Project Part 6: Final Report & Code Completion

Name: Sage Thomas

Project description: Simple Timesheet is a Ruby on Rails web app that manages an online timesheet system. Users can create or join companies. Then they can clock in and out while starting or ending work and their time will be tracked and summed for the current pay period. Owners and managers can add employees to a company and promote them to manager. They can also view employees' clocked time for a pay period.

Features that were implemented: (Note: "Admin" refers to "Owner" in the web app)

1	Can create a new user account with name, email, and password
2	Can create a new company
3	Can request other users to join company as employee or manager
4	User can have and join multiple companies
5	User can clock in and clock out
6	User can view their timesheet and hours
7	Manager/Admin can view all employees' timesheets and hours
10	Admin can delete company
11	Admin can set start and stop schedule of pay periods
12	Users can delete their accounts

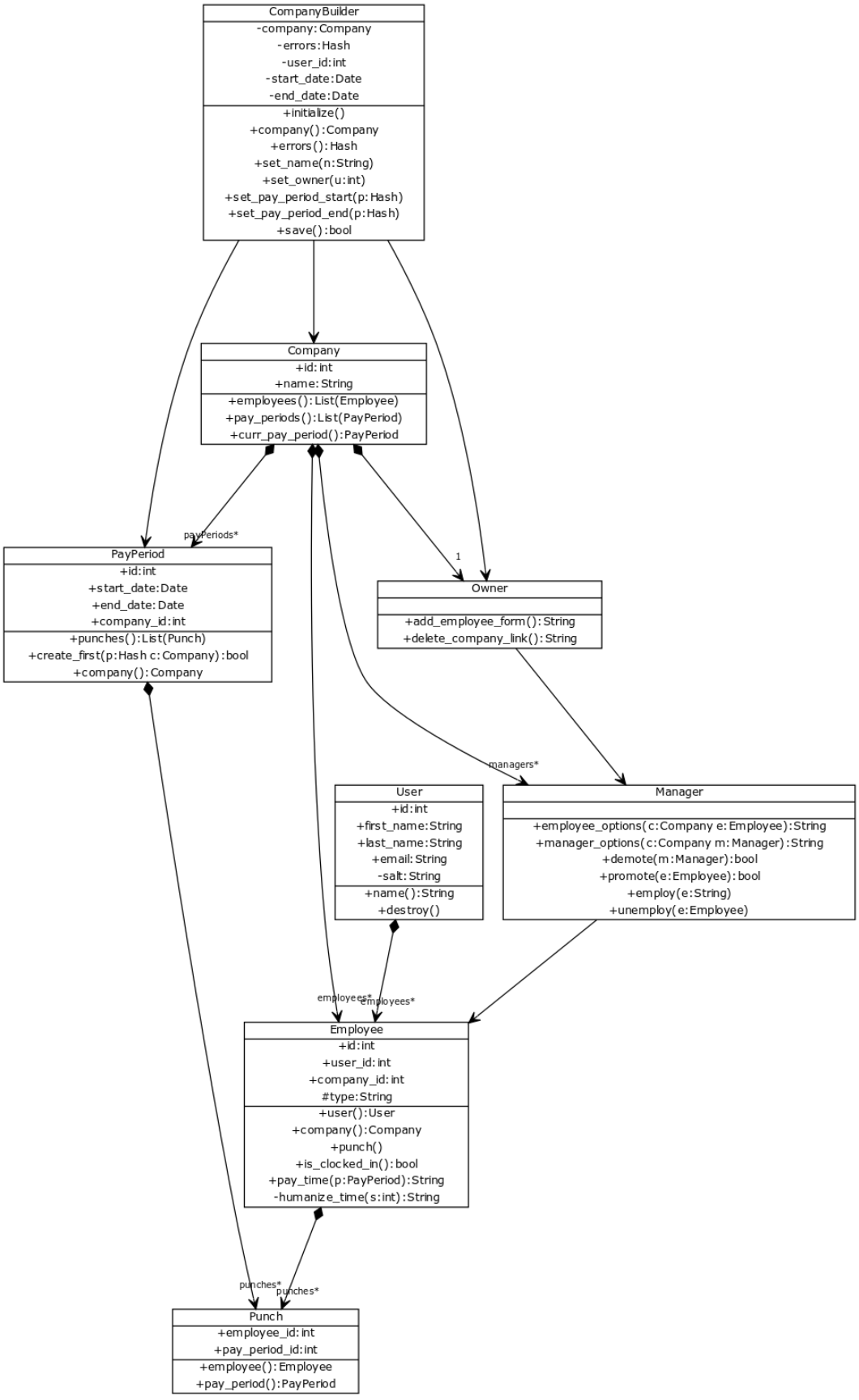
I was able to implement all the major features that I set out to implement.

Features that were not implemented:

8	Manager/Admin can edit timesheet
9	Manager/Admin can download timesheet(s) in csv format

These features, while useful, were not necessary and did not require me to practice any new object-oriented methods.

Final Class Diagram:


CompanyBuilder
-company:Company
-errors:Hash
-user_id:int
-start_date:Date
-end_date:Date
+initialize()
+company():Company
+errors():Hash
+set_name(n:String)
+set_owner(u:int)
+set_pay_period_start(p:Hash)
+set_pay_period_end(p:Hash)
+save():bool
Company
+id:int
+name:String
+employees():List(Employee)
+pay_periods():List(PayPeriod)
+curr_pay_period():PayPeriod
payPeriods*
1
PayPeriod
+id:int
+start_date:Date
+end_date:Date
+company_id:int
+punches():List(Punch)
+create_first(p:Hash c:Company):bool
+company():Company
Owner
+add_employee_form():String
+delete_company_link():String
managers*
User
+id:int
+first_name:String
+last_name:String
+email:String
-salt:String
+name():String
+destroy()
Manager
+employee_options(c:Company e:Employee):String
+manager_options(c:Company m:Manager):String
+demote(m:Manager):bool
+promote(e:Employee):bool
+employ(e:String)
+unemploy(e:Employee)
employees*
employees*
Employee
+id:int
+user_id:int
+company_id:int
#type:String
+user():User
+company():Company
+punch()
+is_clocked_in():bool
+pay_time(p:PayPeriod):String
-humanize_time(s:int):String
punches*
punches*
Punch
+employee_id:int
+pay_period_id:int
+employee():Employee
+pay_period():PayPeriod

Many things changed in my final class diagram for a few reasons. Some relations were poorly judged when I first made it. Other changes were mostly made to accommodate Ruby on Rails. Ruby on Rails is a very strict (and somewhat convoluted) framework so I had to make work arounds. There are many standard model methods that I did not include in this diagram because it would take up too much space and I did not write them. Additionally, Controller classes that define API functionality is not displayed here because those are also standard and any custom functionality that they implement should go through a method defined in one of the displayed classes. I also used single table inheritance to store Employees, Managers, and Owners so they extend each other with ease. I wasn't totally sure how to represent this but hopefully it is clear. I also added a CompanyBuilder class to make sure a company is created with an Owner and an initial PayPeriod. The CompaniesController class is not shown but is the director of CompanyBuilder. The original UML diagram was still helpful when setting up the database tables even if not perfect.

Builder Design Pattern:

app/controllers/companies_controller.rb (CompaniesController's create method is the director that parses the common input and takes the built company)

```
  # POST /companies
  # POST /companies.json
  def create
    require 'company_builder'

    company_builder = CompanyBuilder.new
    company_builder.set_name(company_params[:name])
    company_builder.set_owner(session[:user_id])
    company_builder.set_pay_period_start(params)
    company_builder.set_pay_period_end(params)

    respond_to do |format|

      if company_builder.save
        @company = company_builder.company()
        format.html { redirect_to @company, notice: 'Company was
successfully created.' }
        format.json { render :show, status: :created, location:
@company }
```

```
      else
        format.html { render :new }
        format.json { render json: company_builder.errors, status:
:unprocessable_entity }
      end
    end
  end
```

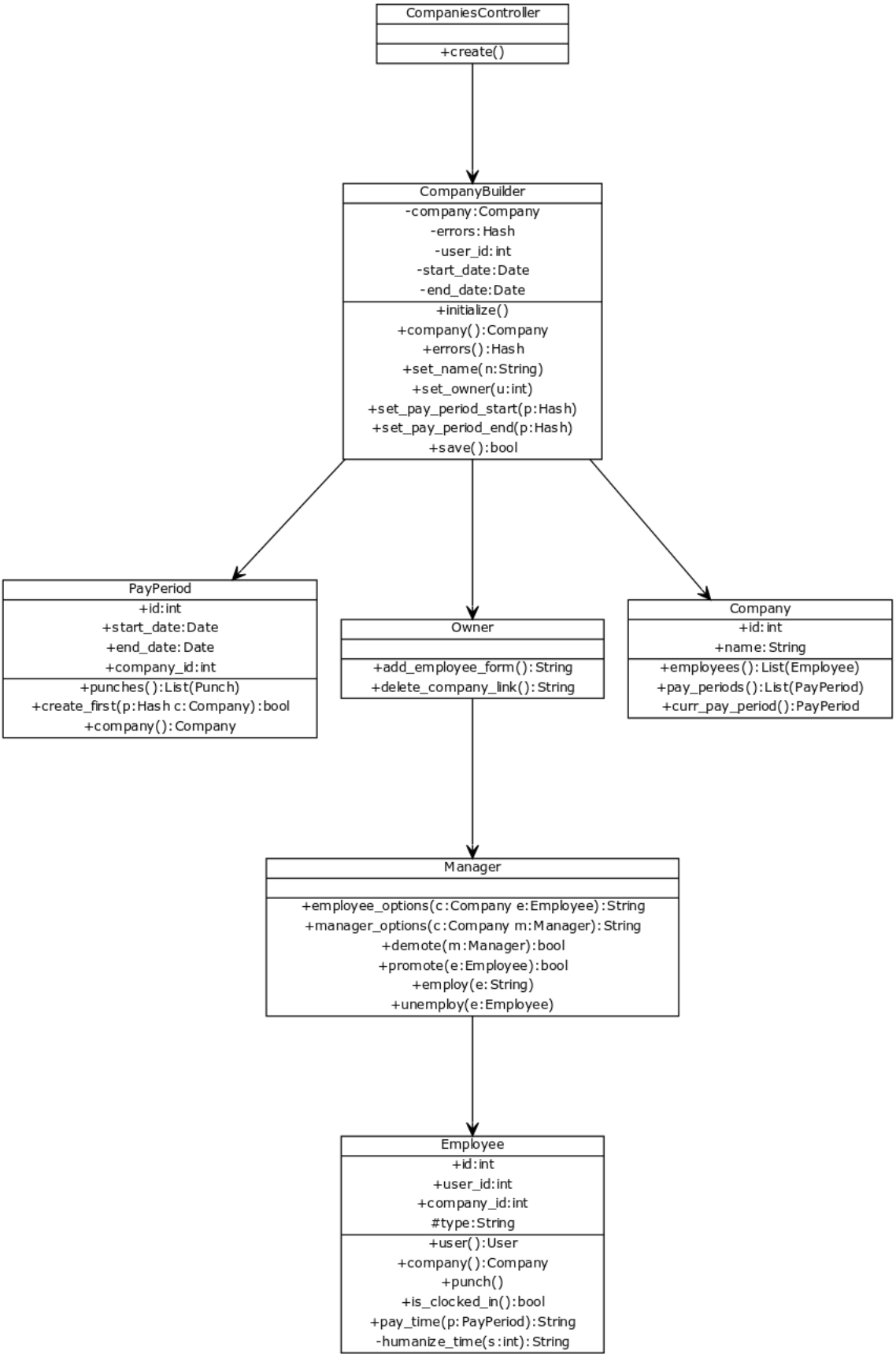
lib/company_builder.rb (CompanyBuilder is the builder that will take configuration and setup the new company and its dependencies. Ruby on Rails makes it hard to build dependency relationships in, so they are done here, after the company has been saved to the database and assigned an id)

```
class CompanyBuilder
    attr_reader :errors

    def initialize
        @company = Company.new
        @errors = {}
    end

    def company
        return @company
    end

    def set_name(name)
        @company.name = name
    end

    def set_owner(user_id)
        @user_id = user_id
    end

    def set_pay_period_start(params)
        @start_date =
Date.new(params[:company]["start_date(1i)"].to_i,
```

```
params[:company]["start_date(2i)"].to_i,
params[:company]["start_date(3i)"].to_i)
    end

    def set_pay_period_end(params)
        @end_date = Date.new(params[:company]["end_date(1i)"].to_i,
params[:company]["end_date(2i)"].to_i,
params[:company]["end_date(3i)"].to_i)
    end

    def save
        begin
            @company.save!
        rescue => e
            @errors[:company] = e.message
            return false
        end

        begin
            @pay_period = PayPeriod.new(start_date: @start_date,
end_date: @end_date, company_id: @company.id)
            @pay_period.save!
        rescue => e
            @company.destroy
            @errors[:pay_period] = e.message
            return false
        end

        begin
            @owner = Owner.new(company_id: @company.id, user_id:
@user_id)
            @owner.save!
        rescue => e
            @company.destroy # Will also destroy the pay period
            @errors[:owner] = e.message
            return false
```

```
        end

        return true
    end
end
```

Class Diagram:


CompaniesController
+create()
CompanyBuilder
-company:Company
-errors:Hash
-user_id:int
-start_date:Date
-end_date:Date
+initialize()
+company():Company
+errors():Hash
+set_name(n:String)
+set_owner(u:int)
+set_pay_period_start(p:Hash)
+set_pay_period_end(p:Hash)
+save():bool
PayPeriod
+id:int
+start_date:Date
+end_date:Date
+company_id:int
+punches():List(Punch)
+create_first(p:Hash c:Company):bool
+company():Company
Owner
+add_employee_form():String
+delete_company_link():String
Company
+id:int
+name:String
+employees():List(Employee)
+pay_periods():List(PayPeriod)
+curr_pay_period():PayPeriod
Manager
+employee_options(c:Company e:Employee):String
+manager_options(c:Company m:Manager):String
+demote(m:Manager):bool
+promote(e:Employee):bool
+employ(e:String)
+unemploy(e:Employee)
Employee
+id:int
+user_id:int
+company_id:int
#type:String
+user():User
+company():Company
+punch()
+is_clocked_in():bool
+pay_time(p:PayPeriod):String
-humanize_time(s:int):String

Implementation:

I implemented the Builder design pattern by creating the CompanyBuilder class in the lib directory, so I could import it into my CompaniesController class. I didn't make an interface because that is usually bad practice in Ruby unless absolutely needed, which it wasn't. The typical builder design didn't work perfectly for my needs to I adapted it. The director specifies parameters step by and the builder verifies that the parameters are valid when it can (like with date data from the user submitted json). Normally the builder class would simply return the built Company, but Rail's api design wants to make sure that everything saves correctly before continuing so I made a save() method. This method saves the initialized Company, Owner, and PayPeriod. If an error occurs, it deletes anything that the method previously saved, records the error to a hash, and returns false. That way Rails is notified that the Company creation failed, and it can show the user the error. If everything works, the new Company is retrieved, and the user is sent to its new page. I chose the builder design pattern because creating a new company is a complex process that requires verifying several database commits and many possible origins of error messages. Without it, I would have to create many if statements and messy code to make sure the logic was proper. The CompanyBuilder streamlines the process in a readable way.

What I have learned about the process of analysis and design

Planning out the design ahead of time is incredibly important. After I created my original UML diagram and started implementing it in Ruby, I immediately started to realize the mistakes I made. I tried for a very long time to fix the issues in my head and in the code. After struggling and having to redo my implementation a few times I went back to the UML diagram and fixed the problems I found in it. The Ruby implementation was more straight forward after that. Knowing a framework before hand would help as well. Adapting Rail's ActiveRecord associations to my UML diagram proved confusing until I read the documentation thoroughly. Knowing all the major design patterns in my head was cool too because I was always looking for a place in the code with confusing logic that I could simplify. The design patterns don't translate as well in Ruby as they do Java, but I think I was able to adapt.