2 - Solution

1. Create an Edit handler to update the profile. We only want to validate against the display name here as we do not ask the user for a bio when they register so we want to allow this to be optional.

```
using System.Threading;
using System.Threading.Tasks;
using Application.Core;
using Application.Interfaces;
using AutoMapper;
using FluentValidation;
using MediatR;
using Microsoft.EntityFrameworkCore;
using Persistence;

namespace Application.Profiles
{
    public class Edit
    {
        public class Command : IRequest<Result<Unit>>
        {
            public string DisplayName { get; set; }
            public string Bio { get; set; }
        }

        public class CommandValidator : AbstractValidator<Command>
        {
            public CommandValidator()
            {
                RuleFor(x => x.DisplayName).NotEmpty();
            }
        }

        public class Handler : IRequestHandler<Command, Result<Unit>>
        {
            private readonly DataContext _context;
```

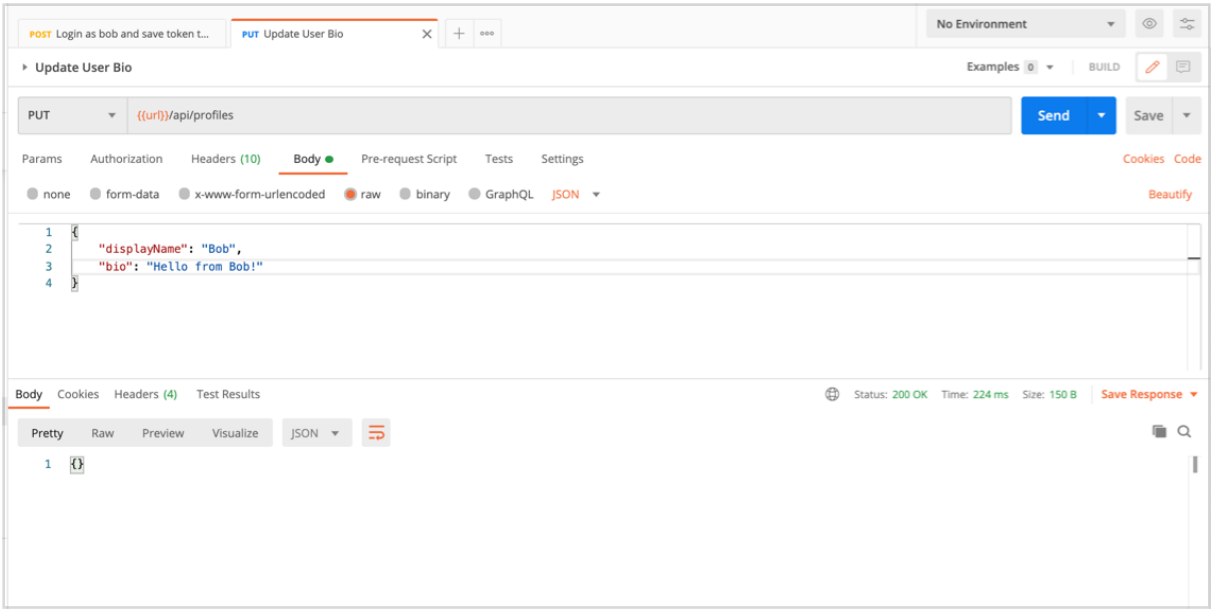
```
            private readonly IUserAccessor _userAccessor;
            public Handler(DataContext context, IUserAccessor userAccessor)
            {
                _userAccessor = userAccessor;
                _context = context;
            }

            public async Task<Result<Unit>> Handle(Command request,
CancellationToken cancellationToken)
            {
                var user = await _context.Users.FirstOrDefaultAsync(x =>
                    x.UserName == _userAccessor.GetUsername());

                user.Bio = request.Bio ?? user.Bio;
                user.DisplayName = request.DisplayName ?? user.DisplayName;

                var success = await _context.SaveChangesAsync() > 0;

                if (success) return Result<Unit>.Success(Unit.Value);

                return Result<Unit>.Failure("Problem updating profile");
            }
        }
    }
}
```

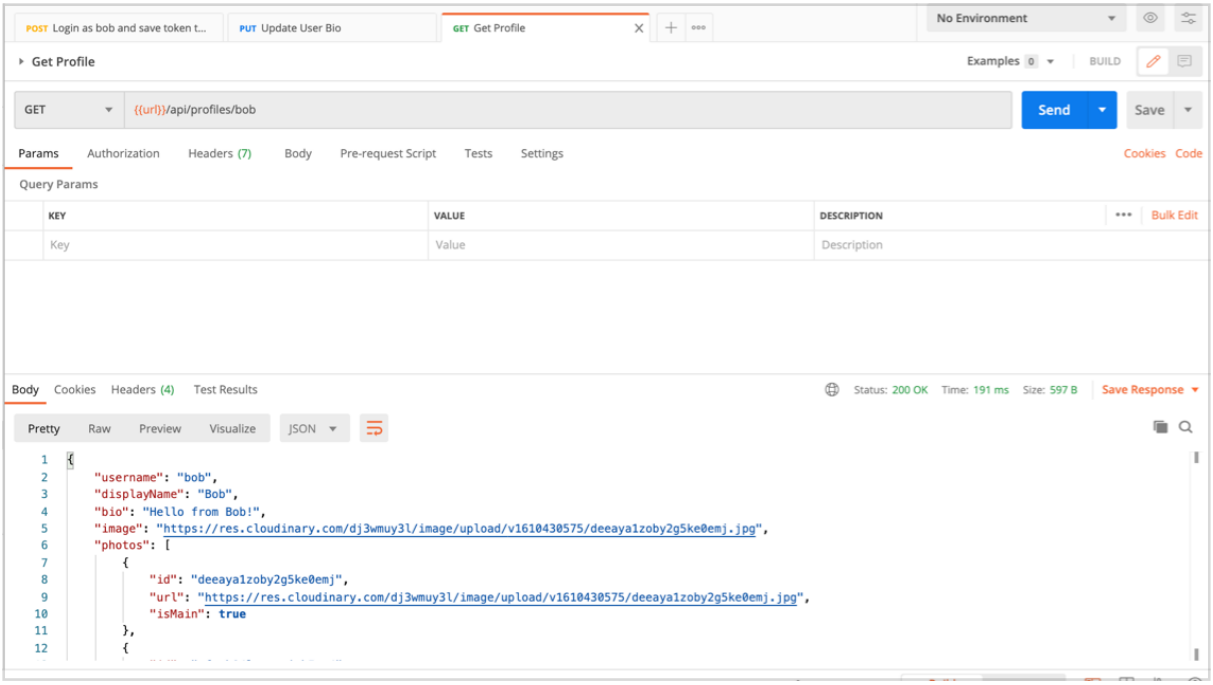
2. Update the ProfilesController and add an endpoint for editing the profile.

```
using System.Threading.Tasks;
using Application.Profiles;
using Microsoft.AspNetCore.Mvc;

namespace API.Controllers
{
    public class ProfilesController : BaseApiController
```

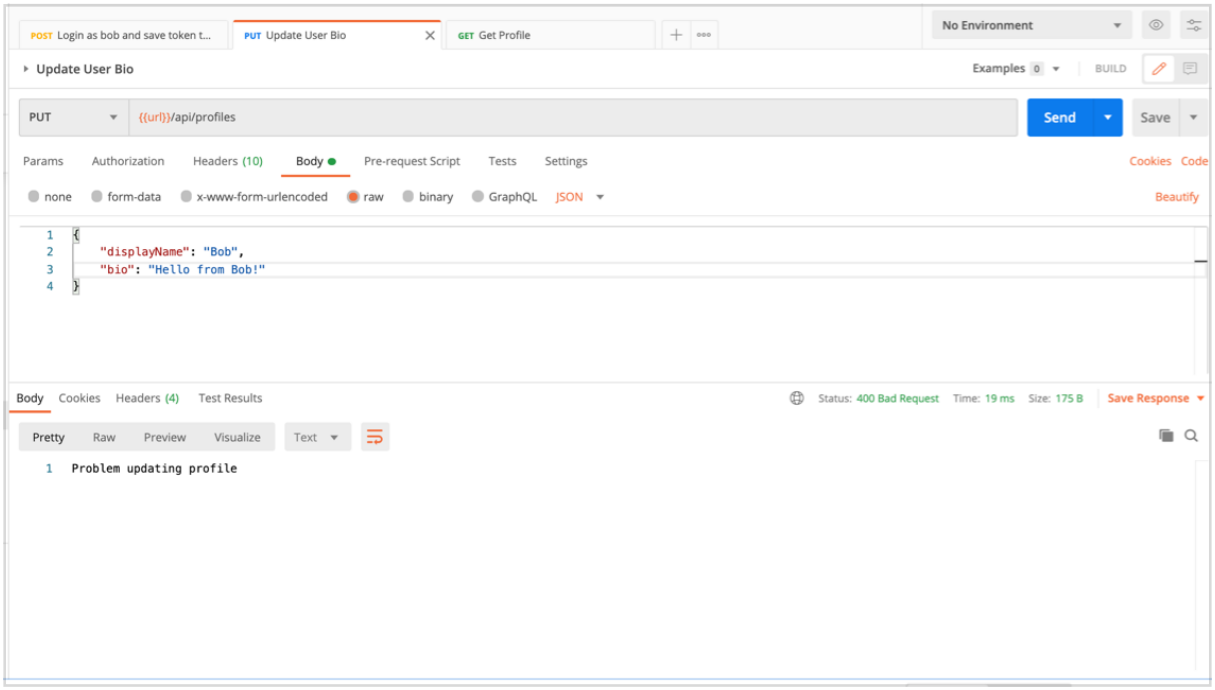
```
    {
        [HttpGet("{username}")]
        public async Task<IActionResult> GetProfile(string username)
        {
            return HandleResult(await Mediator.Send(new Details.Query{Username
= username}));
        }

        [HttpPut]
        public async Task<IActionResult> Edit(Edit.Command command)
        {
            return HandleResult(await Mediator.Send(command));
        }
    }
}
```

3. Test the endpoint in Postman using the 3 requests in the Module 18 folder:





If we try and update the profile again without changing it we will get a 400 bad request as there were no changes to save to the Database:



If you wanted a different behaviour here so that you always get a 200 back even if no changes were saved then you can mark the entity as modified regardless of whether there were any changes by updating the Edit handler with the following code:

```
            public async Task<Result<Unit>> Handle(Command request,
CancellationToken cancellationToken)
```

```
            {
                var user = await _context.Users.FirstOrDefaultAsync(x =>
                    x.UserName == _userAccessor.GetUsername());

                user.Bio = request.Bio ?? user.Bio;
                user.DisplayName = request.DisplayName ?? user.DisplayName;

                _context.Entry(user).State = EntityState.Modified;

                var success = await _context.SaveChangesAsync() > 0;

                if (success) return Result<Unit>.Success(Unit.Value);

                return Result<Unit>.Failure("Problem updating profile");
            }
```

Now even sending up the same request will always result in a successful response. This is not really relevant to our app as we will prevent form submission if the user has not actually updated their profile (note: may need to restart API to test this functionality).

4. Updating the client. Add a new request in the agent.ts. Note the use of Partial<Profile> as we are only allowing the user to update 2 of the properties contained in the Profile type.

```
const Profiles = {
    get: (username: string) => requests.get<Profile>(`/profiles/${username}`),
    uploadPhoto: (file: Blob) => {
        let formData = new FormData();
        formData.append('File', file);
        return axios.post<Photo>('photos', formData, {
            headers: {'Content-type': 'multipart/form-data'}
        })
    },
    setMainPhoto: (id: string) => requests.post(`/photos/${id}/setMain`, {}),
    deletePhoto: (id: string) => requests.del(`/photos/${id}`),
    updateProfile: (profile: Partial<Profile>) => requests.put(`/profiles`,
profile)
}
```

5. Add a helper method in the user store to set the display name:

```
setDisplayName = (name: string) => {
    if (this.user) this.user.displayName = name;
}
```

6. Add a method to update the profile in the profile store. The types are a bit tricky here as we are setting the Profile with the existing profile and overwriting any changes to the profile from the partial profile we are passing in as a parameter so we need to make use of the 'as Profile' to make TypeScript happy.

```
    updateProfile = async (profile: Partial<Profile>) => {
        this.loading = true;
        try {
            await agent.Profiles.updateProfile(profile);
            runInAction(() => {
                if (profile.displayName && profile.displayName !==
store.userStore.user?.displayName) {
                    store.userStore.setDisplayName(profile.displayName);
                }
                this.profile = {...this.profile, ...profile as Profile};
                this.loading = false;
            })
        } catch (error) {
            console.log(error);
            runInAction(() => this.loading = false);
        }
    }
```

7. Create a Profile Edit form component in the Profiles feature folder. We are passing down

the 'setEditMode' function that we will need to create using 'useState' in the parent component here so we can turn off the edit mode once the submission is complete.

```
import { Form, Formik } from "formik";
import { observer } from "mobx-react-lite";
import { Button } from "semantic-ui-react";
import MyTextArea from "../../app/common/form/MyTextArea";
import MyTextInput from "../../app/common/form/MyTextInput";
import { useStore } from "../../app/stores/store";
import * as Yup from 'yup';

interface Props {
    setEditMode: (editMode: boolean) => void;
}

export default observer(function ProfileEditForm({setEditMode}: Props) {
    const {profileStore: {profile, updateProfile}} = useStore();
    return (
        <Formik
            initialValues={{displayName: profile?.displayName, bio:
profile?.bio}}
            onSubmit={values => {
                updateProfile(values).then(() => {
                    setEditMode(false);
                })
            }}
            validationSchema={Yup.object({
                displayName: Yup.string().required()
            })}
        >
            {({isSubmitting, isValid, dirty}) => (
                <Form className='ui form'>
                    <MyTextInput placeholder='Display Name'
name='displayName' />
                    <MyTextArea rows={3} placeholder='Add your bio'
name='bio' />
                    <Button
                        positive
```

```
                        type='submit'
                        loading={isSubmitting}
                        content='Update profile'
                        floated='right'
                        disabled={!isValid || !dirty}
                    />
                </Form>
            )}
        </Formik>
    )
})
```

8. Create a ProfileAbout component that will make use of the form, as well as displaying the user bio. Note: the style 'whiteSpace: 'pre-wrap'' will preserve line breaks that are entered into the text area here.

```
import React, {useState} from 'react';
import {useStore} from "../../app/stores/store";
import {Button, Grid, Header, Tab} from "semantic-ui-react";
import ProfileEditForm from "./ProfileEditForm";
import { observer } from 'mobx-react-lite';

export default observer(function ProfileAbout() {
    const {profileStore} = useStore();
    const {isCurrentUser, profile} = profileStore;
    const [editMode, setEditMode] = useState(false);

    return (
        <Tab.Pane>
            <Grid>
                <Grid.Column width='16'>
                    <Header floated='left' icon='user' content={`About $
{profile?.displayName}`} />
                    {isCurrentUser && (
                        <Button
```

```
                            floated='right'
                            basic
                            content={editMode ? 'Cancel' : 'Edit Profile'}
                            onClick={() => setEditMode(!editMode)}
                        />
                    )}
                </Grid.Column>
                <Grid.Column width='16'>
                    {editMode ? <ProfileEditForm setEditMode={setEditMode} /> :
<span style={{whiteSpace: 'pre-wrap'}}>{profile?.bio}</span>}

                </Grid.Column>
            </Grid>
        </Tab.Pane>
    )
})
```

9. Update the ProfileContent component. Note that since we are getting the profile directly from the store we do not need to pass down the profile here.

```
import { observer } from 'mobx-react-lite';
import React from 'react';
import { Tab } from 'semantic-ui-react';
import { Profile } from '../../app/models/profile';
import ProfileAbout from './ProfileAbout';
import ProfilePhotos from './ProfilePhotos';

interface Props {
    profile: Profile;
}

export default observer(function ProfileContent({profile}: Props) {
    const panes = [
        {menuItem: 'About', render: () => <ProfileAbout />},
        {menuItem: 'Photos', render: () => <ProfilePhotos profile={profile} /
>},
```

```
        {menuItem: 'Events', render: () => <Tab.Pane>Events Content</
Tab.Pane>},
        {menuItem: 'Followers', render: () => <Tab.Pane>Followers Content</
Tab.Pane>},
        {menuItem: 'Following', render: () => <Tab.Pane>Following Content</
Tab.Pane>},
    ];

    return (
        <Tab
            menu={{fluid: true, vertical: true}}
            menuPosition='right'
            panes={panes}
        />
    )
})
```

10. Update the Profile Card as well.

```
import { observer } from 'mobx-react-lite';
import React from 'react';
import { Link } from 'react-router-dom';
import { Card, Icon, Image } from 'semantic-ui-react';
import { Profile } from '../../app/models/profile';

interface Props {
    profile: Profile;
}

export default observer(function ProfileCard({profile}: Props) {
    return (
        <Card as={Link} to={`/profiles/${profile.username}`}>
            <Image src={profile.image || '/assets/user.png'} />
            <Card.Content>
                <Card.Header>{profile.displayName}</Card.Header>
                <Card.Description>{profile.bio}</Card.Description>
```

```
            </Card.Content>
            <Card.Content extra>
                <Icon name='user' />
                20 followers
            </Card.Content>
        </Card>
    )
})
```

11. Test it works. If someone has a lot of text in their profile then it will make our attendee cards look quite bad, so add a truncate function to restrict the amount of text displayed in the attendee profile cards.

```
export default observer(function ProfileCard({profile}: Props) {
    function truncate(str: string | undefined) {
        if (str) {
            return str.length > 40 ? str.substring(0, 37) + '...' : str;
        }
    }

    return (
        <Card as={Link} to={`/profiles/${profile.username}`}>
            <Image src={profile.image || '/assets/user.png'} />
            <Card.Content>
                <Card.Header>{profile.displayName}</Card.Header>
                <Card.Description>{truncate(profile.bio)}</Card.Description>
            </Card.Content>
            <Card.Content extra>
                <Icon name='user' />
                20 followers
            </Card.Content>
        </Card>
    )
})
```

12. Feature complete! Now we can commit our changes into source control in preparation for the next section.

#reactivities/section 18#