PPA Membership Maintenance System

Sprint Report 3

Capstone Computing Project 2

Group SD07

Semester 2, 2018

Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:	Ronaka	Student ID:	19211817
Other name(s):	Janith		
Unit name:	Capstone Computing Project 2	Unit ID:	ISAD3001
Lecturer / unit coordinator:	Dr. Hannes Herrmann	Tutor:	Ms. Geethanjalie Wimalarathne
Date of submission:	14/09/2018	Which assignment?	Sprint Report 3

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature: _____ Date of signature: 10/09/2018

(By submitting this form, you indicate that you agree with all the above text.)

Table of Contents

1. Introduction

1.1 Group Introduction

All the members in our group have successfully completed the CCP1 in 2017. We enrolled for the CCP2 module in 2018 2nd semester, therefore this project is started in semester 2 of this year (2018). Because of that we have to done the main documentations and tasks of the project such as SRS, task allocation, initial requirement gathering etc. in semester 2. Each of the group members has to do a workload of 2 semesters within this semester, in order to complete the project successfully.

1.2 Project Introduction

PPA Membership Maintenance System is going to be used for membership management and some other important administration tasks such as event planning, donation collecting, accounts handling etc. of past pupil association of Sirimavo Bandaranaike Vidyalaya. Currently all these operations are manually performed by the committee. The main intention of the system is to automate most of those tasks and perform the semi-automated tasks easily and conveniently.

We use MEAN stack to develop this application, JIRA as the project management tool and bitbucket as the online repository. The application has main 4 parts as Membership Services, Accountings, Event Planning and Reporting. These four sections are interconnected with each other as per their functionalities.

2. Progress Update

Sprint 3: 31st August – 14th September

2.1 Allocated Tasks for the Sprint 3

Task ID	Task	Task Status
Task 14.1	New Membership Page Designing	Completed
Task 14.2	New Membership GUI Implementation	Completed
Task 14.3	New Membership Page Validation Methods	Completed
Task 14.4	Membership Database Design	Completed
Task 14.5	New Membership Page CRUD operations	Completed
Task 7.3	Sprint 3 Retrospective	Completed
Task 6.3	Planning the user stories for the sprint 4	Completed

2.2 Planned Tasks for the Sprint 4

Task ID	Task	Time Estimation
Task 15.1	All membership Requests screen Design	1h
Task 15.2	All membership Requests screen Implementation	3h
Task 15.3	Membership Requests Accept Functionality	1h
Task 15.4	Membership Requests Reject Functionality	1h
Task 15.5	All membership Requests Validation Methods	1h
Task 15.6	Detailed membership request screen GUI Implementation	3h
Task 16.1	All Letters Requests GUI Designing & Implementation	3h
Task 16.2	All Letter Requests Screen Filtering Methods	3h
Task 16.5	Letter Request Screen Accept Functionality	2h
Task 16.6	Letter Request Screen Reject Functionality	2h
Task 7.4	Sprint 4 Retrospective	0.5h
Task 6.4	Planning the user stories for the sprint 5	1h
Task 8	Design Meetings	1h

3. Task Break Down

3.1 Task 14.1

(Commits: 99a135a)

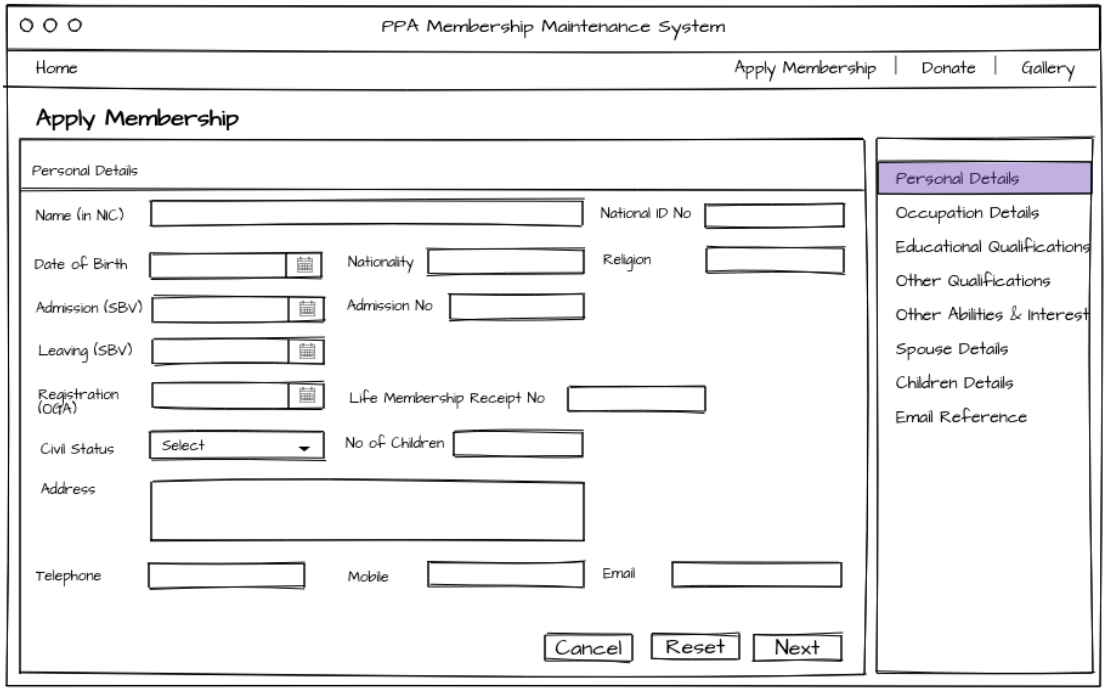
New Membership Page Designing

Estimate Time:	2 Hours
Actual Time:	2 Hours
Actual Time (this sprint):	2 Hours

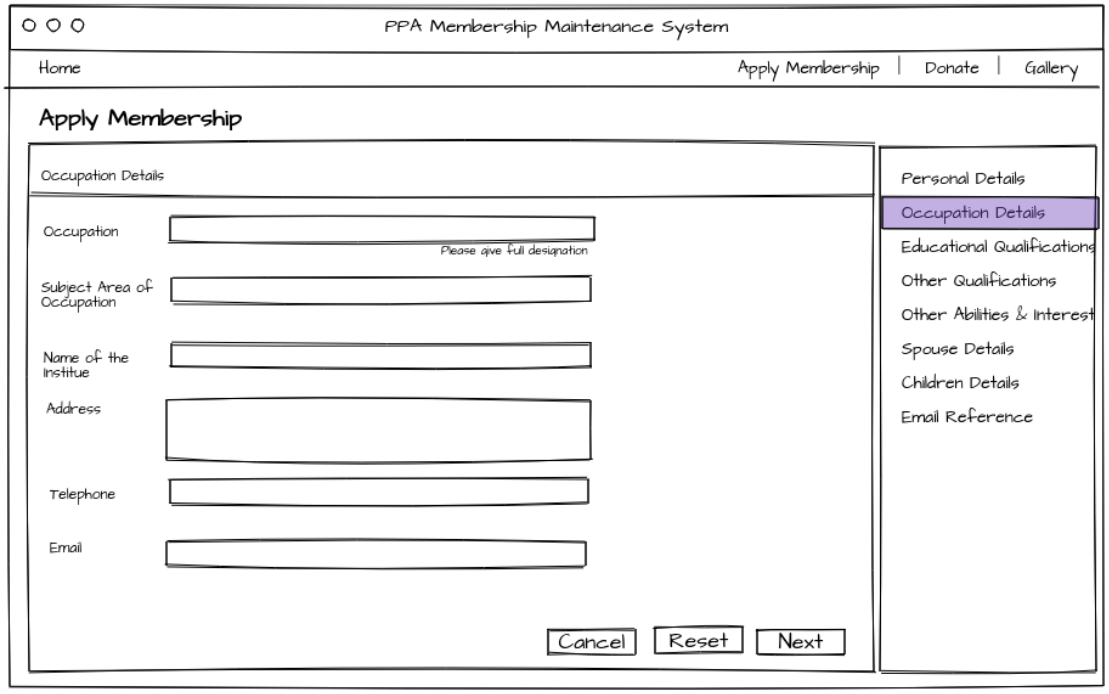
Description

Since this is a past pupil association system the registration process is a lengthy one. The client provided us the registration form which is currently using as a hard copy. That was very descriptive and lengthy document. So I thought to break the existing form into separate sections and navigate the user through a set of steps to fill the required fields. That makes the registration process easy as well as minimize the boring reading part. When designing the GUIs I thought to have the separate sections as separate forms which can be submitted as proceeding.

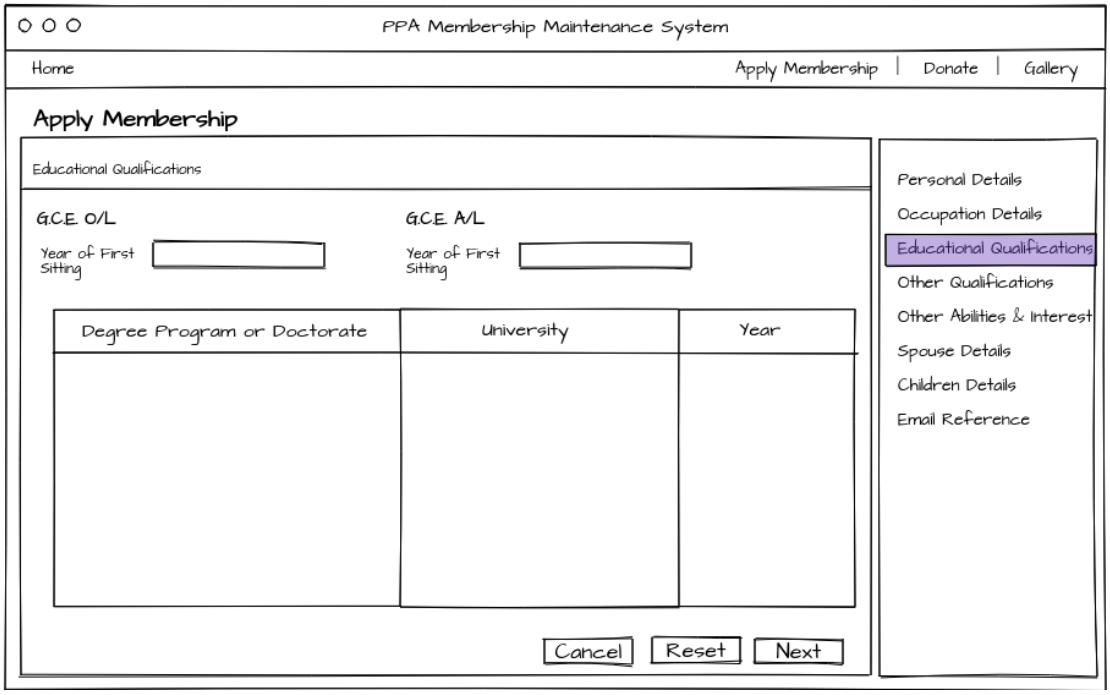
Personal Details Section



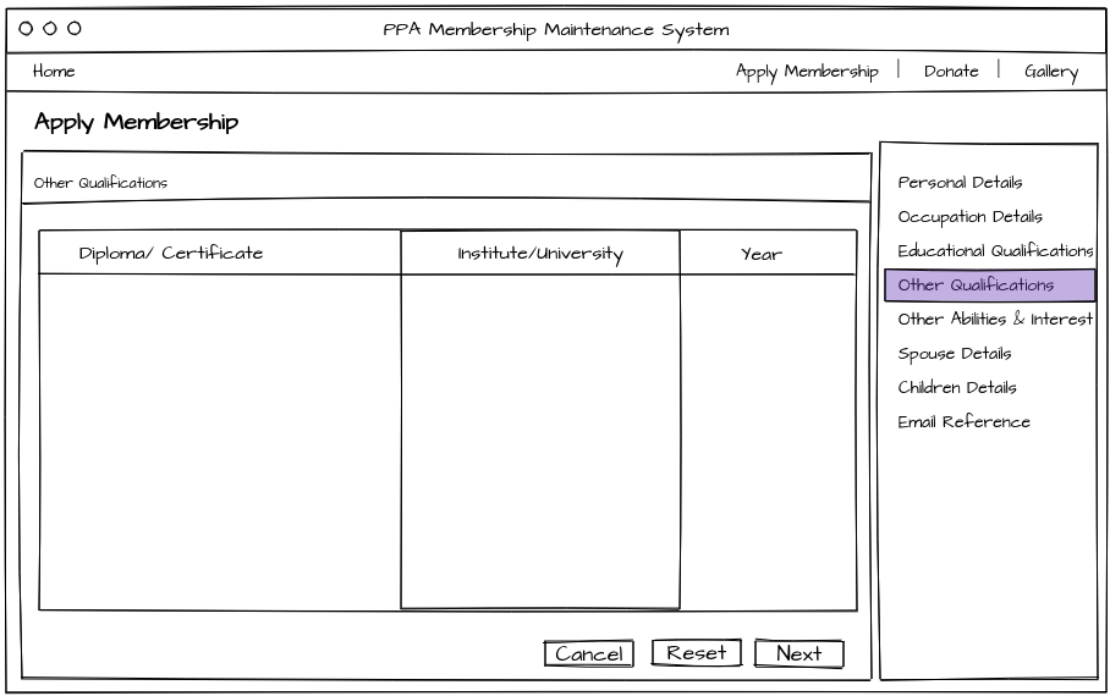
Occupation Details Section



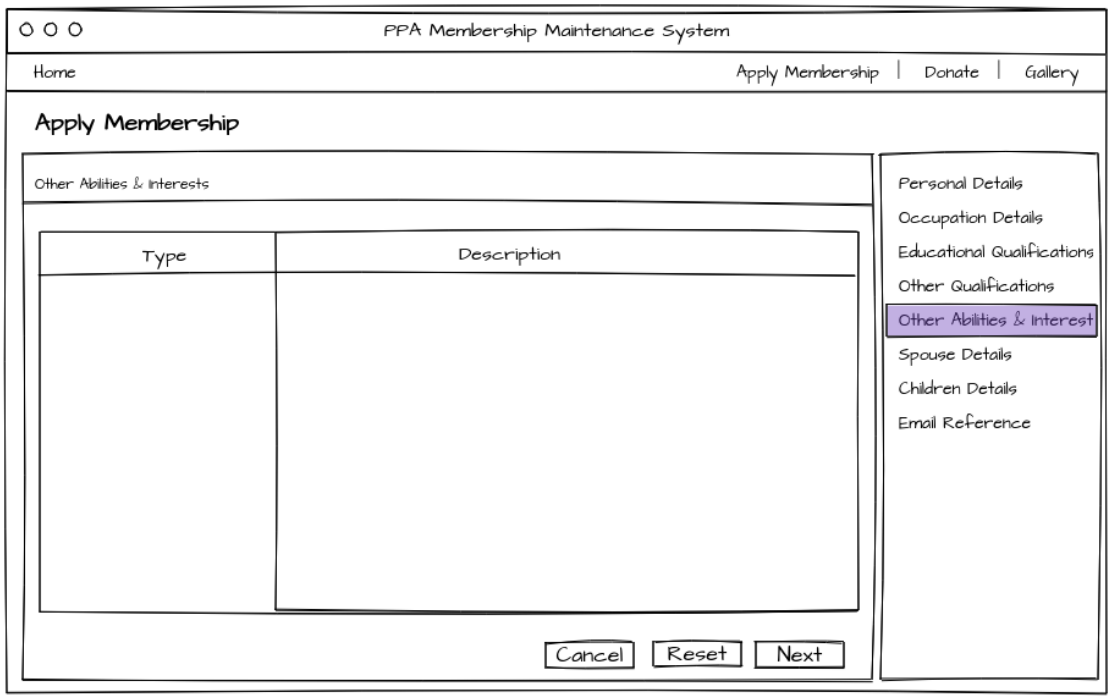
Educational Qualifications Section



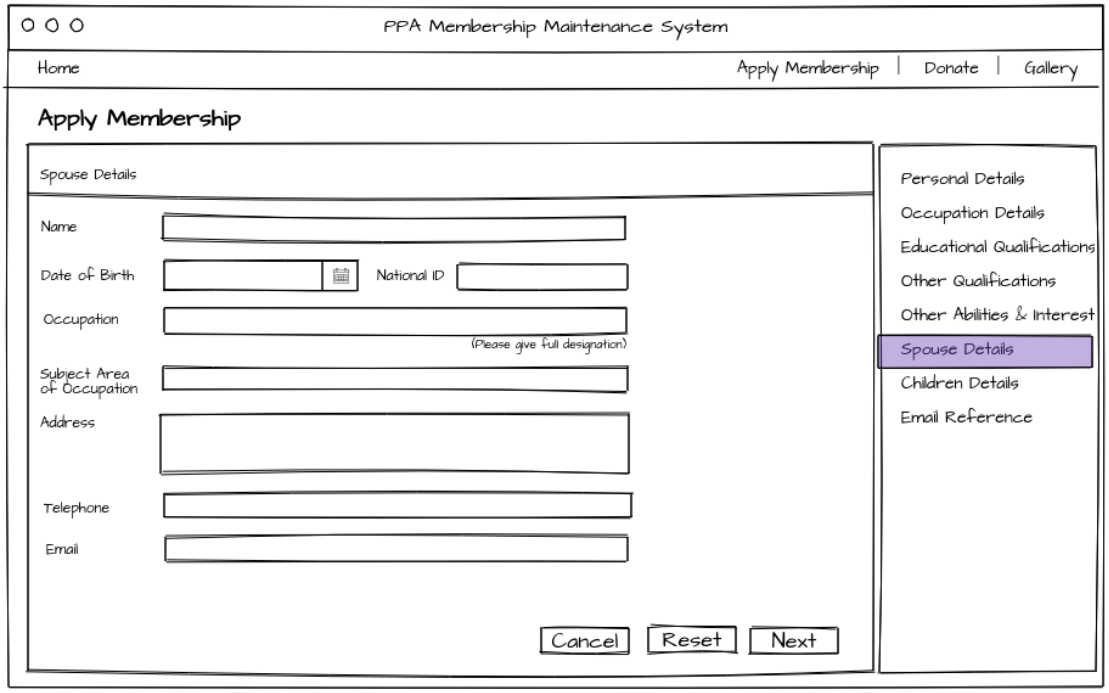
Other Qualifications Section



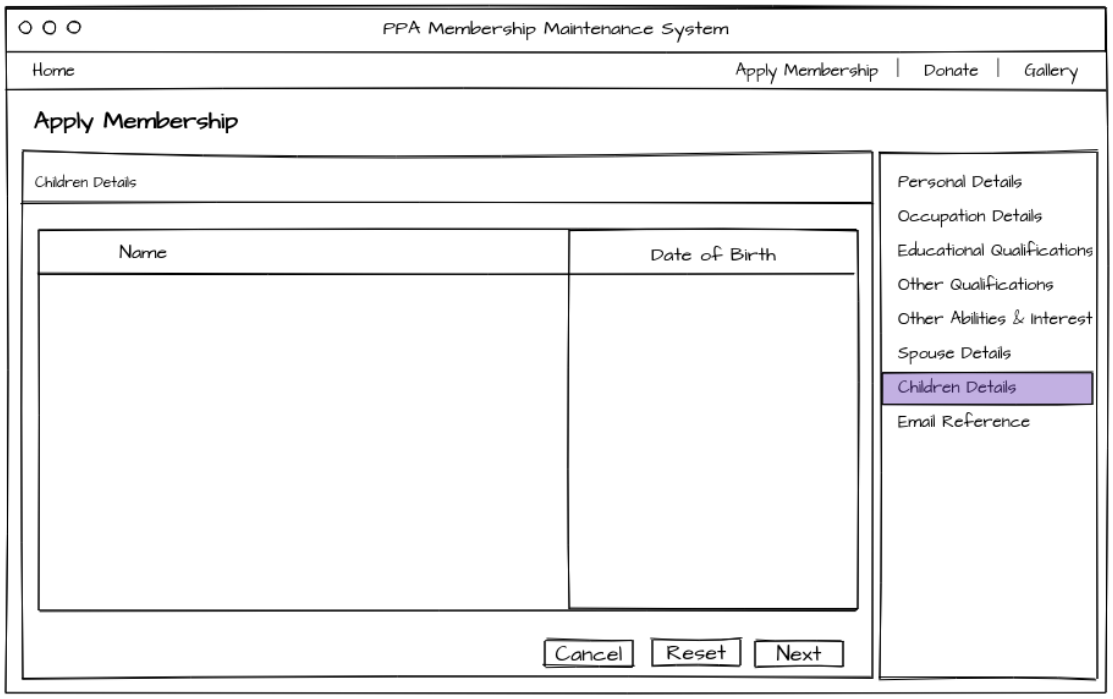
Other Abilities & Interests Section



Spouse Details Section



Children Details Section



3.2 Task 14.2

(Commits: f20b2a2, b1e5eea, 60ac810)

New Membership GUI Implementation

Estimate Time:	12 Hours
Actual Time:	13 Hours
Actual Time (this sprint):	13 Hours

Commits & Build Reports

- f20b2a2
- b1e5eea
- 60ac810

Commits

All branches

Find commits

Author	Commit	Message	Date	Builds
Janith Ronaka	60ac810 M	Merge branch 'master' of https://bitbucket.org/Computing_Projects_SLIIT/2018_sd...	11 hours ago	
Janith Ronaka	b1e5eea	PPA-238: Membership Request GUI Draft Designing	11 hours ago	
Janith Ronaka	f20b2a2	PPA-273: All membership requests screen - GUI	4 days ago	

1. f20b2a2 – Build Report

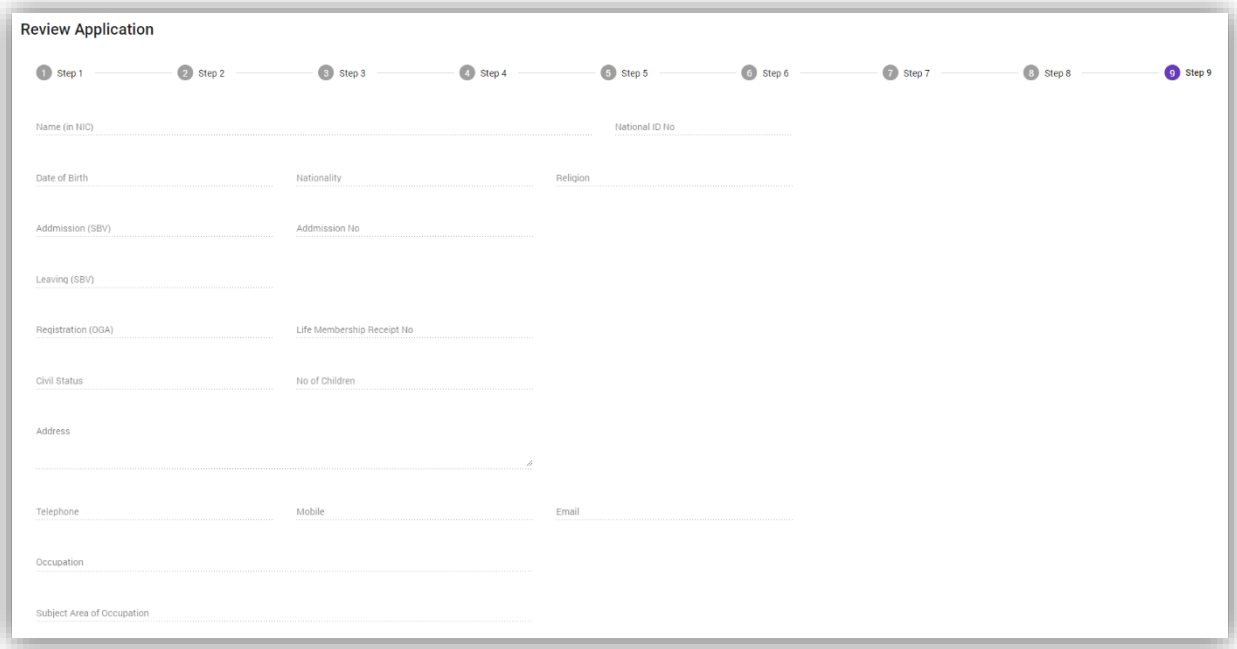
#69 PPA-273: All membership requests screen - GUI
Janith Ronaka f20b2a2 master
Successful 5 days ago 1 min 12 sec

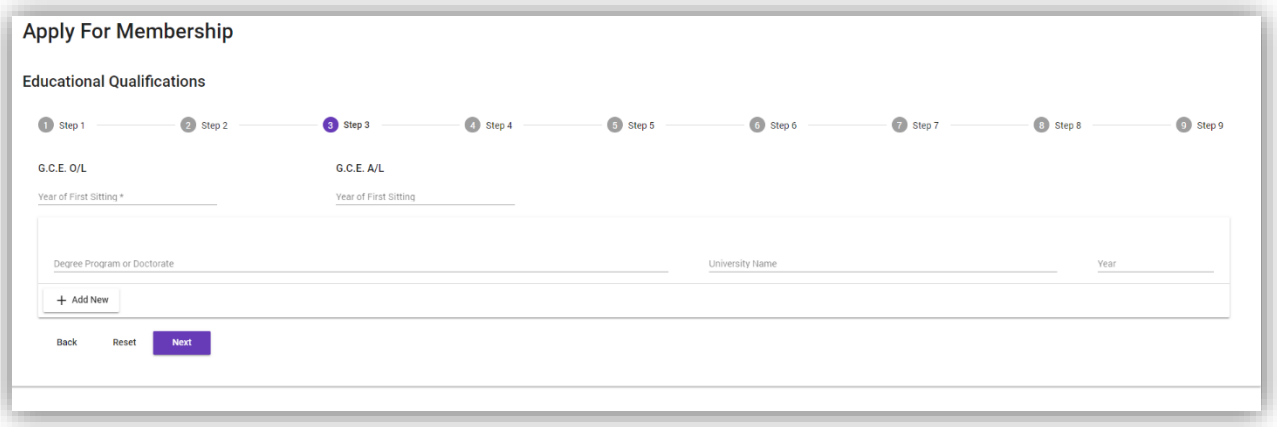
2. 60ac810 – Build Report

Pipeline	Status	Started	Duration
#77 Merge branch 'master' of https://bitbucket.org/Computing_Proje... Janith Ronaka 60ac810 master	Successful	12 hours ago	1 min 14 sec

Description

I changed the initial wireframe design when I implementing the GUI. When designing the wireframe I thought to show the current editing category of the form at the right side of the screen in a list box. But when implementing I thought it's better to use a stepper component to traverse through the application form since that is very clear and modern. However I had to do bit of learning to implement the stepper functionality and resolve the bugs that rose when implementing. After the implementation the





Technical Information

For this task I used the mainly material stepper and its relevant components. Following are the new components relevant to this implementation.

- membership-request.component.html
- membership-request.component.ts

membership-request.component.html

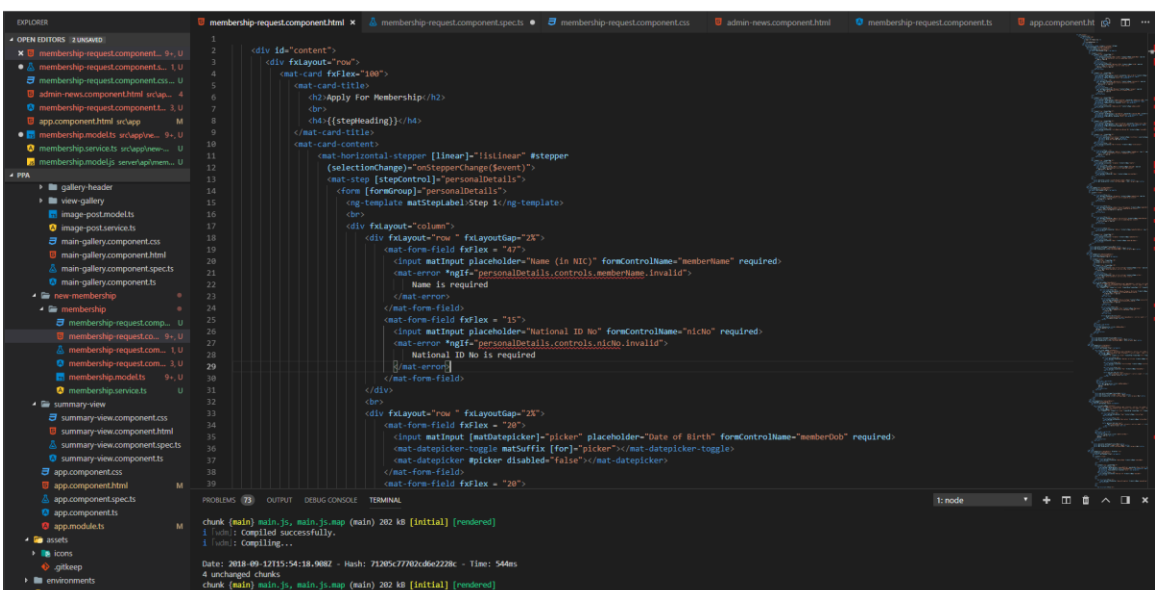


Figure 1:membership-request.component.html

I used the following links as learning materials when implementing the GUI.

https://material.angular.io/components/stepper/overview

https://blog.karmacomputing.co.uk/angular-6-dynamically-add-rows-reactive-forms-how-to/

https://angular.io/guide/reactive-forms

At first I thought to implement each section of the application form as separate components. But if the user leave the application without completing the whole registration form it's a waste of space saving an incomplete registration information in the database. That was the main point I went to a stepper functionality. So the application saves the registration data of the user only if she has completed all the required data fields. The last step of the stepper I implemented as a reviewing screen. It displays all the data entered by the user in the previous steps in non-editable fields. If user needs to change any of the data in there she can just go back to the relevant step in the stepper and change it and comeback to the final step.

membership-request.component.ts

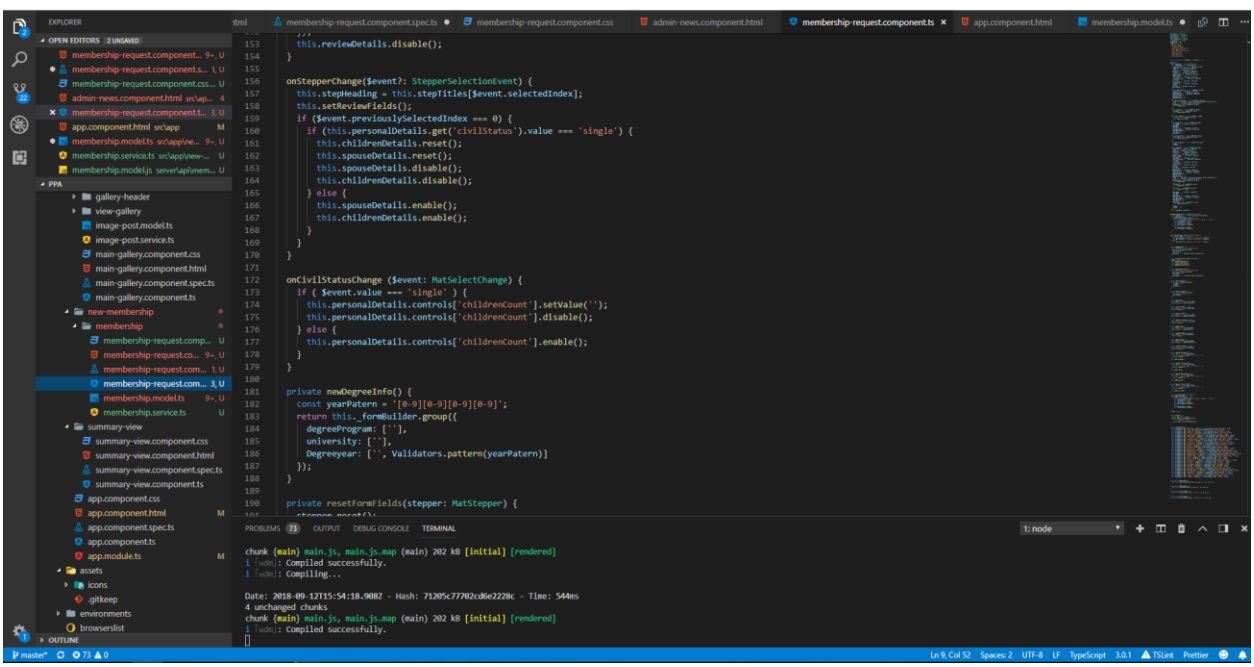


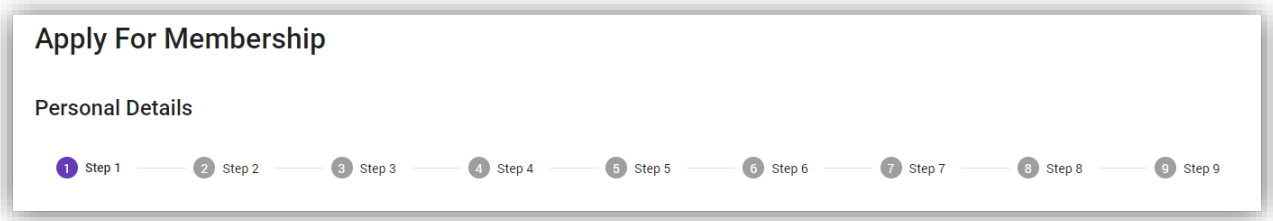
Figure 2: membership-request.component.ts

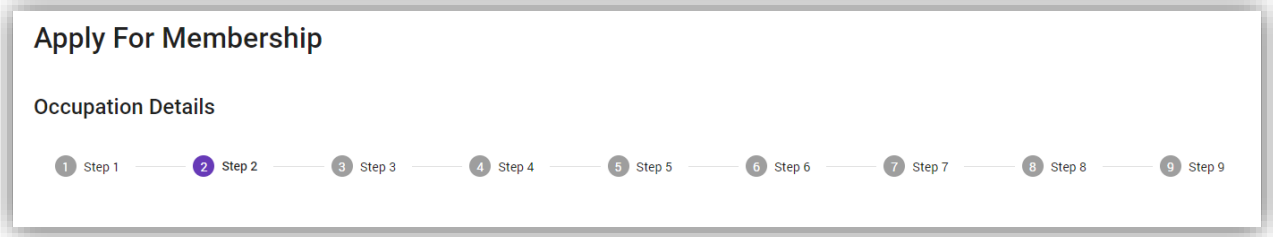
Since I have used reactive form approach for the registration form I made separate forms groups for each section. All the form groups are initialize when the coponent initializes (via *ngOnInit* method). The review fields are updated each time the user changes the current step. Each dynamic field in the registration form is generated through the custom methods such as *newInterestsInfo(), newDegreeInfo(), newOtherQualifInfo()* etc.

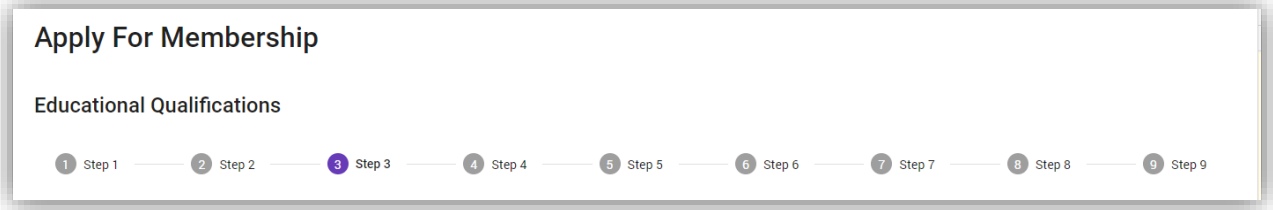
Testing

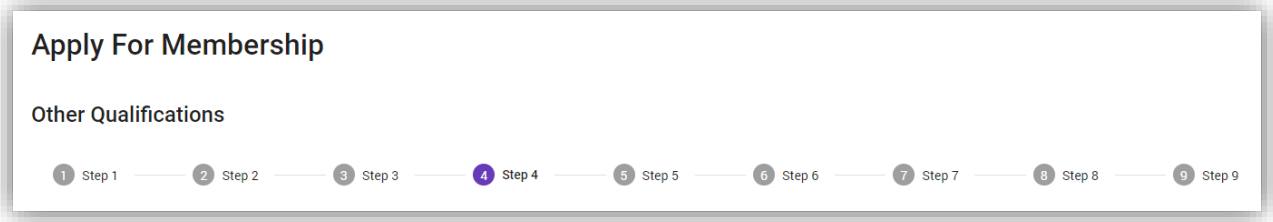
Test Case 1

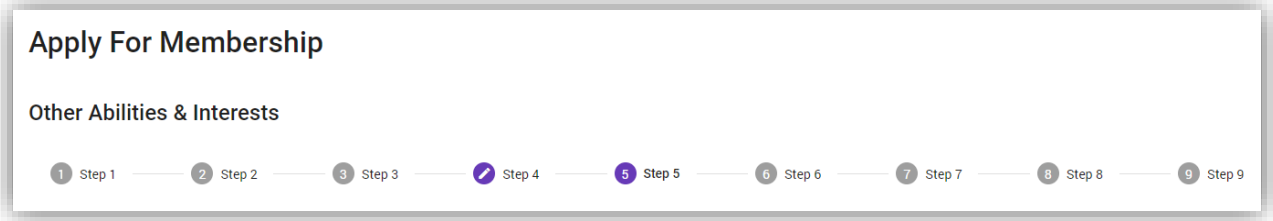
Test Case	Current Step Name should be visible at the top of the page
Expected Behavior	When the user navigate through the stepper each step should change the current heading at the top of the page
Test Steps	a) Navigate through each of the step in the stepper by clicking Next & Back buttons b) Navigate through the steps by clicking the relevant step c) Navigate through the steps by using both above methods
Test Status	**Passed**





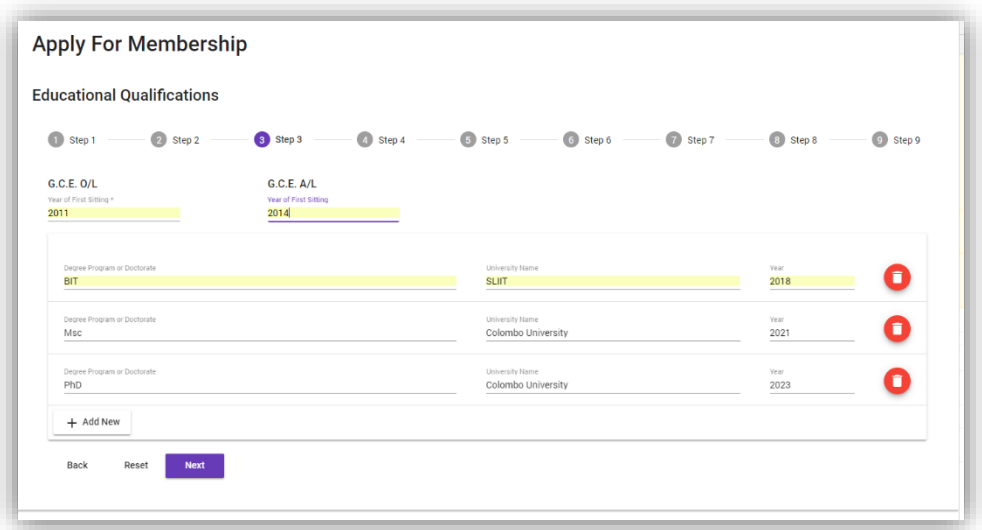


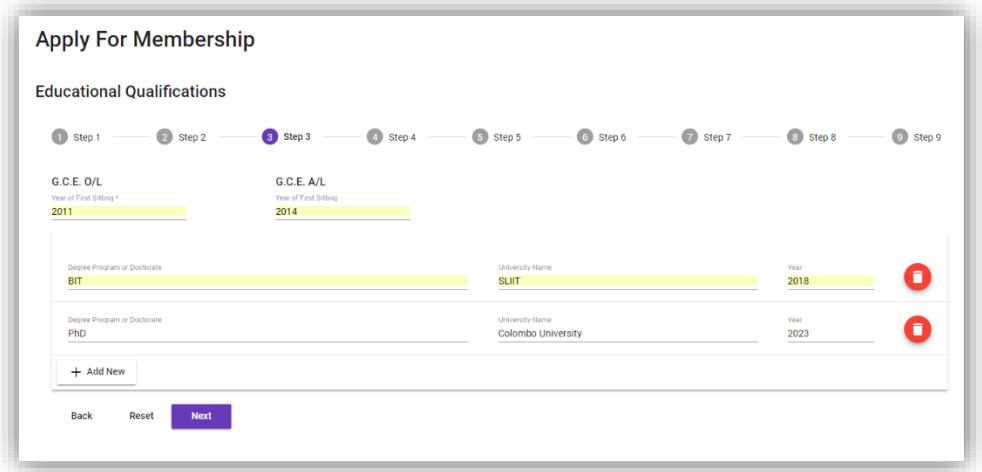




Test Case 2

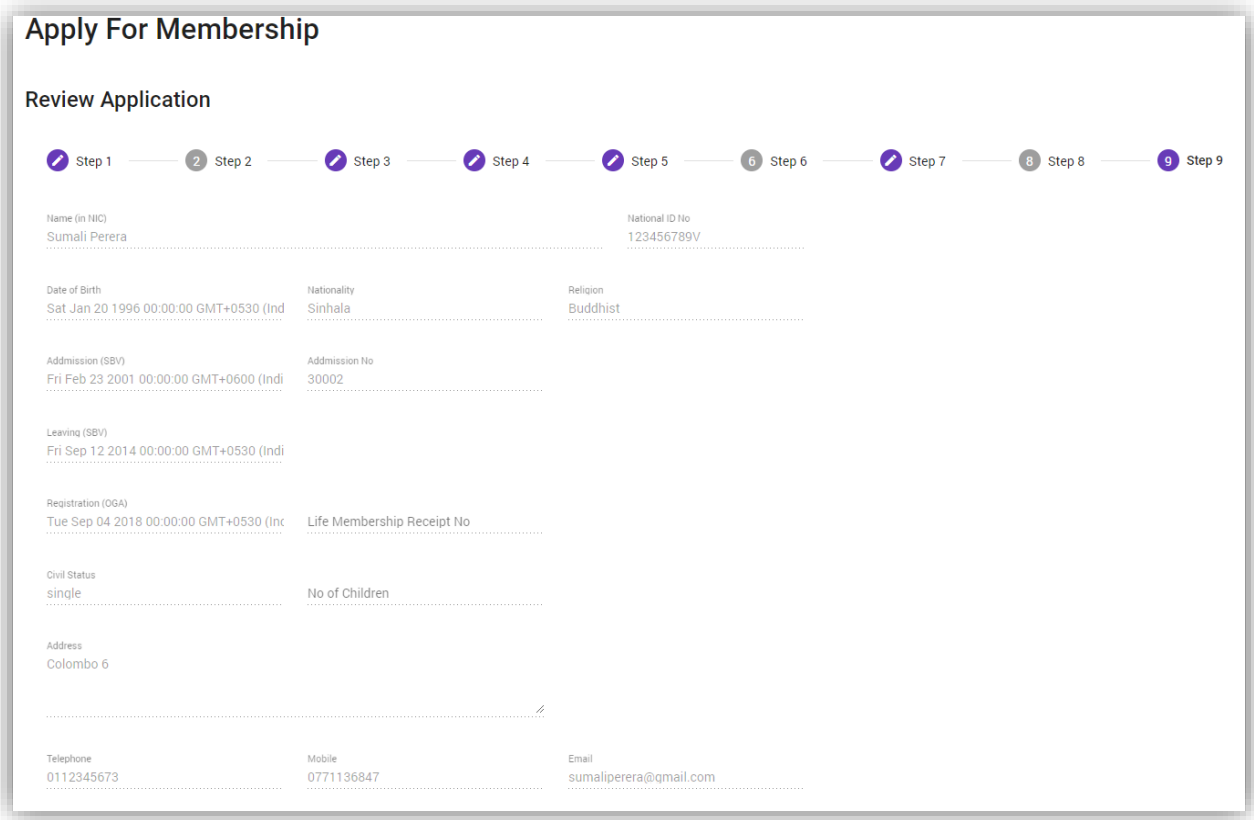
Test Case	New rows should be added and removed in the Educational Qualifications step
Expected Behavior	A new row should be added in the Educational Qualifications step when click on the "Add New" button Once the row is added there should be delete buttons for all the rows When the user click the delete button of a particular row that row should be deleted immediately
Test Steps	a) Navigate to the Educational Qualifications Step b) Click on the "Add New" button c) Observe the newly added row and the delete button at the right side of the row d) Click the delete button e) Observe the row is deleted
Test Status	**Passed**





Test Case 3

Test Case	All the data entered in the steps should be reflected in the review application step in real time
Expected Behavior	Once the data inserted into the form fields in each step, all those data should be added and visible in the final step "Review Application"
Test Steps	a) Enter the required data in each step b) Click on the last step in the stepper c) Observe all the new data is visible in the review application step
Test Status	**Passed**



3.3 Task 14.3

(Commits: fde46b5, 318c804)

New Membership Page Validation

Estimate Time:	2 Hours
Actual Time:	3 Hours
Actual Time (this sprint):	3 Hours

Commits & Build Reports

- fde46b5
- 318c804

Commits

All branches

Find commits

Author	Commit	Message	Date	Builds
Janith Ronaka	318c804 M	Merge branch 'master' of https://bitbucket.org/Computing_Projects_SLIIT/2018_sd07 # Conflicts: # Application/ppa/src/app/app.component.html # Application/ppa/src/app/...	35 seconds ago	
Janith Ronaka	fde46b5	PPA-276: Detailed membership request screen - Implementation	3 minutes ago	

1. 318c804 – Build Report

Pipeline	Status	Started	Duration
#81 Merge branch 'master' of https://bitbucket.org/Computing_Proje... Janith Ronaka 318c804 master	Successful	8 minutes ago	1 min 32 sec

Description

Since I have used the Reactive Forms approach for this implementation, all the validations have declared in the ts file form group. Mainly I used only the required validation, but also used a pattern validation to detect errors in the numerical fields. There was also some custom validations such as when the civil status is set as Single the No of children field get disabled, when the children count set as 0 or null the Children Details step get disabled etc.

Technical Details

For the required validation I used the in-built validation in the angular core Validator component. Each required field is marked in the form group as below.

membership-request.component.ts × | membership.model.ts ● | membership.service.ts | membership-request.component.htm

```
38    constructor(private _formBuilder: FormBuilder) { }
39
40    ngOnInit() {
41      const yearPatern = '[0-9][0-9][0-9][0-9]';
42      this.stepHeading = this.stepTitles[0];
43      this.personalDetails = this._formBuilder.group({
44        memberName: ['', Validators.required],
45        nicNo: ['', Validators.required],
46        memberDob: ['', Validators.required],
47        nationality: ['', Validators.required],
48        religion: ['', Validators.required],
49        addmisionDate: ['', Validators.required],
50        addmissionNo: [''],
51        leavingDate: ['', Validators.required],
52        registrationDate: ['', Validators.required],
53        receiptNo: [''],
54        civilStatus: ['', Validators.required],
55        childrenCount: [''],
56        homeAddress: ['', Validators.required],
57        homeTel: [''],
58        mobileNo: ['', Validators.required],
59        personalEmail: ['']
60      });
61      this.occupationDetails = this._formBuilder.group({
62        memberOccup: ['', Validators.required],
63        memberWorkArea: ['', Validators.required],
64        memberEmployer: ['', Validators.required],
65        memberWorkAddress: [''],
```

Figure 3: membership-request.component.ts

I call to the method onCivilStatusChange() when the selection is changed in the civil status field. That method checks whether the civil status is Single or Married and set enable the Spouse Details and Children Detail steps accordingly.

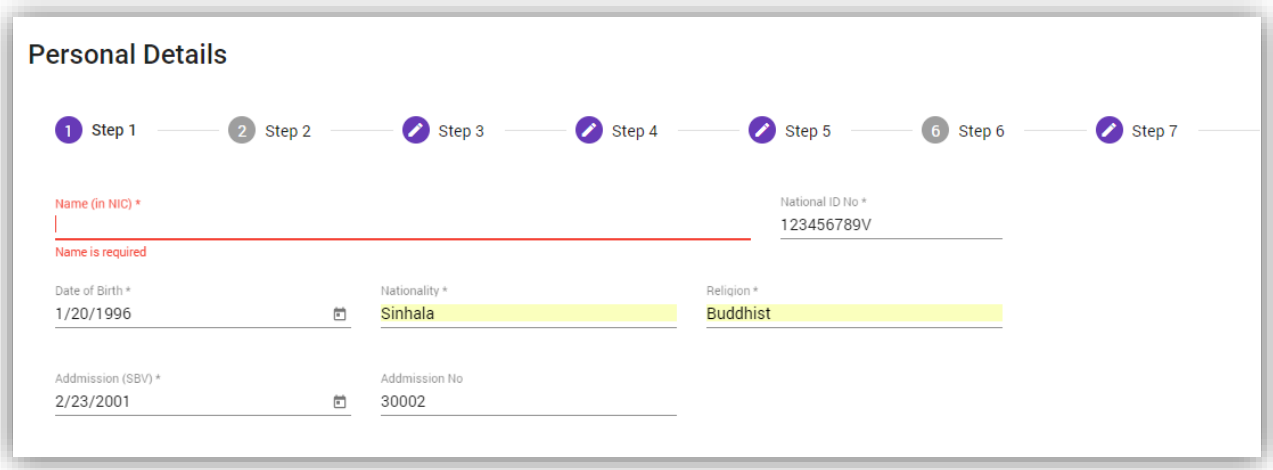
```
onCivilStatusChange ($event: MatSelectChange) {
  if ( $event.value === 'single' ) {
    this.personalDetails.controls['childrenCount'].setValue('');
    this.personalDetails.controls['childrenCount'].disable();
  } else {
    this.personalDetails.controls['childrenCount'].enable();
  }
}
```

Figure 4: membership-request.component.ts

Testing

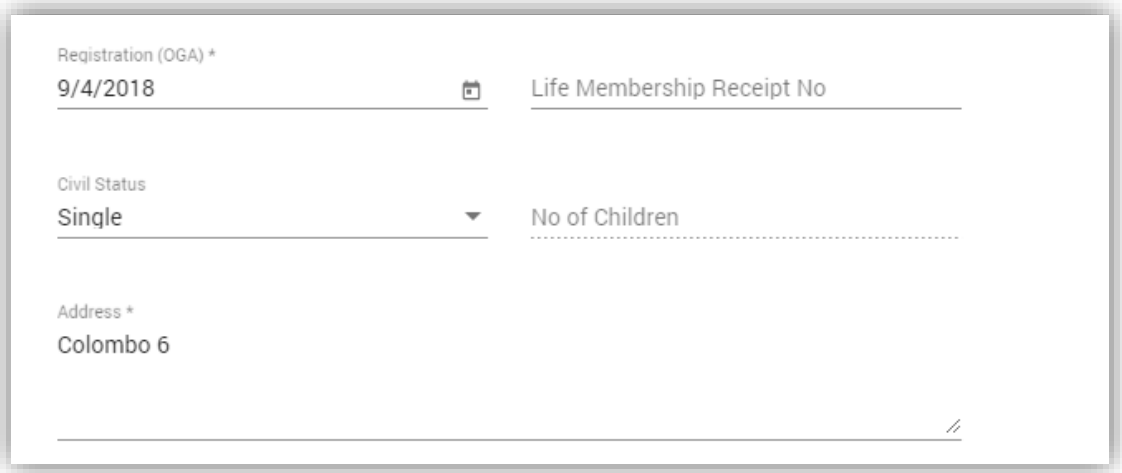
Test Case 1

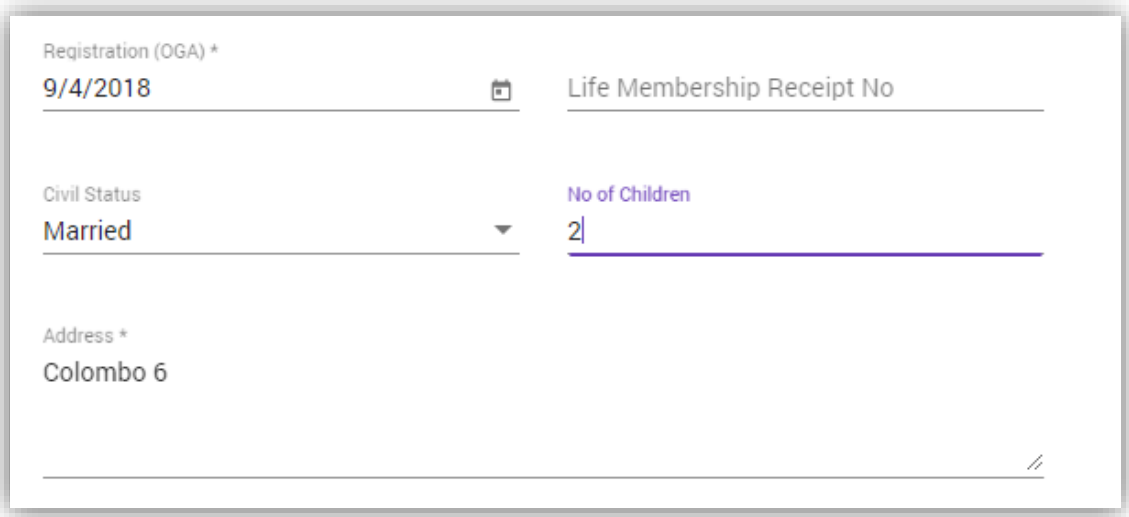
Test Case	Required fields should be validated in real time
Expected Behavior	Once a required field is focused and left without entering any data into it that should show and error text under the field.
Test Steps	a) Set the cursor on a required field in any step b) Now without entering anything click on another field c) Observe the error message under the required field
Test Status	**Passed**



Test Case 2

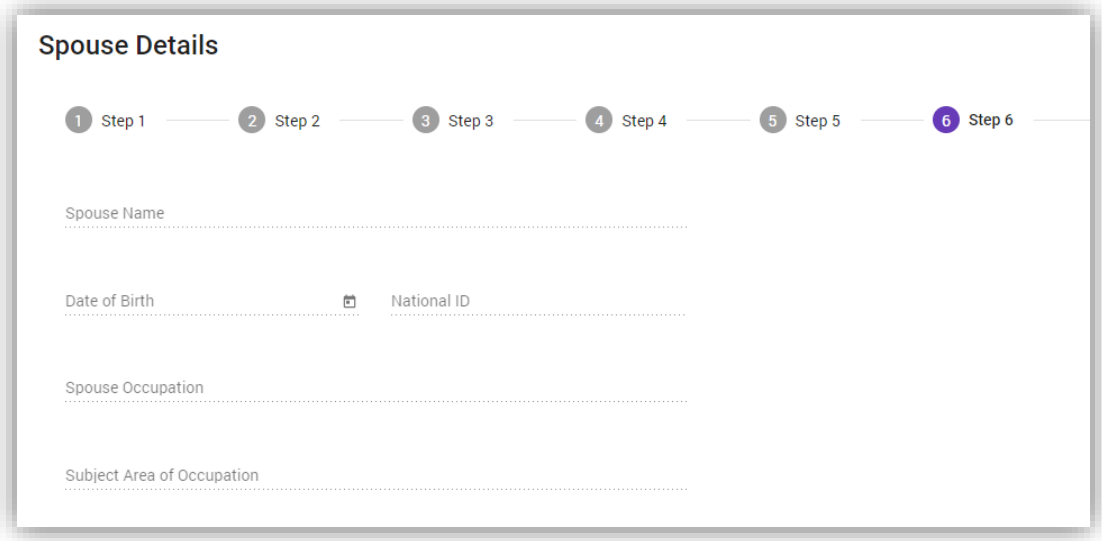
Test Case	Check validations for No of Children Field
Expected Behavior	If the civil status set as Single, No of Children field should be disabled
Test Steps	a) Set the Civil Status in the step 1 as Single b) Observe the disabled No of Children field c) Now set the Civil Status as Married d) Observe the enabled No of Children field
Test Status	**Passed**

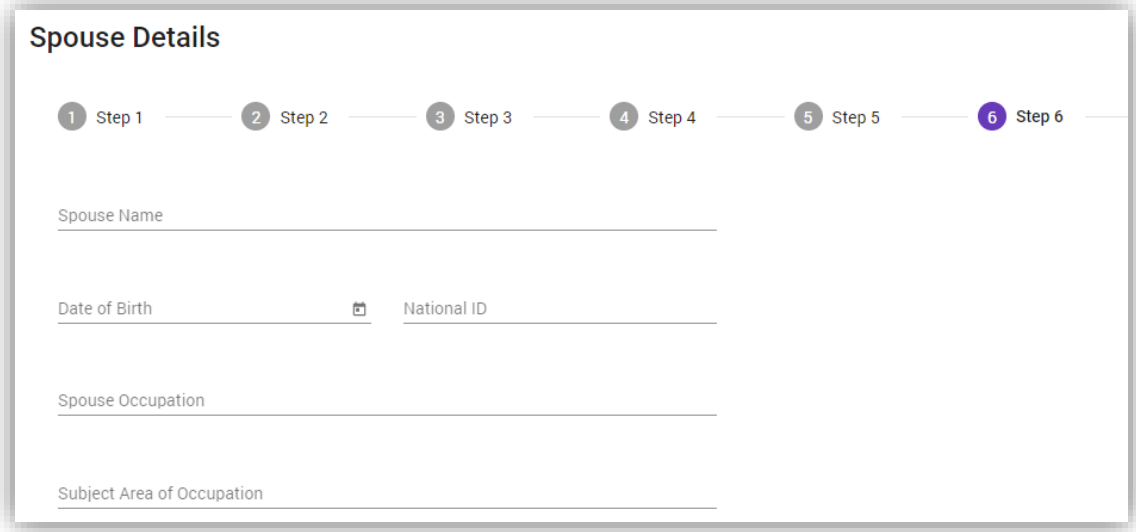




Test Case 3

Test Case	Check validations for Spouse Details Step
Expected Behavior	If the civil status set as Single, Spouse Details step should be disabled
Test Steps	a) Set the Civil Status in the step 1 as Single b) Observe the disabled fields in the Spouse Details step c) Now set the Civil Status as Married d) Observe the enabled and editable fields in the Spouse Details step
Test Status	**Passed**





3.4 Task 14.4

(Commits: fde46b5, 318c804)

Membership Database Design

Estimate Time:	1 Hours
Actual Time:	1 Hours
Actual Time (this sprint):	1 Hours

Commits & Build Reports

- fde46b5
- 318c804

Commits

All branches

Find commits

Author	Commit	Message	Date	Builds
Janith Ronaka	318c804 M	Merge branch 'master' of https://bitbucket.org/Computing_Projects_SLIIT/2018_sd07 # Conflicts: # Application/ppa/src/app/app.component.html # Application/ppa/src/app/...	35 seconds ago	
Janith Ronaka	fde46b5	PPA-276: Detailed membership request screen - Implementation	3 minutes ago	

2. 318c804 – Build Report

Pipeline	Status	Started	Duration
#81 Merge branch 'master' of https://bitbucket.org/Computing_Proje... Janith Ronaka 318c804 master	Successful	8 minutes ago	1 min 32 sec

Description

Following are the server files that were related to this development.

membership-request.component.ts | membership.model.ts | membership.service.ts | membership.model.js | membership-request.component.html

```
var mongoose = require('../../dbConfig');
var Schema = mongoose.Schema;

var MemberSchema = new Schema({
  dataType: {
    type: String,
    required: false,
    enum: ['Profile', 'Request', 'Rejected'],
    default: 'Request'
  },
  memberName: {
        type: String,
        required: true
    },
    nicNo: {
        type: String,
        required: true
    },
    memberDob: {
        type: String,
        required: false
    },
    nationality: {
      type: String,
      required: true
    },
    religion: {
        type: String,
        required: true
    },
    addmisionDate: {
      type: String,
```

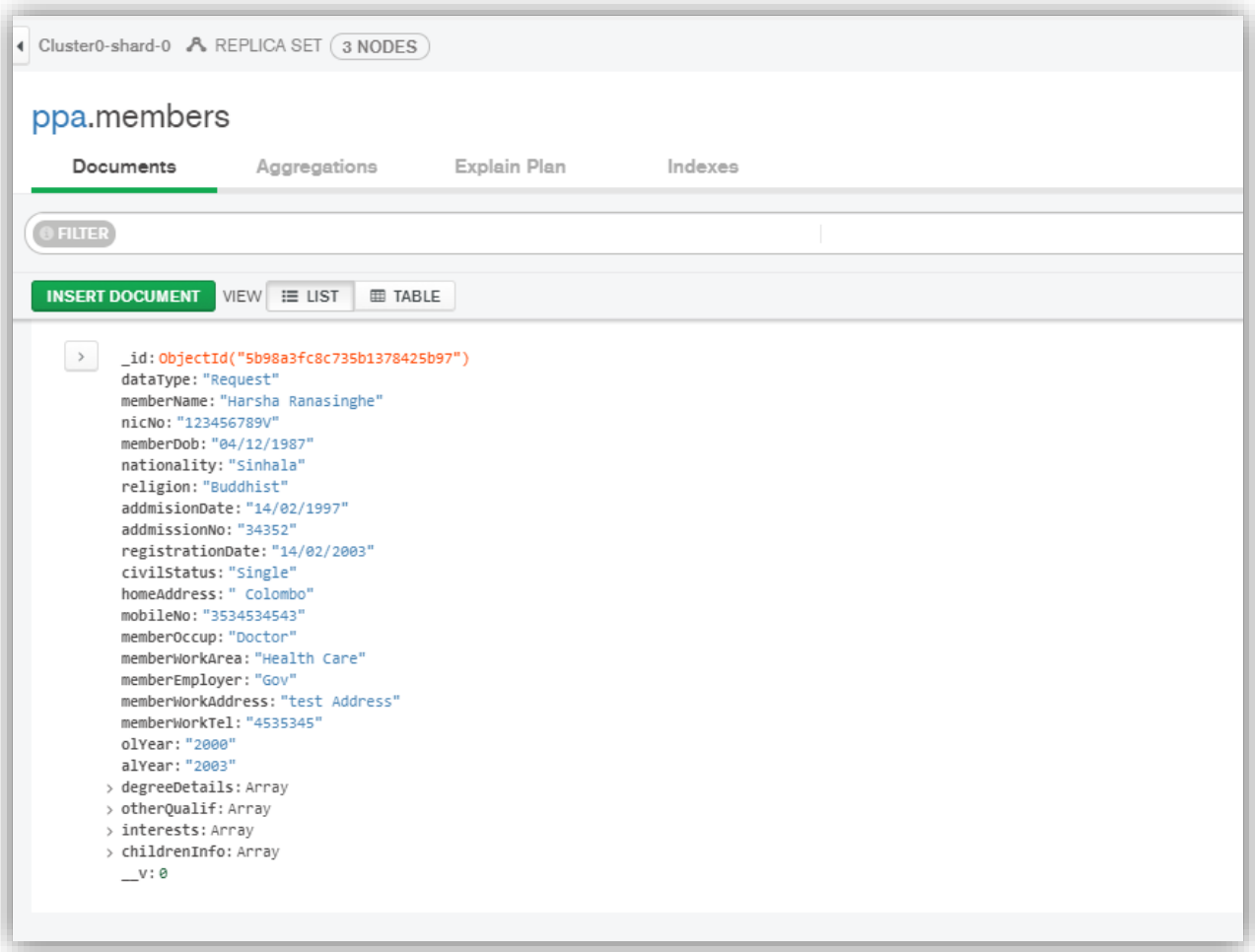
Figure 5: membership.model.js

In the membership.model.js file I defined the structure for the database table.

Testing

Test Case 1

Test Case	Check the database table creation
Expected Behavior	When a membership object is inserted for the first time, Members collection should be generated
Test Steps	a) Prepare a post request to insert a data into the members collection b) Set the request type as "POST" c) Click the Send button d) Observe thenewly created collection in the MongoDB Compass
Test Status	**Passed**



3.5 Task 14.5

(Commits: fde46b5, 318c804)

New Membership Page CRUD Operations

Estimate Time:	6 Hours
Actual Time:	5 Hours
Actual Time (this sprint):	5 Hours

Commits & Build Reports

- fde46b5
- 318c804

Commits

All branches

Find commits

Author	Commit	Message	Date	Builds
Janith Ronaka	318c804 M	Merge branch 'master' of https://bitbucket.org/Computing_Projects_SLIIT/2018_sd07 # Conflicts: # Application/ppa/src/app/app.component.html # Application/ppa/src/app/...	35 seconds ago	
Janith Ronaka	fde46b5	PPA-276: Detailed membership request screen - Implementation	3 minutes ago	

3. 318c804 – Build Report

Pipeline	Status	Started	Duration
#81 Merge branch 'master' of https://bitbucket.org/Computing_Proje... Janith Ronaka 318c804 master	Successful	8 minutes ago	1 min 32 sec

Description

I wrote the addMembership() method to insert the new records to the database. In the addMembership() I call to the generateMembershipObj() method to create the Membership type object with the values passed from the front end. Then I pass that object to the back end insert method.

```
18  addMembership(formData: FormGroup) {
19    this.http.post<{ status: any, message: any }>('http://localhost:4200/api/membership',
20                                                  this.generateMembershipObj(formData)).subscribe();
21  }
22
23  generateMembershipObj(membershipForm: FormGroup): Membership {
24    const membershipObj: Membership = {
25      _id: null,
26      memberName: membershipForm.get('memberName').value,
27      nicNo: membershipForm.get('nicNo').value,
28      memberDob: membershipForm.get('memberDob ').value,
29      nationality: membershipForm.get('nationality').value,
30      religion: membershipForm.get('religion').value,
31      addmisionDate: membershipForm.get('addmisionDate').value,
32      addmissionNo: membershipForm.get('addmissionNo').value,
33      leavingDate: membershipForm.get('leavingDate').value,
34      registrationDate: membershipForm.get('registrationDate').value,
35      receiptNo: membershipForm.get('receiptNo').value,
36      civilStatus: membershipForm.get('civilStatus').value,
37      childrenCount: membershipForm.get('childrenCount').value,
38      homeAddress: membershipForm.get('homeAddress').value,
39      homeTel: membershipForm.get('homeTel').value,
40      mobileNo: membershipForm.get('mobileNo').value,
41      personalEmail: membershipForm.get('personalEmail').value,
42      memberOccup: membershipForm.get('memberOccup').value,
43      memberWorkArea: membershipForm.get('memberWorkArea').value,
44      memberEmployer: membershipForm.get('memberEmployer').value,
45      memberWorkAddress: membershipForm.get('memberWorkAddress').value,
```

Figure 6: membership.service.ts

I use this interface to store the registration form data. Without sending a form data object I add the form field data into this interface and send that to the database. The following MembershipService class acts as a bridge between the backend and front by doing db transactions in-between the two ends.

```
1   export interface Membership {
2     _id: string;
3     memberName: string;
4     nicNo: string;
5     nationality: string;
6     addmisionDate: string;
7     addmissionNo: string;
8     leavingDate: string;
9     registrationDate: string;
10    receiptNo: string;
11    civilStatus: string;
12    childrenCount: number;
13    homeAddress: string;
14    homeTel: string;
15    mobileNo: string;
16    personalEmail: string;
17    memberOccup: string;
18    memberWorkArea: string;
19    memberEmployer: string;
20    memberWorkAddress: string;
21    memberWorkTel: string;
22    memberWorkEmail: string;
23    olYear: number;
24    alYear: number;
25    degreeDetails: [{
26      degreeProgram: string,
27      university: string,
28      degreeYear: number
29    }];
30    otherQualif: [{
31      diplomaName: string,
32      institute: string,
33      diplomaYear: string
```

Figure 8: membership.model.ts

Following are the server files that were related to this development.

membership-request.component.ts | membership.model.ts | membership.service.ts | membership.service.js | membership-request.component.html

```
var mongoose = require('./membership.model');
var MemberSchema = mongoose.model('Member');

function Controller() {

    this.insert = (data) => {
        return new Promise((resolve, reject) => {
        var MemberSchema = mongoose.model('Member');
            var membership = new MemberSchema(data);
            membership.save().then((newAccount) => {
                console.log('Data Inserted')
                resolve({
                    status: 201,
                    message: newAccount
                })
            }).catch((err) => {
                console.log()
                reject({
                    status: 500,
                    message: "Error:- " + err
                })
            })
        })
    };

    this.getAll = () => {
        return new Promise((resolve, reject) => {
          MemberSchema.find().then(data => {
                resolve({
                    status: 200,
                    message: data
                });
            }).catch((err) => {
```

Figure 7: membership.service.js

All the direct CRUD operation methods with the database table have written in the membership.service.js file

membership-request.component.ts | membership.model.ts | membership.service.ts | membership.route.js | membership-request.component.html

```
var express = require('express');
var router = express.Router();

var MemberController = require('./membership.controller');

router.get('/all', (req, res) => {
  MemberController.getAll().then((data) => {
        res.setHeader('Content-Type', 'application/json');
        res.status(data.status).send(JSON.stringify(data.message));
    }).catch((err) => {
        console.log(err);
        res.status(err.status).send(err.message);
    })
})

router.get('/:id', (req, res) => {
  MemberController.get(req.params.id).then((data) => {
        res.setHeader('Content-Type', 'application/json');
        res.status(data.status).send(JSON.stringify(data.message));
    }).catch((err) => {
        console.log(err);
        res.status(err.status).send(err.message);
    })
})

router.post('/', (req, res) => {
  MemberController.insert(req.body).then((data) => {
        res.status(data.status).send(data.message);
    }).catch((err) => {
        res.status(err.status).send(err.message);
    })
})
```

Figure 8: membership.route.js

Method routing in the membership server files have configured in the membership.route.js.

All the business logic and the wrapper methods of CRUD operations have written in the membership.controller.js.

membership-request.component.ts | membership.model.ts | membership.service.ts | membership.controller.js | membership-request.component.html

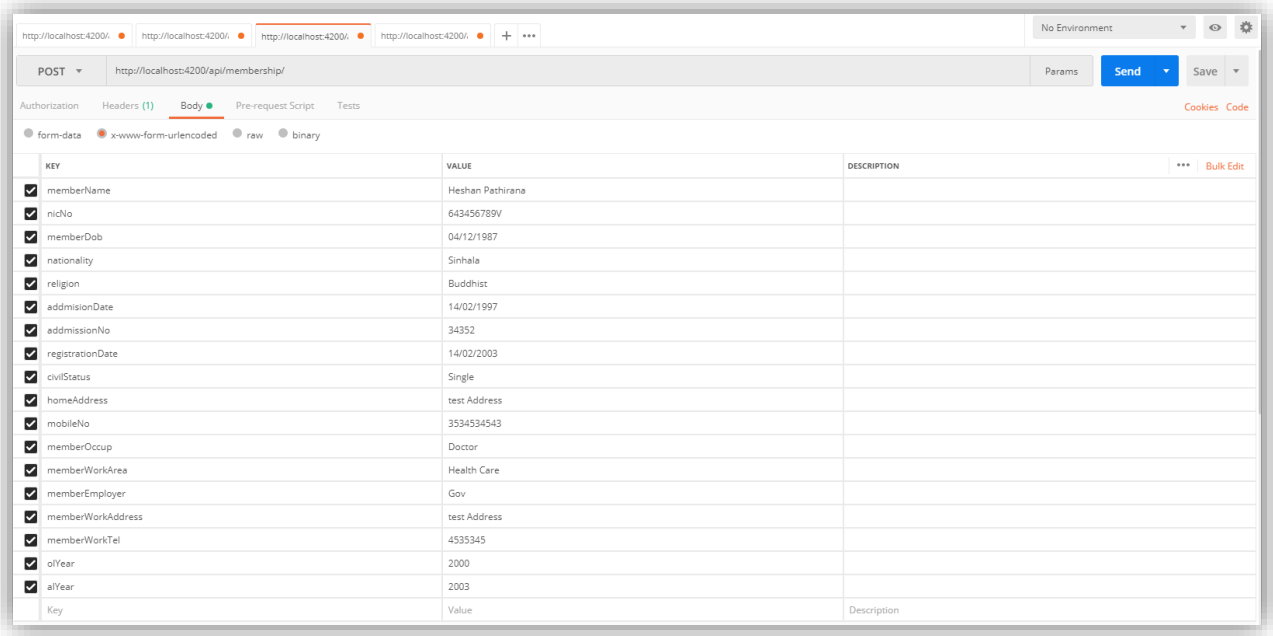
```javascript
var MemberSchema = require('./membership.service');

function Controller() {
    this.insert = (data) => {
        var response = MemberSchema.insert(data);
        return response;
    };

    this.getAll = (filter) => {
        var response = MemberSchema.getAll(filter);
        return response;
    };

    this.get = (id) => {
        var response = MemberSchema.get(id);
        return response;
    }

    this.update = (data) => {
        var response = MemberSchema.update(data);
        return response;
    }

    this.delete = (id) => {
        var response = MemberSchema.delete(id);
        return response;
    }
}

module.exports = new Controller();
```

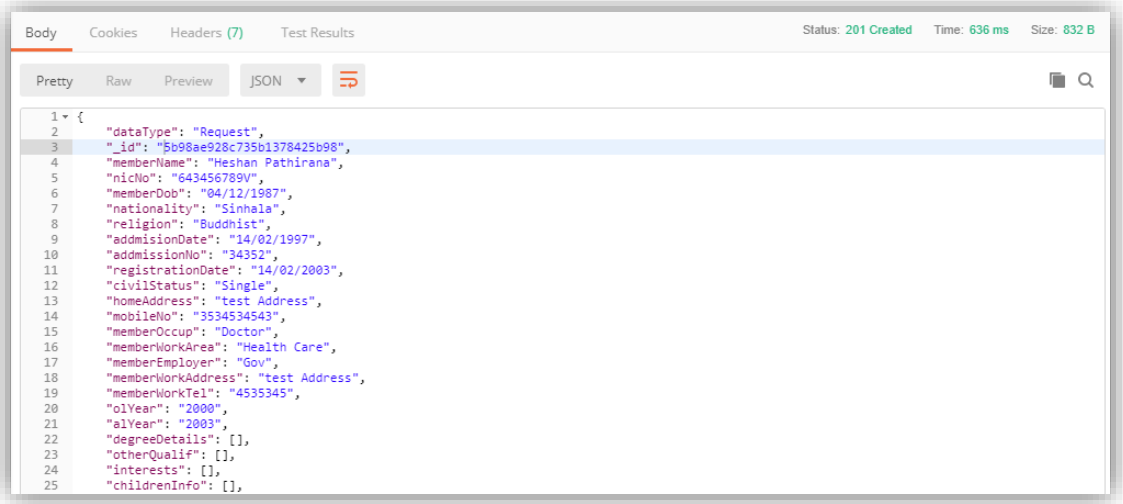
Figure 9: membership.controller.js

Testing

Test Case 1

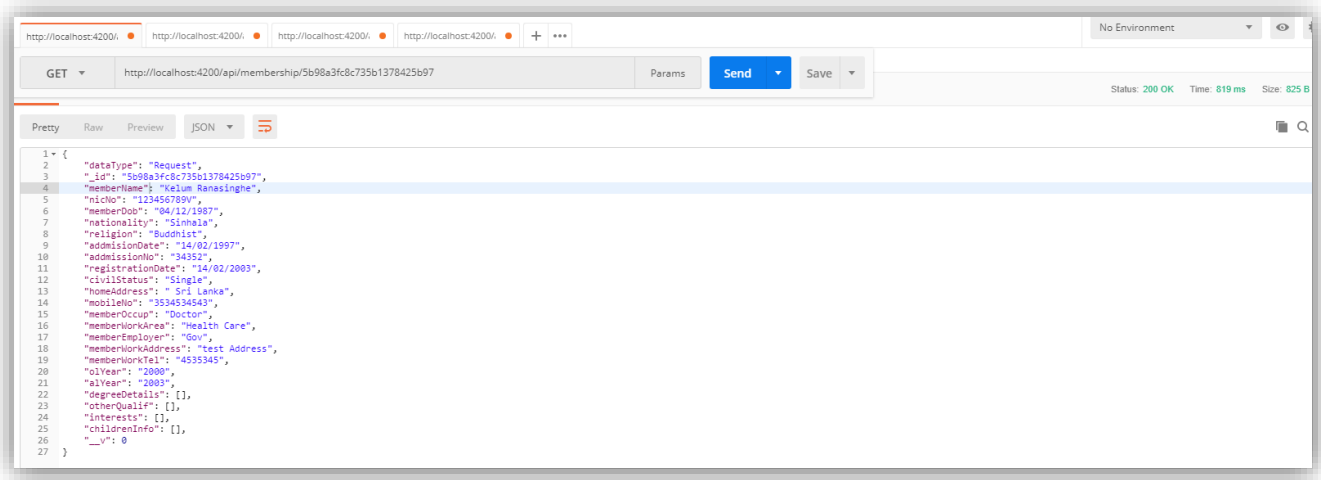
Test Case	Check the insert operation
Expected Behavior	When all the required field data are provided the record should be successfully saved in to the database
Test Steps	a) Set values for all the required fields b) Set the request type as "POST" c) Click the Send button d) Observe the saved data information in the output
Test Status	**Passed**





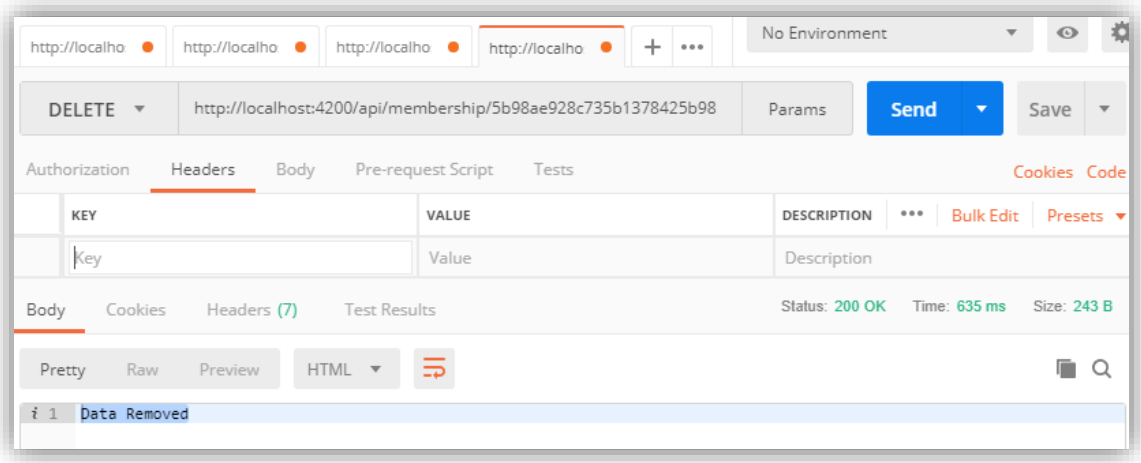
Test Case 2

Test Case	Check the Get operation
Expected Behavior	When a membership id is provided it should return the membership object.
Test Steps	a) Enter the object id of the membership item at the end of the URL b) Set the request type as "GET" c) Click the Send button d) Observe the membership data returned to the output
Test Status	**Passed**



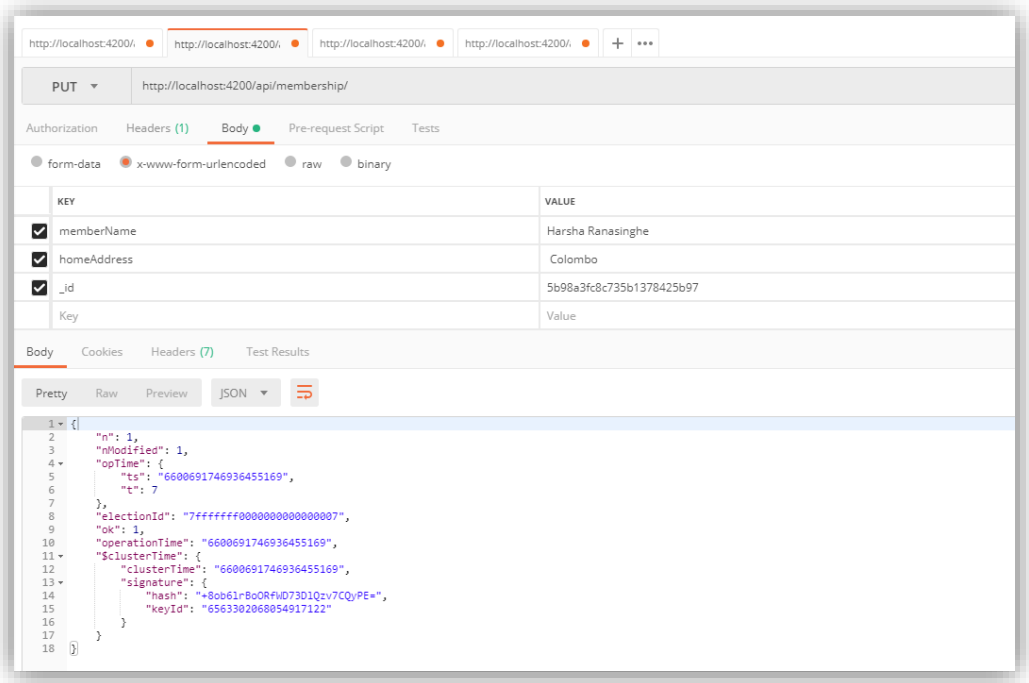
Test Case 3

Test Case	Check the Delete operation
Expected Behavior	When a membership id is provided it should delete the membership object.
Test Steps	e) Enter the object id of the membership item at the end of the URL f) Set the request type as "DELETE" g) Click the Send button h) Observe the response message "Data Removed"
Test Status	**Passed**



Test Case 4

Test Case	Check the Update operation
Expected Behavior	When a membership id is provided and send the update fields it should update the membership object.
Test Steps	a) Enter the object id of the membership item in a body field b) Enter the fields that need to be updated c) Set the request type as "PUT" d) Click the Send button e) Observe the response
Test Status	**Passed**



3.7 Task 6.3

Planning the user stories for the sprint 4

Estimate Time:	1 Hour
Actual Time:	1 Hour
Actual Time (this sprint):	1 Hour

Description

Please find the sprint 4 planning meeting minutes from the following link.

Bitbucket Link:

https://bitbucket.org/Computing_Projects_SLIIT/2018_sd07/src/master/Documents/Sprint%20Documents/Sprint%204/Sprint%204%20Planning%20Minutes.pdf

3.9 Task 7.3

Sprint 3 sprint retrospective

Estimate Time:	0.5 Hours
Actual Time:	0.5 Hours
Actual Time (this sprint):	0.5 Hours

Please refer the following Sprint Retrospective section for more details.

4 Development Methodology

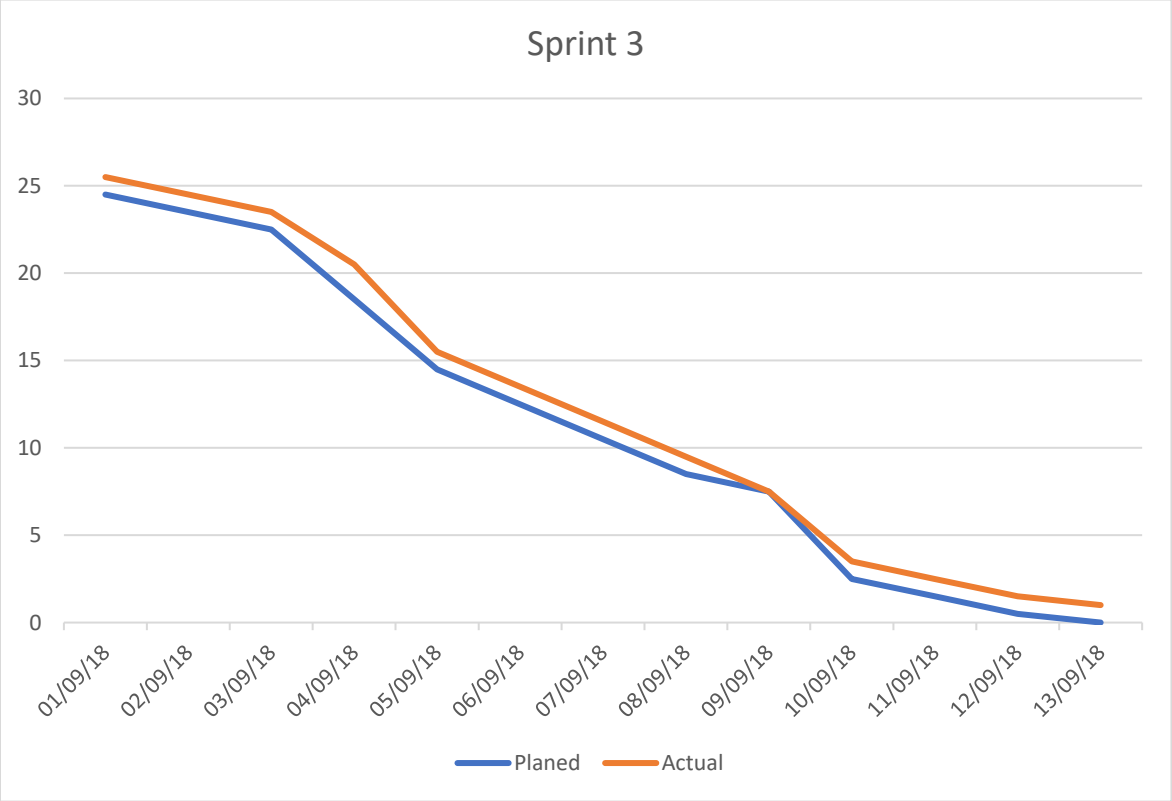
4.1 Minutes

We couldn't start the JIRA sprint during this sprint because of everyone was busy with their mid exams and couldn't find time prepare the sprint. So we hope to just close the user stories/ tasks which each of us have completed during this sprint at the beginning of the next sprint.

We had 5 standup meetings within this sprint and the minutes of those meetings can be found from the following link.

Standup Meeting Minutes:
https://bitbucket.org/Computing_Projects_SLIIT/2018_sd07/src/master/Documents/Standup%20Meeting%20Minutes/Sprint%203/Standup%20Meeting%20-%20Sprint%203.pdf

4.2 Burndown Chart



4.3 Sprint Retrospective (Task 7.3)

Estimate Time: 0.5 Hours

Actual Time: 0.5 Hours

Actual Time (this sprint): 0.5 Hours

Description

As expected everything went well in this sprint. I was able to successfully complete all the tasks that I assigned for myself at the beginning of the sprint. There was bit lengthy researching and learning part during the membership request page implementation since I'm new to the dynamic form building. However I managed to complete that task up and running at the end of the sprint. There were no any considerable issues within this sprint.

4.4 Time Management

Task ID	**Task**	**Task Status**	**Estimated Times (h)**	**Actual Times (h)**
Task 14.1	New Membership Page Designing	Completed	2	2
Task 14.2	New Membership GUI Implementation	Completed	12	13
Task 14.3	New Membership Page Validation Methods	Completed	2	3
Task 14.4	Membership Database Design	Completed	1	1
Task 14.5	New Membership Page CRUD operations	Completed	6	5
Task 7.3	Sprint 3 Retrospective	Completed	1	1
Task 6.3	Planning the user stories for the sprint 4	Completed	0.5	0.5
Total Time			**24.5**	**25.5**