**Initial Setup/Design Decisions – Needs to be done ASAP**

1. **Create initial repos/files**
   1. **GitHub Repo – Yenny**
      1. Sam will save this file in the repo
   2. **ASP.NET webAPI setup – Mike, Omar, & Samaja + Steve & Ron for salt/pepper stuff**
      1. **Create the models**
      2. **CORS Policies needed**
         1. SQL case sensitive
         2. What about the “bad CORS policy?” Copy & paste this in. 🡨allows React app to hit the API
      3. **EF Core with migrations (see below for tables needed)**
      4. **Folder structure – put a dummy file in each folder**
         1. 0. Models & DTOs
         2. 1. Controller Layer
         3. 2. Service Layer
            1. Interfaces
         4. 3. Data Access Layer
            1. Interfaces
            2. Data Context
   3. Create the initial React App
      1. Folder structure (just a starting point)
         1. Components
            1. NavBar (across the top)
            2. Log In Screen
            3. Logged In Screen
            4. Leave Management
            5. Performance Management
            6. Employee Management (Managers)
            7. My Profile
         2. CSS file we can all import (use Bootstrap template?)
2. **Version Control**
   1. Each team
      1. Has its own branch for the feature they’re assigned
   2. Pushing to main – how do we want to handle this?
3. **Testing**
   1. Each team handles validation (in front end and backend) of their feature, including coming up with what to validate
4. **Models/Tables Needed**
   1. Use Entity Relationship Diagram
      1. New Table:
         1. Encryption
            1. Salt
      2. All IDs should be Guids
   2. Sam’s Notes:
      1. Employee
         1. First Name
         2. Last Name
         3. Employee ID
         4. Address
            1. Street
            2. 2nd Line
            3. City
            4. State
            5. Zip
         5. Email address
         6. Manager ID
         7. Birthday
      2. Manager – **child of employee? -OR- employee type indicator field?**
         1. Everything in employee
         2. List of employee IDs
      3. Leave Requests
         1. Employee ID
         2. Start Date
         3. End Date
         4. Leave Type
         5. Manager ID
         6. Manager Approval Status
      4. Leave Balances
         1. Employee Id
         2. Leave Type
         3. Leave Balance
      5. Goals
         1. Employee Id
         2. Goal Id
         3. Goal description
         4. Goal weight
         5. Deadline
         6. Self Rating by Goal
         7. Manager Rating by Goal
      6. Company Holidays
5. **Other decisions/considerations**
   1. CSS styling/overall look of the front-end
      1. Way to handle this: 1 person from each team be the styling expert so we have uniformity across the whole web app front end
   2. DevOps
   3. Testing – any kind of cross-app testing plan & execution
   4. Scrum process
      1. When to demo to each other? Good to have deadlines for this so we’re not cramming at the end – maybe aim to demo on 6/28, 7/2, and 7/5?
      2. Do we want to use boards? Think it’d be helpful, but some folks might think it’s too much admin

**Features/Teams**

1. **Leave Management** 
   1. Employee:
      1. CREATE/POST: Apply for leave/leaves
         1. View manager’s name while applying for leave
      2. READ: Get details about applied leaves
      3. \*\*Get holiday data
      4. \*\* Get email notification when leave is approved or rejected
   2. Manager:
      1. For themselves/as an employee: All tasks in “Employee” section
      2. For their team/as a manager:
         1. UPDATE: Approve/reject leave
         2. Get email notification when an employee applies for leave
         3. Interact with a dummy API to update a payroll service for changes in paid or unpaid leave
            1. This may include endpoints to update payroll records based on approved leaves
            2. The dummy service will periodically send error status codes, requiring proper handling of HTTP status codes
2. **Employee Management**
   1. Employee:
      1. Log in
         1. Passwords should be encrypted using appropriate hashing algorithm
      2. Change password
      3. Request a forgotten password (which will be sent to their registered email)
      4. Edit basic profile information like phone number and address
      5. View birthday and work anniversaries “of the employees”? – maybe just view their own? Or their coworkers? Or none?
   2. Manager:
      1. For themselves/as an employee: All tasks in the “Employee” section
      2. For their team/as a manager:
         1. View birthdays and work anniversaries of everyone that reports to them
         2. See the list of employees who report to me
3. **Performance Reviews**
   1. Employee:
      1. Enter the performance review document with deliverables, achievements, and areas of improvement in a textual form
      2. Get notified when the manager provides feedback for the performance review
   2. Manager:
      1. For themselves/as an employee: All tasks in “Employee” section
      2. For their team/as a manager:
         1. Review and provide feedback and score for the self-performance review provided by the employees who report to this manager
         2. Get an email notification when an employee submits the performance review
         3. Include reports representing the results of the performance reviews for their team
4. **Goal Setting – merge with 3**
   1. Employee:
      1. Create the goals and targets for the year
      2. Each goal and target will have a deadline and a weight
   2. Manager:
      1. For themselves/as an employee: All tasks in the “Employee” section
      2. For their team/as a manager: N/A

**\*\* I think Goal Setting should be combined with Performance & given to a single team, but I don’t know what the 4th team would do instead. Options:**

1. Could have 3 teams instead of 4
2. Maybe something with the dummy API idk how complex that is?
3. Are we gonna have to do something for an admin role? In the doc, they only mention admins here:

A close-up of a text

Description automatically generated

1. “Firefighting”
   1. Late requirements?
   2. Helping out other teams
2. Design overall CSS/look of the frontend
3. Other?

**Feature Teams**

* **Leave Management**

1. Marcus
2. Helen
3. Phil
4. Aaron
5. David
6. Veda

* **Employee Management**

1. Corey
2. Samantha
3. Ron
4. Samaja
5. Steve

* **Performance Reviews**

1. Ross
2. Mike
3. Omar
4. Bradley
5. Yenny
6. Josh
7. Fernando

**Initial Setup**

**Have 1 person do these steps:**

1. Clone the GitHub repo
   1. git clone https://github.com/240415-NET/project3.git
2. Create the gitignore file using this command
   1. dotnet gitignore
3. In VS Code
   1. Run this command to create the API
      1. dotnet net webapi --use-controllers -o NameOfYourAPI
   2. Run two commands for adding packages
      1. dotnet add package Microsoft.EntityFrameworkCore.SqlServer
      2. dotnet add package Microsoft.EntityFrameworkCore.Design
   3. Create the Data folder & the DataContext.cs file inside of it
      1. Add in this using statement:
         1. using Miscrosoft.EntityFrameworkCore;
      2. Create the DataContext class, which will inherit from the DbContext class (which comes preloaded with Microsoft.EntityFrameworkCore)
         1. Create a constructor for the class
         2. Add in your DbSet properties (which will be the table names that get generated by EF Core)
   4. Program.cs
      1. Read the connection string
         1. string connectionString = File.ReadAllText(@"../../ConnectionString.txt");
      2. Add in the Builder.Services.AddDbContext<NameOfDbContextClassYouCreated>
         1. builder.Services.AddDbContext<DataContext>(options =>options.UseSqlServer(connectionString));
   5. Create the models – ours are User and Activity.
      1. Label the primary key with the [Key] annotation
         1. Add in using System.ComponentModel.DataAnnotations;
      2. Add in anything needed to reflect the relationship between users & activity

**Everyone else needs to do this part:**

1. Clone the GitHub repo
   1. git clone https://github.com/240415-NET/project3.git
2. Create the connection string file
3. Run two commands for adding packages
   1. dotnet add package Microsoft.EntityFrameworkCore.SqlServer
   2. dotnet add package Microsoft.EntityFrameworkCore.Design

**Setting Up Layers & Creating Initial Methods in Each Layer**

1. **Create the Controller Layer class**
   1. You can copy the WeatherForecaseController.cs file so you can get the using statement, the [ApiController] & [Route], & then change the name of the class so it still inherits from ControllerBase. Just delete out the body of the code block for the class.
   2. Add a property for the interface that will be in the next layer down (the Service Layer)
   3. Add a constructor that takes in an object of type interface from the Service Layer
      1. Register the dependency in Program.cs so that an item of type IUserService (or whatever you call the Service Layer interface) will be passed to this constructor when it’s called
   4. Create a method inside of this class
      1. Before the method signature, include an http verb like [HttpPost]
      2. This method will call a method in the Service Layer
2. **Create Service Layer file and class**
   1. Have the class implement an interface (in the Service Layer)
   2. Add a property for the interface that will be in next layer down (the Data Layer)
   3. Add a constructor that takes in an object of type interface from the Data Layer
      1. Register the dependency in Program.cs so that an item of type IUserStorage (or whatever you call the Data Layer interface) will be passed to this constructor when it’s called
   4. Create the method inside of this Service Layer class
      1. This method will call a method in the Data Layer
3. **Create the Service Layer interface**
   1. Copy and paste the method signature from the Service Layer method into this interface
4. **Create the Data Layer file and class**
   1. Have the class implement an interface (in the Data Layer)
   2. Include a property for the DbContext (in ActivityTracker, we’ve named ours DataContext)
   3. Add a constructor that takes in an item of type DataContext
   4. Create the method inside of this class
      1. This method will call a method off of the DataContext object
5. **Create the Data Layer interface**
   1. Copy and paste the method signature from the Data Layer method into this interface
6. **Run these commands**
   1. dotnet ef migrations add YourMigrationName
   2. dotnet ef database update
   3. dotnet run

**Creating Any Future Methods in These Layers**

1. **Add the method to the Controller Layer**
   1. This will call a Service Layer method that we’ll create in the next step
2. **Add the method to the Service Layer**
   1. This will call a Data Layer method that we’ll create in the next step
3. **Add the method to the Data Layer**