present (use "git add" to track)
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git add .
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git commit -m "Create functional features at week 22"
[Developer03 4650adf] Create functional features at week 22
 1 file changed, 5 insertions(+)
 create mode 100644 functional.py
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git log
commit 4650adf1d212aad443b914f624fd990b120326ee (HEAD -> Developer03)
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Fri Apr 15 06:48:33 2022 -0700

    Create functional features at week 22

commit b652c205035005492cfa92bc163ea5277bbb0a43 (master)
Merge: 263934d ec7c243
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Fri Apr 15 06:19:57 2022 -0700

    Finish login and sigin code at week 32

commit ec7c2430bd3ace1897ee350791b3ea8276d7644f (Developer02)
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Fri Apr 15 06:12:57 2022 -0700

    Finish log in and sign in interface code!

commit 263934de39ba9db32f4dba7eb264ad6f4f4148aa (Developer01)
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Thu Apr 14 22:14:32 2022 -0700

    Finish database code!
```

At week 32 – the latest finish time of creating functional features task – we merge this branch to master branch

```
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git branch
  Developer01
  Developer02
* Developer03
  master
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git checkout master
Switched to branch 'master'
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git branch
  Developer01
  Developer02
  Developer03
* master
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git merge Developer03
Updating b652c20..4650adf
Fast-forward
 functional.py | 5 +++++
 1 file changed, 5 insertions(+)
 create mode 100644 functional.py
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ ls
Assignment1.docx  busproject.py  database.py  functional.py  interface.py
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$
```

Branch 4:

Right after branch Developer01, Developer02. Developer03 finish, we create Developer04 at week 32.

```
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git checkout -b Developer04
Switched to a new branch 'Developer04'
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git branch
  Developer01
  Developer02
  Developer03
* Developer04
  master
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$
```

Create ticket.py which contains code for ticket features

```
#
# ticket.py - Assuming this is code for creating ticket feature
#
```

Stage and commit the file.

```
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ vim ticket.py
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ vim ticket.py
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git status
On branch Developer04
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        ticket.py

nothing added to commit but untracked files present (use "git add" to track)
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git add .
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git commit -m "Create ticket features at week 32"
[Developer04 89e01b8] Create ticket features at week 32
 1 file changed, 4 insertions(+)
 create mode 100644 ticket.py
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git log
commit 89e01b892d181eeaadee64af3640d53dff7323fa (HEAD -> Developer04)
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Fri Apr 15 07:00:12 2022 -0700

    Create ticket features at week 32

commit 4650adf1d212aad443b914f624fd990b120326ee (master, Developer03)
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Fri Apr 15 06:48:33 2022 -0700

    Create functional features at week 22

commit b652c205035005492cfa92bc163ea5277bbb0a43
Merge: 263934d ec7c243
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Fri Apr 15 06:19:57 2022 -0700

    Finish login and sigin code at week 32

commit ec7c2430bd3ace1897ee350791b3ea8276d7644f (Developer02)
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Fri Apr 15 06:12:57 2022 -0700

    Finish log in and sign in interface code!

commit 263934de39ba9db32f4dba7eb264ad6f4f4148aa (Developer01)
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Thu Apr 14 22:14:32 2022 -0700

    Finish database code!
```

When this code is finish at latest time at week 39, we stage and commit the code first:

```
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ vim ticket.py
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git status
On branch Developer04
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   ticket.py

no changes added to commit (use "git add" and/or "git commit -a")
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git add .
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git commit -m "Finish ticket features!"
[Developer04 368dd65] Finish ticket features!
 1 file changed, 1 insertion(+)
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git log
commit 368dd6534057dbf1ae5e6b55a5ce37b270c9cb4a (HEAD -> Developer04)
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Fri Apr 15 07:03:04 2022 -0700

    Finish ticket features!

commit 89e01b892d181eeaadee64af3640d53dff7323fa
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Fri Apr 15 07:00:12 2022 -0700

    Create ticket features at week 32

commit 4650adf1d212aad443b914f624fd990b120326ee (master, Developer03)
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Fri Apr 15 06:48:33 2022 -0700

    Create functional features at week 22

commit b652c205035005492cfa92bc163ea5277bbb0a43
Merge: 263934d ec7c243
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Fri Apr 15 06:19:57 2022 -0700

    Finish login and sigin code at week 32

commit ec7c2430bd3ace1897ee350791b3ea8276d7644f (Developer02)
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Fri Apr 15 06:12:57 2022 -0700

    Finish log in and sign in interface code!
```

Then we merge it to master branch at week 39.

```
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git checkout master
Switched to branch 'master'
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git branch
  Developer01
  Developer02
  Developer03
  Developer04
* master
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git merge Developer04
Updating 4650adf..368dd65
Fast-forward
 ticket.py | 5 +++++
 1 file changed, 5 insertions(+)
 create mode 100644 ticket.py
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ ls
Assignment1.docx  busproject.py  database.py  functional.py  interface.py  ticket.py
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$
```

Branch 5:

Right after branch Developer04 finish, we create Developer05 at week 39.

```
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git checkout -b Devloper05
Switched to a new branch 'Devloper05'
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git branch
  Developer01
  Developer02
  Developer03
  Developer04
* Devloper05
  master
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$
```

Create journey.py which contains code for journey features

```
#
# journey.py - Assuming this is code for creating journey features
#

~
~
```

Stage and commit the file

```
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ vim journey.py
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git status
On branch Devloper05
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        journey.py

nothing added to commit but untracked files present (use "git add" to track)
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git add .
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git commir -m "Create journey features at week 39"
git: 'commir' is not a git command. See 'git --help'.

The most similar command is
        commit
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git commit -m "Create journey features at week 39"
[Devloper05 d5ac786] Create journey features at week 39
 1 file changed, 5 insertions(+)
 create mode 100644 journey.py
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git log
commit d5ac7861c3c4eacc4de2c7dcb6633716543f5923 (HEAD -> Devloper05)
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Fri Apr 15 07:16:54 2022 -0700

    Create journey features at week 39

commit 368dd6534057dbf1ae5e6b55a5ce37b270c9cb4a (master, Developer04)
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Fri Apr 15 07:03:04 2022 -0700

    Finish ticket features!

commit 89e01b892d181eeaadee64af3640d53dff7323fa
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Fri Apr 15 07:00:12 2022 -0700

    Create ticket features at week 32

commit 4650adf1d212aad443b914f624fd990b120326ee (Developer03)
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Fri Apr 15 06:48:33 2022 -0700

    Create functional features at week 22

commit b652c205035005492cfa92bc163ea5277bbb0a43
Merge: 263934d ec7c243
```

When this code is finish at latest time at week 44, we stage and commit the code first:

```
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ vim journey.py
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git status
On branch Devloper05
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   journey.py

no changes added to commit (use "git add" and/or "git commit -a")
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git add .
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git status
On branch Devloper05
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   journey.py

ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git commit -m "Finish journey features!"
[Devloper05 1724cb8] Finish journey features!
 1 file changed, 1 insertion(+), 1 deletion(-)
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git log
commit 1724cb8dfb2fd3edea4f42f35e44944e284d5c84 (HEAD -> Devloper05)
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Fri Apr 15 07:19:00 2022 -0700

    Finish journey features!

commit d5ac7861c3c4eacc4de2c7dcb6633716543f5923
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Fri Apr 15 07:16:54 2022 -0700

    Create journey features at week 39

commit 368dd6534057dbf1ae5e6b55a5ce37b270c9cb4a (master, Developer04)
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Fri Apr 15 07:03:04 2022 -0700

    Finish ticket features!

commit 89e01b892d181eeaadee64af3640d53dff7323fa
Author: Thi Van Anh Duong <90023112@study.curtincollege.edu.au>
Date:   Fri Apr 15 07:00:12 2022 -0700

    Create ticket features at week 32

commit 4650adf1d212aad443b914f624fd990b120326ee (Developer03)
```

Then we merge it to master branch at week 44.

```
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git branch
  Developer01
  Developer02
  Developer03
  Developer04
* Devloper05
  master
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git checkout master
Switched to branch 'master'
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git branch
  Developer01
  Developer02
  Developer03
  Developer04
  Devloper05
* master
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git merge Developer05
merge: Developer05 - not something we can merge
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git merge Devloper05
Updating 368dd65..1724cb8
Fast-forward
 journey.py | 5 +++++
 1 file changed, 5 insertions(+)
 create mode 100644 journey.py
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ ls
Assignment1.docx  busproject.py  database.py  functional.py  interface.py  journey.py  ticket.py
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$
```

Push:

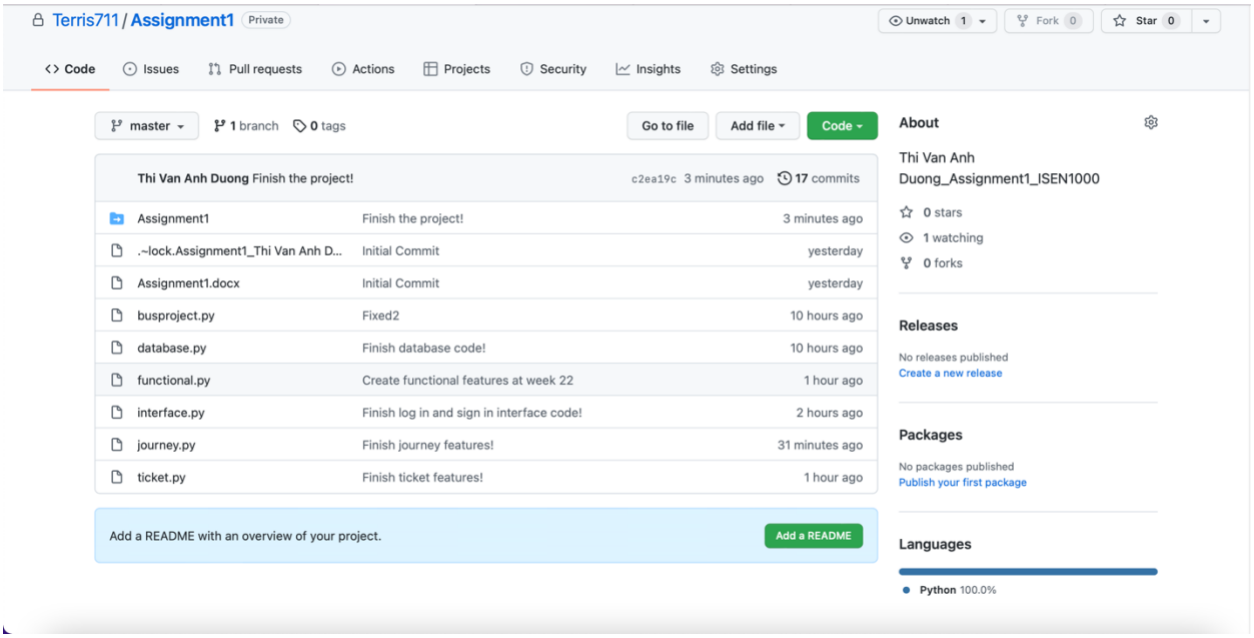
Clone repository:

```
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git branch
  Developer01
  Developer02
  Developer03
  Developer04
  Devloper05
* master
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git clone https://github.com/Terris711/Assignment1.git
Cloning into 'Assignment1'...
Username for 'https://github.com': Terris711
Password for 'https://Terris711@github.com':
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 627 bytes | 89.00 KiB/s, done.
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$
```

Push the project file to GitHub:

```
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git add .
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git commit -m "Finish the project!"
[master c2ea19c] Finish the project!
 1 file changed, 1 insertion(+)
 create mode 160000 Assignment1
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git push origin master
fatal: 'origin' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git push https://github.com/Terris711/Assignment1.git master
Username for 'https://github.com': Terris711
Password for 'https://Terris711@github.com':
To https://github.com/Terris711/Assignment1.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/Terris711/Assignment1.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$ git push https://github.com/Terris711/Assignment1.git master --force
Username for 'https://github.com': Terris711
Password for 'https://Terris711@github.com':
Enumerating objects: 52, done.
Counting objects: 100% (52/52), done.
Delta compression using up to 4 threads
Compressing objects: 100% (52/52), done.
Writing objects: 100% (52/52), 3.53 MiB | 405.00 KiB/s, done.
Total 52 (delta 26), reused 0 (delta 0)
remote: Resolving deltas: 100% (26/26), done.
To https://github.com/Terris711/Assignment1.git
 + df1cc1f...c2ea19c master -> master (forced update)
ccadmin@CCUbuntu64bit:~/ISEN1000/Assignment1$
```

Let's check GitHub:



All of my files in master branch are in GitHub now !