Friendly

Job Board Spin-Off

Jacob Brokaw | Full Stack Web Developer | Dec 13th, 2019
fsdp.jacobbrokaw.com

Welcome to the Docs

Good afternoon, presentation attendee! Welcome to my project documentation, the place where I explain the magic. Before we get into that though, let me take a moment to introduce myself.

Who am I?

Simply put, I'm a creator. Building computer applications, whether it be native or web, has always been exciting to me. I enjoy learning, and I strive to become a better developer by exposing myself to many different technologies. When building an application, the most rewarding experience to me is tying the backend, middle-tier, and front end together to create a seamless user-experience.

The "Friendship" Board

For my final project, I chose to work on the Job Board. But rather than focusing on jobs, I took the concept and applied it to friendships in the workplace.

Reflection

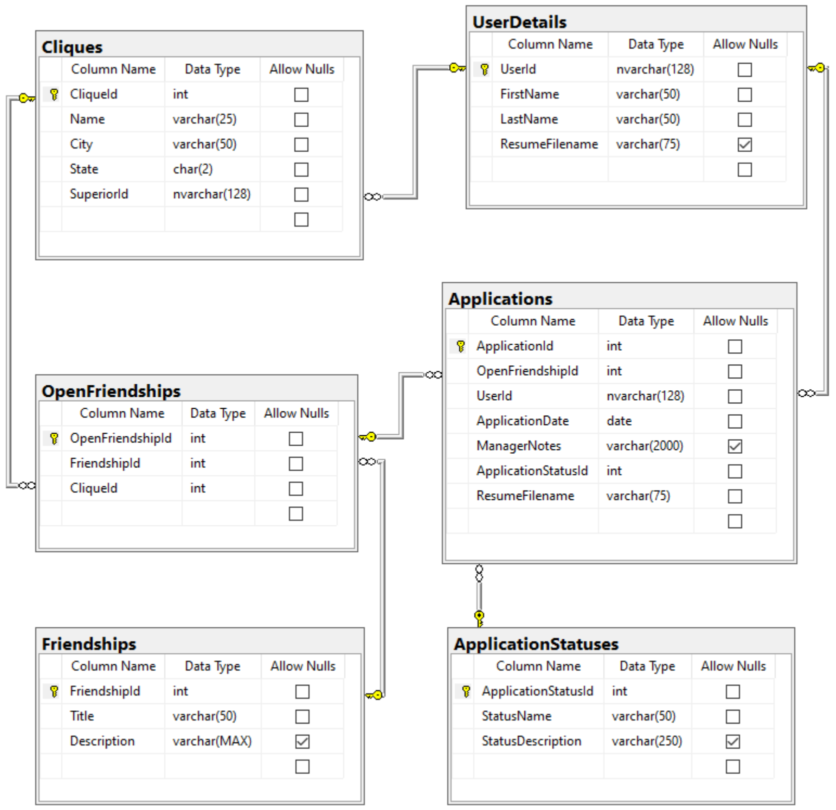
During the creation of Friendly, I focused on improving my project management skills and the way I build an application. This time, I finished every piece of functionality, before touching the UI. When looking back, I can confidently say that the quality and efficiency of my work benefitted tremendously. Of course, there's always room for improvement though. In my future projects, I will continue to refine my workflow.

Project Overview

Friendly's primary features:

- Easy application review process
- Ability to manage cliques (groups) and openings
- Apply to an opening with a single click
- Material Design compliant UI
- Seamless workflow between three types of users, Admin, Superior, and Friend

Database Diagram



Cliques

Column Name	Data Type	Allow Nulls
CliqueId	int	☐
Name	varchar(25)	☐
City	varchar(50)	☐
State	char(2)	☐
SuperiorId	nvarchar(128)	☐
		☐

UserDetails

Column Name	Data Type	Allow Nulls
UserId	nvarchar(128)	☐
FirstName	varchar(50)	☐
LastName	varchar(50)	☐
ResumeFilename	varchar(75)	☑
		☐

Applications

Column Name	Data Type	Allow Nulls
ApplicationId	int	☐
OpenFriendshipId	int	☐
UserId	nvarchar(128)	☐
ApplicationDate	date	☐
ManagerNotes	varchar(2000)	☑
ApplicationStatusId	int	☐
ResumeFilename	varchar(75)	☐
		☐

OpenFriendships

Column Name	Data Type	Allow Nulls
OpenFriendshipId	int	☐
FriendshipId	int	☐
CliqueId	int	☐
		☐

Friendships

Column Name	Data Type	Allow Nulls
FriendshipId	int	☐
Title	varchar(50)	☐
Description	varchar(MAX)	☑
		☐

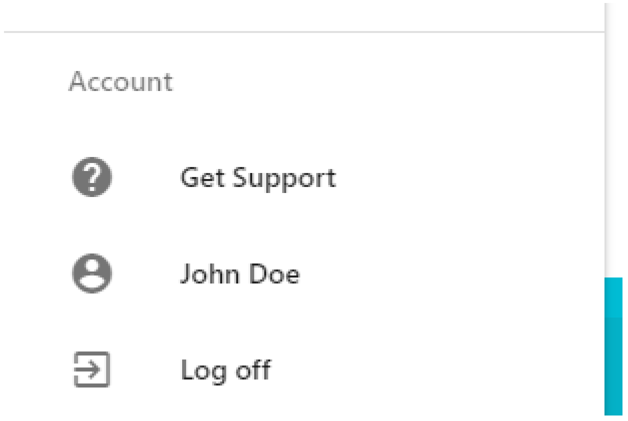
ApplicationStatuses

Column Name	Data Type	Allow Nulls
ApplicationStatusId	int	☐
StatusName	varchar(50)	☐
StatusDescription	varchar(250)	☑
		☐

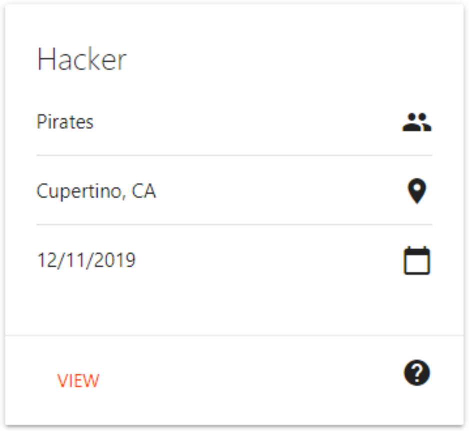
Permissions Overview

Home	Index	Support		
Admin	X	X		
Cliques	Index	Create	Edit	Delete
Admin	X	X	X	X
Friendships	Index	Create	Edit	Delete
Admin	X	X	X	X
Superior	X			
Openings	Index	Create	Edit	Delete
Admin	X			
Superior	X	X	X	X
Friend	X			
Applications	Index	Details	Apply	Review
Admin	X	X		
Superior	X	X		X
Friend	X	X	X	

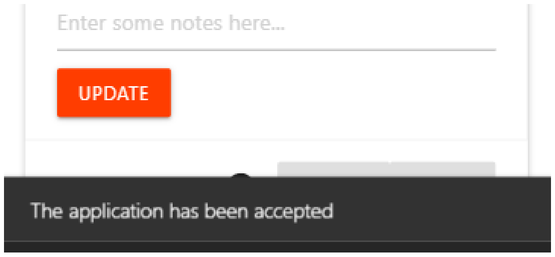
User Interface Highlights



A dynamic account menu sits at the bottom of Friendly's side nav



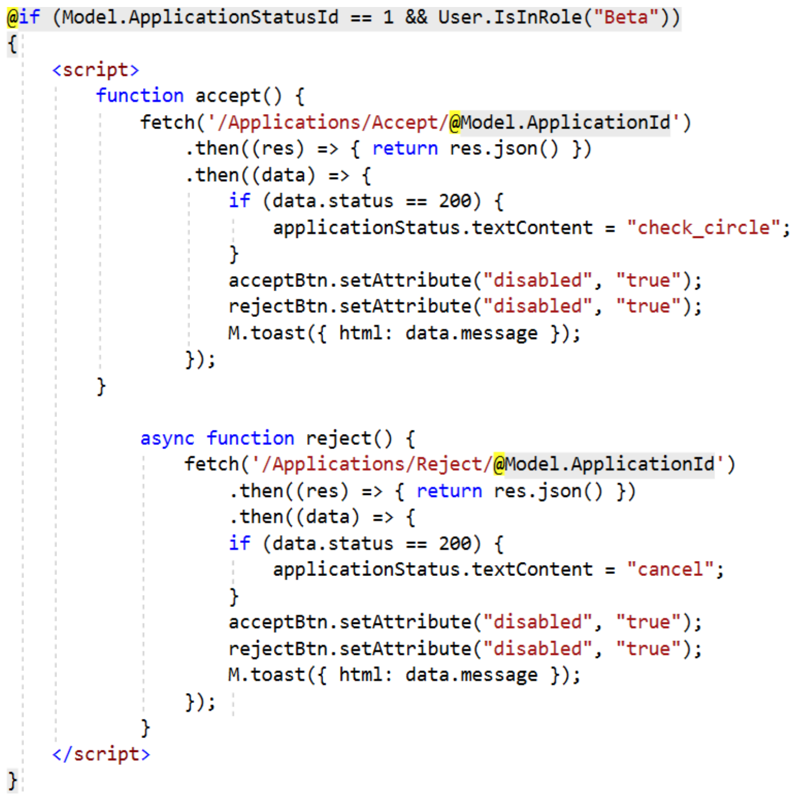
Applications are tiled out into custom, responsive cards



The application has been accepted

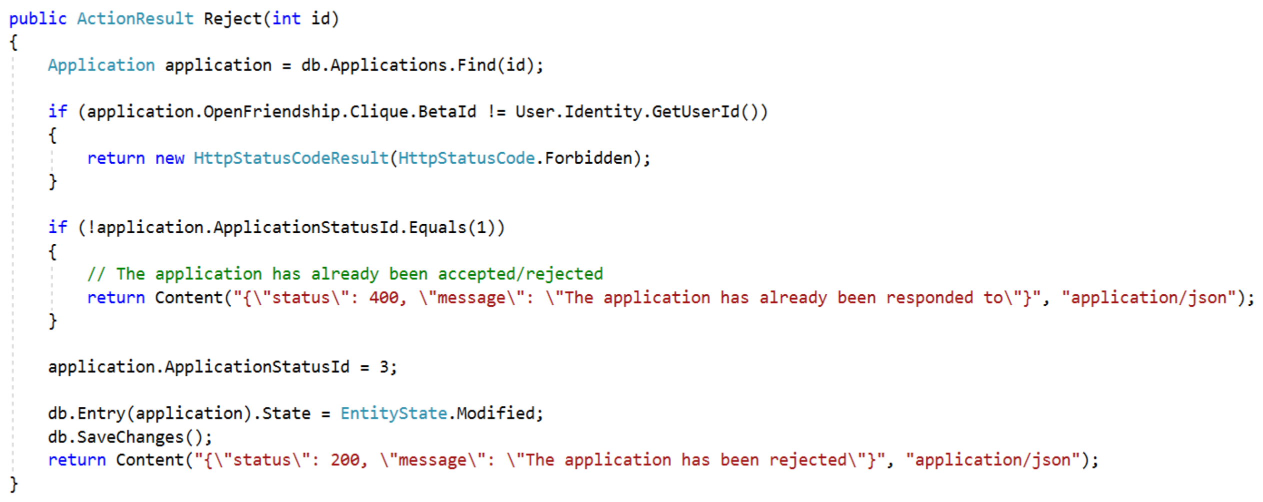
Using Materialize, I implemented toasts (unobtrusive notifications) to display dynamic information

Code Highlights



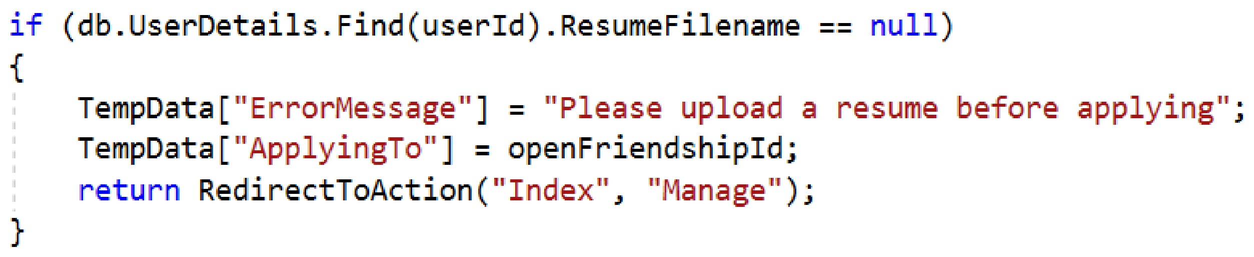
```
@if (Model.ApplicationStatusId == 1 && User.IsInRole("Beta"))
{
    <script>
        function accept() {
            fetch('/Applications/Accept/@Model.ApplicationId')
                .then((res) => { return res.json() })
                .then((data) => {
                    if (data.status == 200) {
                        applicationStatus.textContent = "check_circle";
                    }
                    acceptBtn.setAttribute("disabled", "true");
                    rejectBtn.setAttribute("disabled", "true");
                    M.toast({ html: data.message });
                });
        }

            async function reject() {
                fetch('/Applications/Reject/@Model.ApplicationId')
                    .then((res) => { return res.json() })
                    .then((data) => {
                    if (data.status == 200) {
                        applicationStatus.textContent = "cancel";
                    }
                    acceptBtn.setAttribute("disabled", "true");
                    rejectBtn.setAttribute("disabled", "true");
                    M.toast({ html: data.message });
                });
            }
    </script>
}
```

Rather than changing the status of an application through a form post, users can click a button and asynchronously call to the server.



```
public ActionResult Reject(int id)
{
    Application application = db.Applications.Find(id);

    if (application.OpenFriendship.Clique.BetaId != User.Identity.GetUserId())
    {
        return new HttpStatusCodeResult(HttpStatusCode.Forbidden);
    }

    if (!application.ApplicationStatusId.Equals(1))
    {
        // The application has already been accepted/rejected
        return Content("{\"status\": 400, \"message\": \"The application has already been responded to\"}", "application/json");
    }

    application.ApplicationStatusId = 3;

    db.Entry(application).State = EntityState.Modified;
    db.SaveChanges();
    return Content("{\"status\": 200, \"message\": \"The application has been rejected\"}", "application/json");
}
```

To update the status of an application, the server listens for a get request with an application ID. Instead of sending a View, the server sends a JSON response, allowing for an uninterrupted user experience.



```
if (db.UserDetails.Find(userId).ResumeFilename == null)
{
    TempData["ErrorMessage"] = "Please upload a resume before applying";
    TempData["ApplyingTo"] = openFriendshipId;
    return RedirectToAction("Index", "Manage");
}
```

When applying to an open friendship, a user must have a resume. To ensure this, a simple if statement sends the user to their account management page, passing the ID of the open friendship they're applying to. Once they save their resume, it will continue the application.