Human Resources Team
Zachary DiPasquale
Thomas Faticone
Jarrod Cummings
Ephraim Gbadebo

KennUware HR

Github URL:

https://github.com/RIT-SWEN-343-201705-KennUWare/erp-2175-erp-humanresources
https://github.com/tfaticone/api_gateway_343

Table of Contents

HR System Overview

Description

The KennUware HR ERP is designed help KennUware Corp. to manage the data of all their employees. This system is a small part of a whole application that manage all the resources for the KennUware Corp. Each piece of the KennUware ERP is known as a silo or component. Throughout this document we'll describe how each component within our silo interacts with other components and other silos.

Overview

The KennUware HR silo is component of the KennUware ERP, which is designed to manage the data of KennUware Corps. Employees. The silo can manage personal information of its employees, schedule paycheck to be sent, and allow employees to edit their information, request leaves, and managers to enter reviews of their subordinates. As such, the functionality will require the HR silo to interact with other silos. For example, in order to fully calculate paychecks, HR will be required to interact with sales to get information concerning the amount each employee has sold.

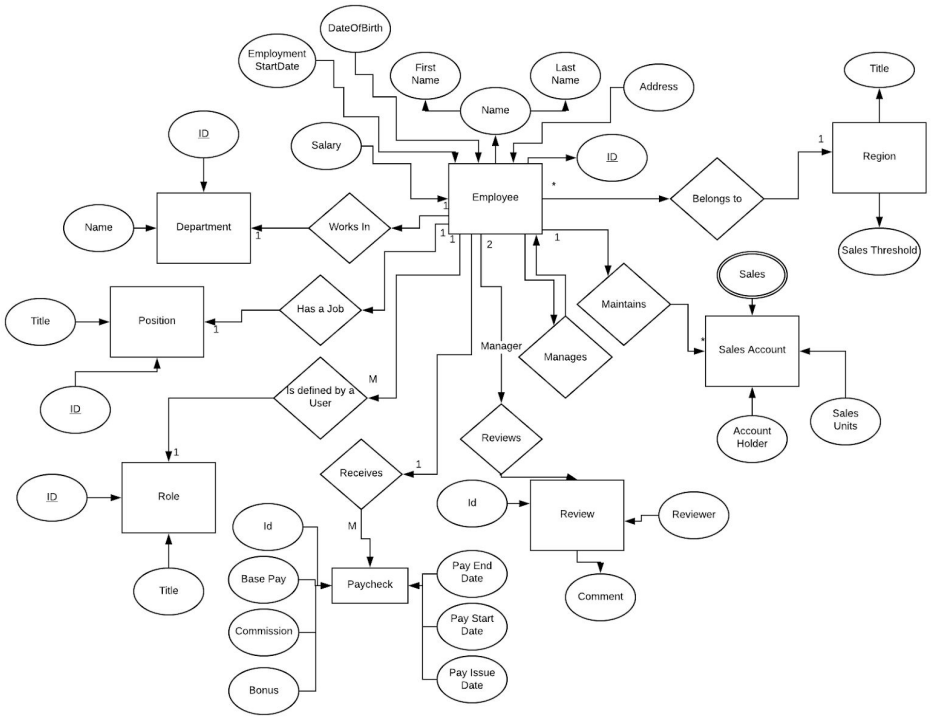


Figure 1-1

In the entity relation diagram (ERD) above (Figure 1-1), the HR web of information and objects are outlined. The major entities that were identified include: Employee, Region, Position, Role, Department, Sales Account, Review, and Paycheck. With the main focused object being Employee all other entities relate back to the Employee. For example, Employee receives paychecks.

Technology Stack

Technology Name	Description
ASP .Net Core	Framework that is used to develop the application by means of an API, MVC, or MVVM-like structure. In our case we chose MVVM due to using Razor Pages.
Entity Framework	ASP .Net Core's ORM. It interfaces between our code and the database. Razor Pages require the Entity Framework to interact with the database/
Razor Pages	The templating language used by ASP .Net core that consists of page models that interact with the business layer to dynamically render information.
Bootstrap	CSS library that keeps a standard look and feel to the website.
MySQL - AWS	Relational database hosted by Amazon web services that allows the HR silo to persist data.
AWS	Amazon Web Services. The cloud-base solution that will run our application.
Docker	Container services designed to run applications independent of the operating system. Thereby making our application runnable across most machines.
JWT	JSON Web Tokens.This will capture information and verify the information is correct and return a token. This token can be passed around the website to verify the logged in employee is valid.

Architecture

Overview

Below shows and describes each layer in our architecture (Fig. 2-1) and each subsystem (Fig. 2-2). There are 3 layers in our silo: Presentation, Domain, and Data Source. Our subsystems

include: Pages, Utilities, API Wrapper, API Controller and, DB Subsystem. In Chart 2-1 and 2-2 we describe what each layer and subsystem does and design choices made in them.

Architecture Diagram

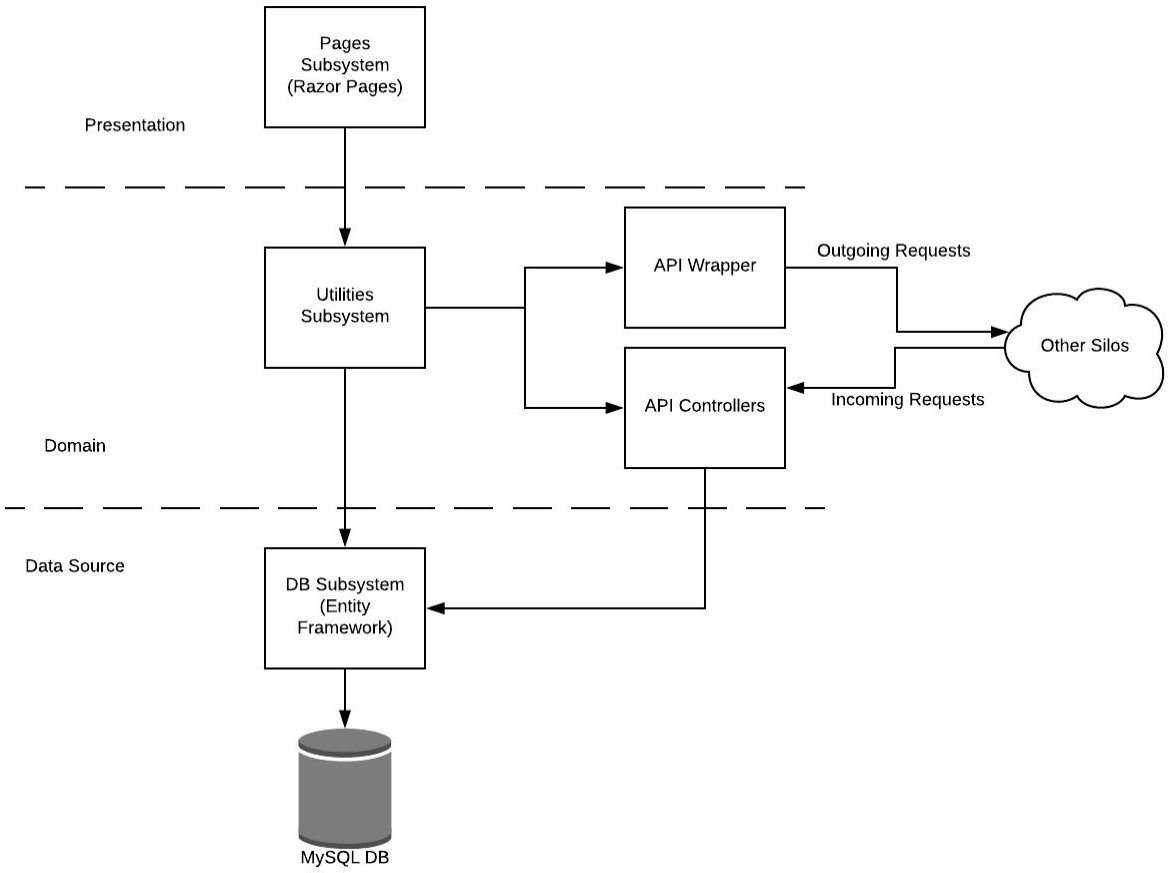


Fig. 2-1

Layer	Description
Data Source	Our data source layer uses the .NET Core Entity Framework to manage our databases and models. Our models are using a Domain Model business logic pattern in order to represent our objects in a clean and organized way.
Domain	Our domain layer holds both .NET Core Web API controller and Utility Objects so we can abstract as much away from the presentation layer as possible and share code between our API's and views
Presentation	The presentation layer is using ASP.NET Core Razor Pages. This is A

	MVVM type architecture. It simplifies much of the building process for us. Each page has its own “PageModel” to determine what can be presented on the view.

Chart 2-1.

UML Class Diagram

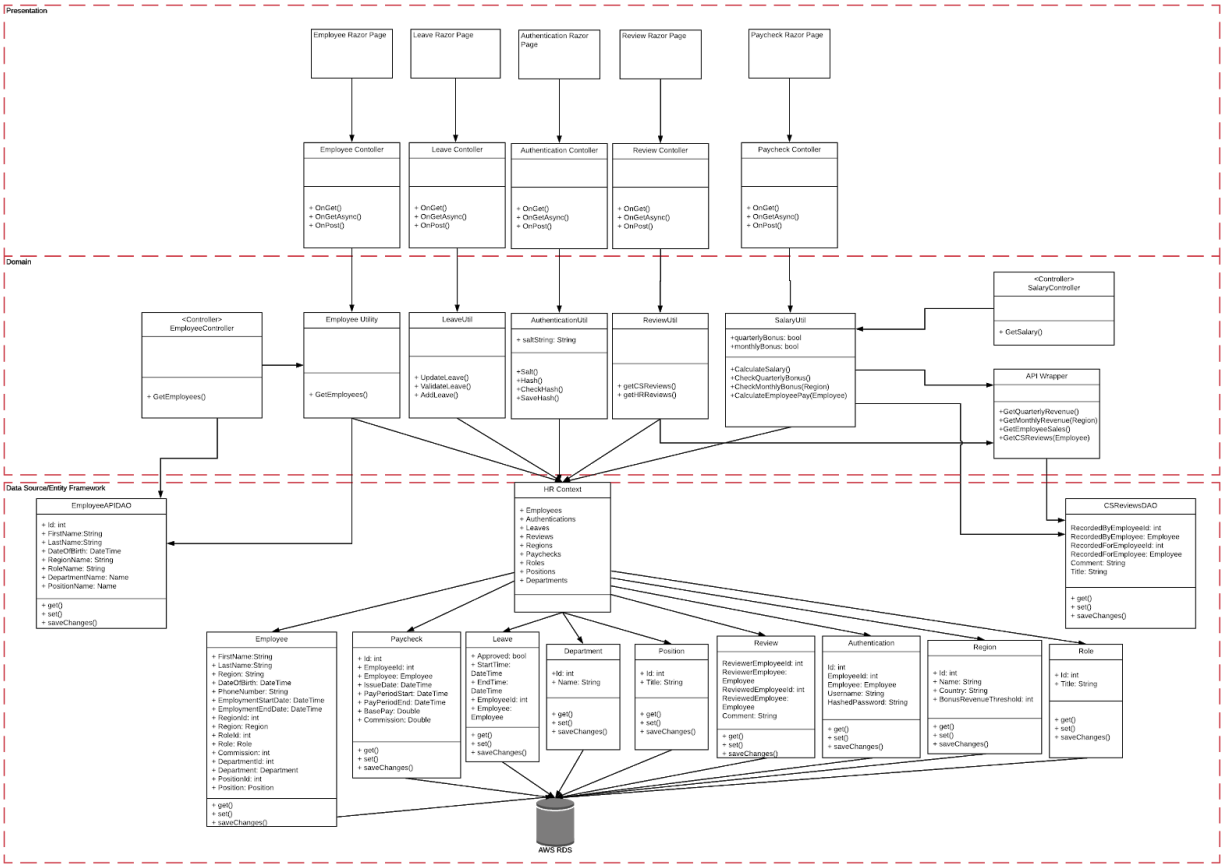


Fig. 2-2

Class Type/Subsystem	Use Case
Model	Each model defines the objects that we are building. Each object defines what will be stored into our database and how they can be interacted with in the views and controllers.
Utility	Utility classes are designed to share any logic needed between the views and the API. Things like calculations and aggregations are placed in these classes to limit the amount of repeated code.
API Controllers	The API Controllers are endpoint used to other silo to access information

	on our system. This includes both employee information and salary calculations.
API Wrapper	Used to access other silos API to gather information we need. This is used for things like gathering employee sales to get commision results from the Sales Silo.
Razor Pages and Page Controllers	Razor pages are a template pattern that is able to gather information provided to it in its page controller. Each controller defines the information available to the view and the view chooses how to present it.

Chart 2-2

APIs

Overview

There are two APIs that we must provide currently. One to get employees and one to generate the paychecks for accounting. These provide ways for other silos to gather information that we store.

Endpoints

Method	URL	Description
GET	/api/employee	Gather employee details with filters. All other silos will likely use this endpoint. Results are returned in a JSON object with pagination.
POST	/api/salary	Accounting will use this endpoint as a trigger. This will calculate each employees paycheck (including base pay, commission pay, and bonus) and return a JSON consisting of the pay start date, pay end date, and the lump sum that needs to be deducted from accounts. This will also store a paycheck history object in HR's database.

Details

GET /api/employee

Returns a list of employees and their information based on filters provided in the url params

Request Object

Name	Type	Required	Description	Example
id	integer	no	ID of the employee.	21
dapartment	string	no	Enables filtering of employees based on department	“Sales”
position	string	no	Filtering employees based on the position	“Sales Representative”
region	string	no	Filtering employees based on region they belong too.	“Southwest”
pageNum	integer	no	The page number of the information the user wants to return based on limit. Default 1	2
limit	integer	no	The maximum number of records returned per page. Default 25	26

Example Requests

URL: /api/employee?id=3

Name	Type	Description	Example
id	integer	ID of the employee.	3
firstName	string	First name of the employee	“Zach”
lastName	string	Last name of the employee	“D”
dateOfBirth	string	DateTime string of the date of birth of the employee	“1996-11-05T00:00:00”
regionName	string	The name of the region the employee is working	“Rochester”

roleName	string	The role of the employee	“Admin”
departmentName	string	The department that the employee is working in	“Accounting”
positionName	string	The position title that the employee is working	“Board Member”

/api/employee?pageNum=1&limit=4

Name	Type	Description	Example
data	array	Contains an array of objects. Each object pertains to one employee (as described above)	Example Below
Page	object	Contains parameters pertaining to the url for the next pagination page.	Example below

Example Response Object


```
{
    "data": [
        {
            "id": 1,
            "firstName": "Zachary",
            "lastName": "DiPasquale",
            "dateOfBirth": "1996-11-05T00:00:00",
            "regionName": "Rochester",
            "roleName": "Admin",
            "departmentName": "HR",
            "positionName": "CEO"
        },
        {
            "id": 3,
            "firstName": "Zach",
            "lastName": "D",
            "dateOfBirth": "1996-11-05T00:00:00",
            "regionName": "Rochester",
            "roleName": "Admin",
            "departmentName": "Accounting",
            "positionName": "Board Member"
        }
    ],
    "page": {
        "nextPage": "?pageNum=2&limit=4"
    }
}
```

POST /api/salary

Calculates and stores all paychecks for all employees within a given range. Returns the total amount expended by the paychecks and the dates the paychecks were calculated for.

Request Object

Name	Type	Description	Example
startDate	string	Indicates the start for the range of dates paychecks should be calculated for.	“06/16/2017”
endDate	string	Indicates the end for the range of dates paychecks should be calculated for. This should be after the start date.	“06/18/2017”
quarterlyBonus	string	Signifies a boolean indicating whether the quarterly bonus should be applied to a paycheck.	“true”
monthlyBonus	string	Signifies a boolean indicating whether the monthly bonus should be applied to a paycheck.	“false”

Return Object

Name	Type	Description	Example
startDate	string	Indicates the start for the range of dates paychecks have been calculated for.	“2008-05-01 12:00:00Z”
endDate	string	Indicates the end for the range of dates paychecks have been calculated for.	“2008-05-02 12:00:00Z”
lumpSum	string	Signified the total amount of money that has been expended by the payout of all paychecks for all employees within the range.	“3400002.24”

APIs to be used

Name	Silo	SOAP/REST	Description
Sales Revenue Information (Quarterly, Monthly, Individual)	Sales	REST	Within salary calculation, there is a need to know individual sales, company sales, and regional sales in order to properly calculate bonus and commission for paycheck. As such, Sales has agreed that there will be an endpoint available to gather this information.
Customer Support Employee Reviews	Customer Support	REST	When an employee goes to their profile page, they are allowed to see various reviews that people have left for them such as managers and customers. As such, Customer support has opened an endpoint that allows HR to GET customer support reviews for a specific employee.

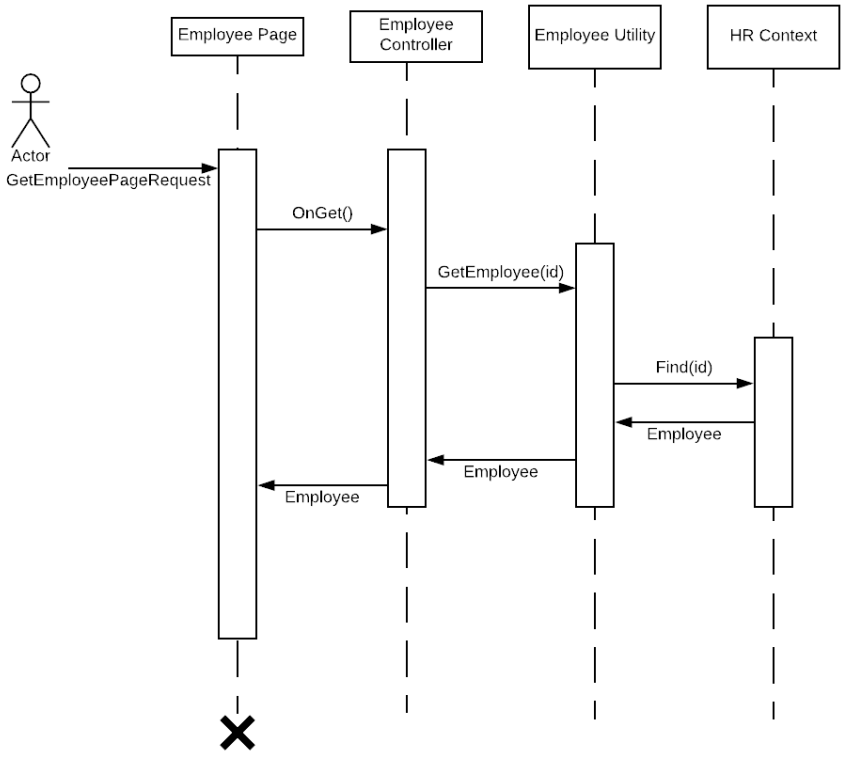
Silo Integration

For testing of these API we are stubbing the data. There are unit tests for each of the stubs to ensure we are being returned the right data type of data. Once we are ready for the API's we will remove the stubs for the API calls. After they are integrated we can test our methods again to ensure they are still returning the same data.

To ensure that we are working with other silos properly we are making sure we have access to each others documents and other API resources. We have communication channels open like Slack and a Google Document so that we can talk about an endpoint whenever we need to.

Sequence Diagrams

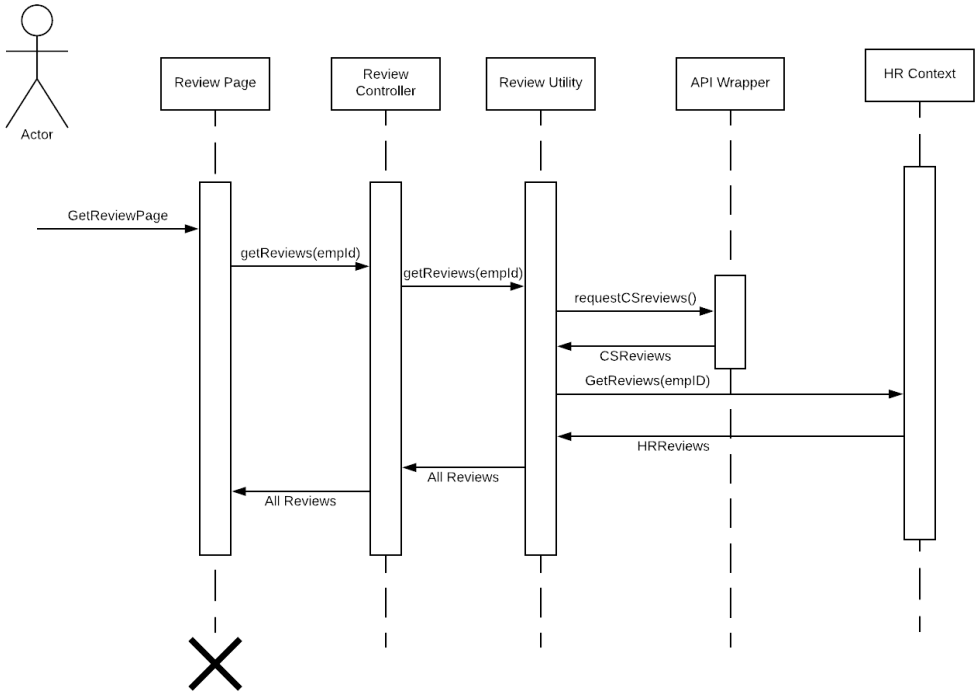
Render a page (with no API calls)



Description

This sequence diagram shows the flow of information through the system as a simple page with no API calls is rendered. It starts with the user making a call to the employee page that calls the unlying controller. This controller will pass the employee id (and the context) to the employee utility. This utility contains all the methods that are needed to go into the data source layer and retrieve the necessary information. As such, when the employee information is returned from the HRContext, the utility passed control back to the controller that inserts the necessary information to the page which is rendered for the user. This sequence is used in a number of spots like leaves.

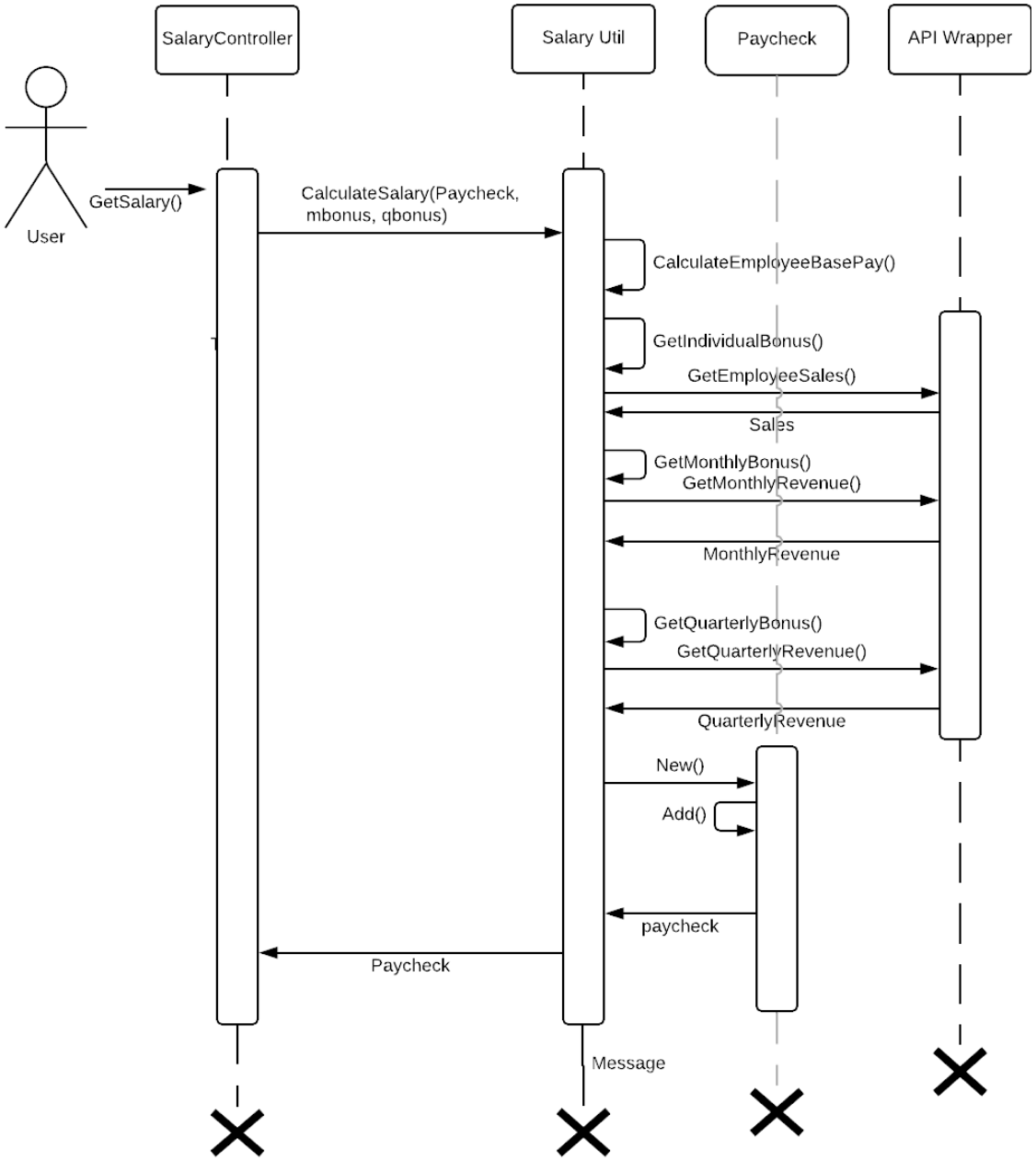
Render a page (with API calls)



Description

This sequence diagram is similar to the first sequence diagram. The only difference is in order to render the reviews page, it is required that a request is made for information from customer support. As such, the Review Utility will use the API Wrapper in order to call out to the customer support API and return the review information. The standard sequence follows after this point.

Calculate Salary/Use Sales API



Description

This sequence diagram depicts the process by which we calculate the payout for an employee. From our salary controller, we call our salary utility class which begins to calculate the amount an employee will be payed. Several calls are made to the Sales Silo's API to get employee

sales information. After bonuses and commission have been properly calculated a paycheck is created and returned.

Hosting Solutions

Overview

We currently are hosted on Amazon Web Services (AWS). This provides a great amount of flexibility, for deploying and managing our applications in the cloud. Since our application can be built in Docker we removed most of the risk of deploying to different services. Our biggest risk of a possible migration would be to migrating our database.

Risk

Possible risks would be a change in schedule, problems with migrating the database, and learning about the new hosting provider. We have built are application in a way that we believe we have mitigated this risk in the best possible way. Features that would need to be switched away from AWS in a migration would be AWS's Relational Database Service (RDS), Elastic Container Service (ECS) and some services to manage urls. All these services have similar counterparts on cloud hosting providers like Azure and Google Cloud Platform.

The role that would be impacted are the configuration/database coordinator. They would be in charge of managing the migration of all the data and services.

API Gateway System Overview

Overview

The KennUware API Gateway has the job of connecting all the silos by means of authentication and forwarding of request. As stated, this API will have the job of authenticating requests and logging in users. This means endpoint will exist in this silo to create users and log them in. For both instances, tokens will be provided to the requestor as these tokens will be used to authenticate any API calls that come across the API gateway. After a request is authenticated, it will be forwarded to the appropriate silo and endpoint.

Technology Stack

Technology Name	Description
Node js	The web server that will handle incoming and outgoing requests. This is the basis for the application.

Express	Javascript library that provides basic functionality like handling requests.
JWT	JSON Web Tokens.This will capture information and verify the information is correct and return a token. This token can be passed around the website to verify the logged in employee is valid.
Sequelize	Javascript Library that acts as the ORM between the code and the database. This provides an intermediary between the business logic and database.
MySQL - AWS	Relational database hosted by Amazon web services that allows the HR silo to persist data.
AWS	Amazon Web Services. The cloud-base solution that will run our application.
Docker	Container services designed to run applications independent of the operating system. Thereby making our application runnable across most machines.

Architecture

Overview

Below shows and describes each layer in our architecture (Fig. 3-1) and each subsystem (Fig. 3-2). There are 2 layers in this API gateway: data layer and business logic. Our subsystems include: Routers and Sequelize.

Architecture Diagram

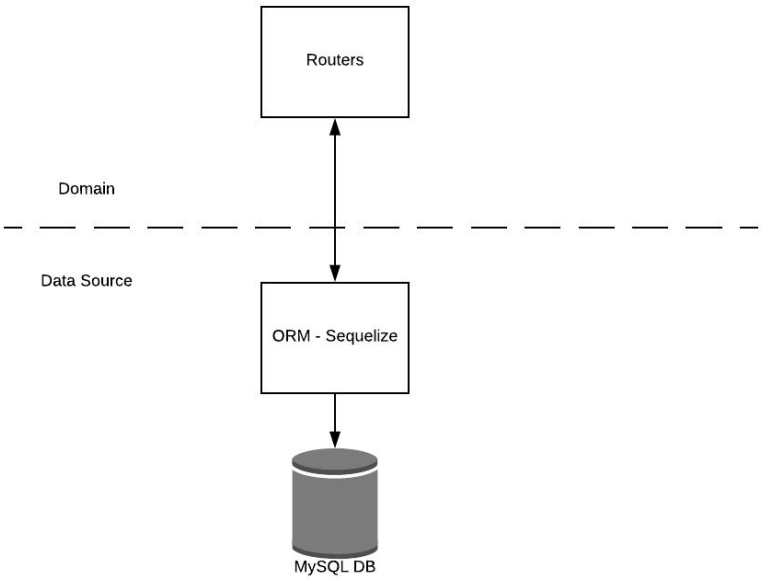
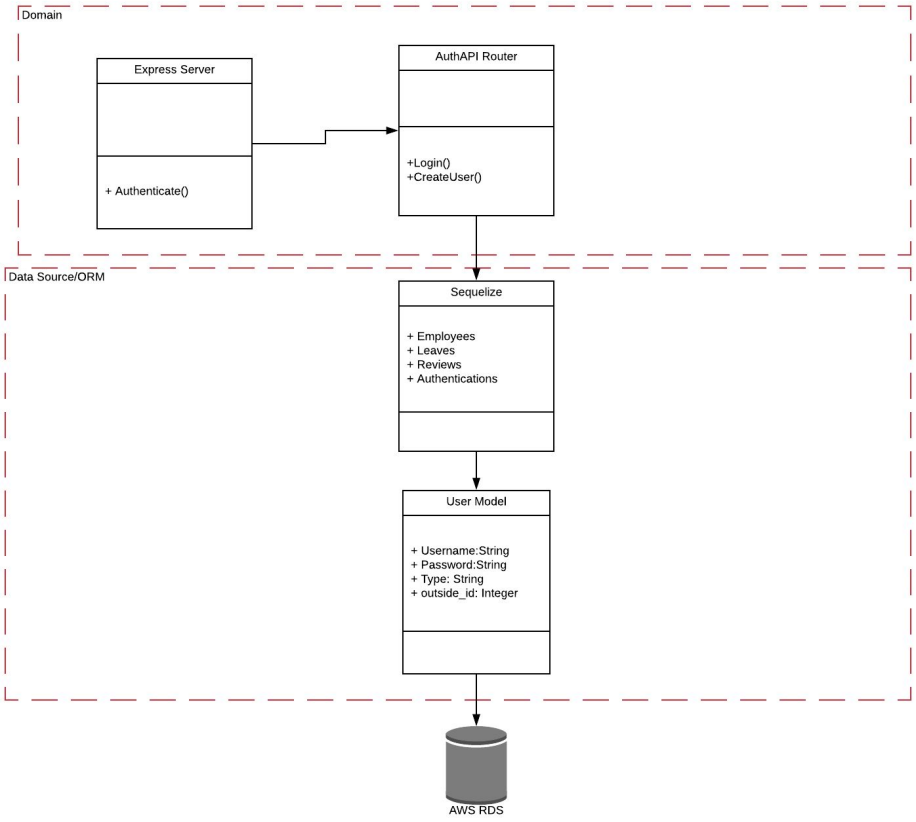


Figure 3-1

Layer	Description
Data Source	Data source consists entirely of Sequelize. This is a javascript ORM that contains models. Specifically, the only model currently is users that contain a username, password, type (employee or customer), and id.
Domain	Domain hold all the logic for routing requests, authenticating people, and creating users. This layer will utilize sequelize to interact with the database.

UML Class Diagram



Layer	Description
Express Server	Handles incoming and outgoing request. Generally forwards requests to appropriate router. If an authenticated endpoint is requested, this will verify the token that has been passed.
AuthAPI Router	Holds all the logic for logging in a user, creating a user, and sending back a token for verification.
Sequelize	ORM that handles all interactions with the database. This utilizes models for standardization.
User Model	The only model that is being handled in this API gateway. This model resembles a generalized user with a username, password, type, and referential id.

APIs

Overview

There are two APIs that we must provide currently. One is to create a user and the other is to log a user in via username and password. All other endpoints are simply passthroughs and documentation should be found in the respective silos. The only difference is a token would be provided that the gateway consumes.

A JWT is returned when authenticated with our system. The JWT returned in Base64 encoded. At this time it is not encrypted and can easy be spoofed. We are planning on discussing encryption in future team coordination meetings. As defined by the JWT standar there are three parts to a JWT: Header, Payload and Secret. The JWT is signed to verify that the token has not changed along the way. An example JWT is displayed below:

```
Header:
{
  "alg": "HS256",
  "typ": "JWT"
}

Payload:
{
  "result": {
    "username": "test2",
    “id”: 1,
    “role”: “Customer Support”
  },
  "iat": 1522635247
}

HMACSHA256(
  base64UrlEncode(Header) + "." +
  base64UrlEncode(Payload),
  `secretkey`
)
```

POST /auth/login

Logs in a user by authenticating against the database.

Request Object

Name	Type	Description	Example
username	string	The username of the account	“test2”
password	string	Password of the account. This is not to be hashed.	“testing123”

Return Object

Name	Type	Description	Example
status	boolean	Indicates whether the request was successful at authenticating or not.	true
token	string	JSON web token that is to be used while interacting with the api gateway endpoints.	See Below
		"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyZXN1bHQiOnsidXNlcm5hbWUiOiJ0ZXN0MiIsInBhc3N3b3JkIjoidGVzdGluZzEyMyIsInR5cGUiOiJjdXN0b21lciJ9LCJpYXQiOjE1MjI2MzUyNDd9.xod4sN7kAC7Fj_uHCAekMJ-ORGuwnQtYxcGeZdaPomI"	

POST /auth/create

Logs in a user by authenticating against the database.

Request Object

Name	Type	Description	Example
username	string	The username of the account	“test2”

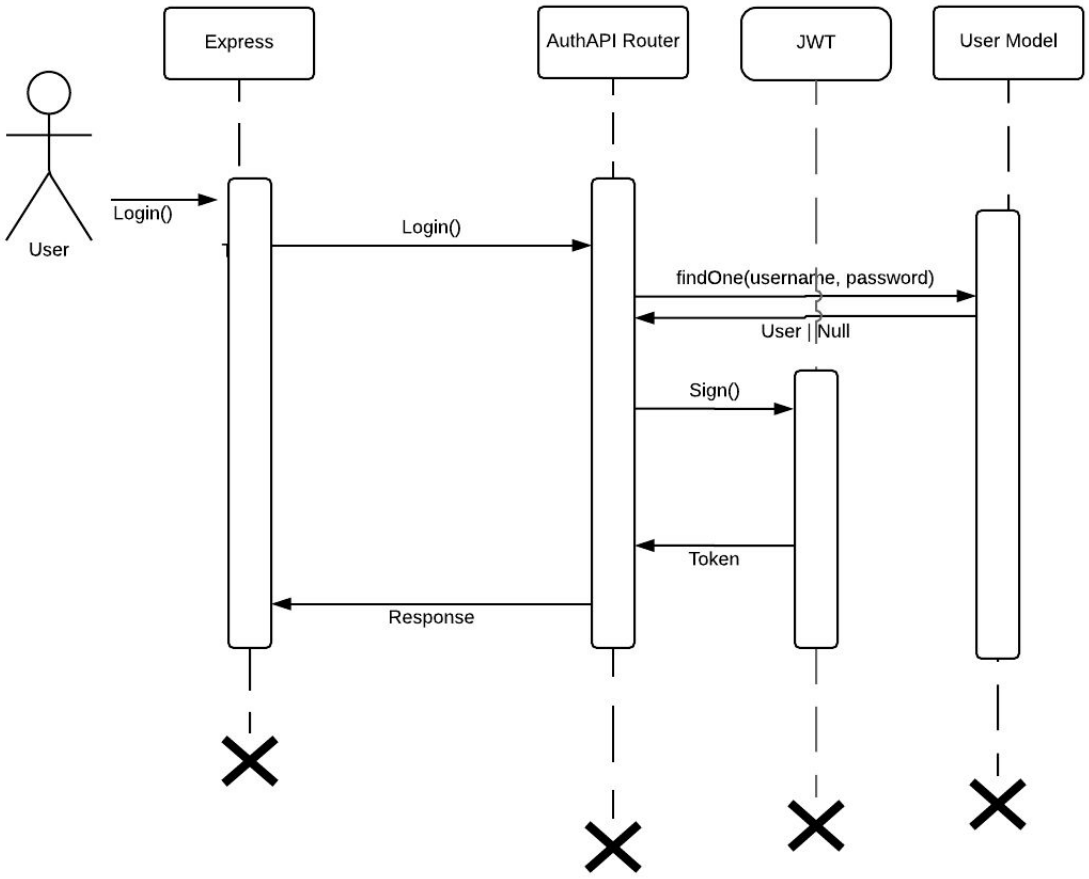
password	string	Password of the account. This is not to be hashed.	“testing123”
type	string	Indicated the type of account being created	“Employee” or “customer”

Return Object

Name	Type	Description	Example
status	boolean	Indicates whether the request was successful at authenticating or not.	true
token	string	JSON web token that is to be used while interacting with the api gateway endpoints.	See Below
		"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyZXN1bHQiOnsidXNlcm5hbWUiOiJ0ZXN0MiIsInBhc3N3b3JkIjoidGVzdGluZzEyMyIsInR5cGUiOiJjdXN0b21lciJ9LCJpYXQiOjE1MjI2MzUyNDd9.xod4sN7kAC7Fj_uHCAekMJ-ORGuwnQtYxcGeZdaPomI"	
message	String	Indicates what type of errors or success the request has encountered	“Account Created”

Sequence Diagrams

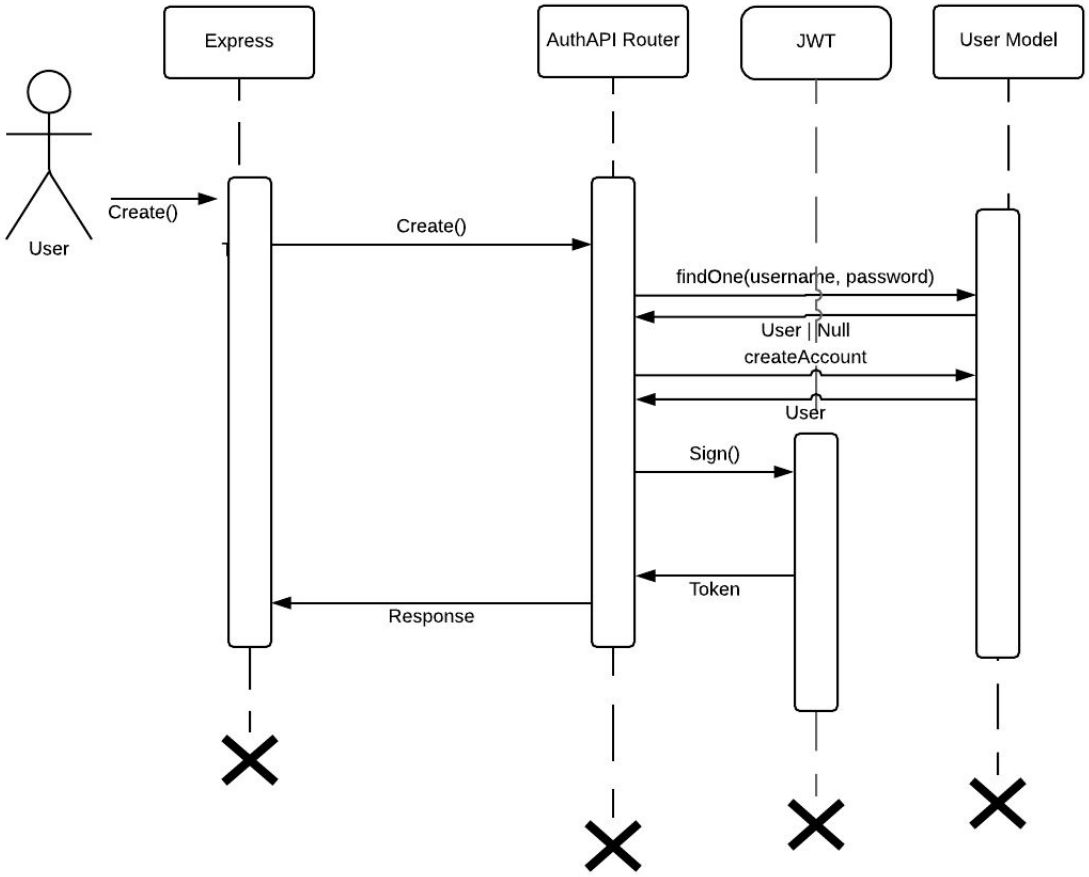
Login



Description

In this sequence, the user will sent a request to the Express server. The express server will allow the AuthAPI Router to handle the request. The Router will check to see if there is a user that matches the credentials set. If there is a user, JWT will sign the user account and return a token. This token will be sent back to express to be bundled with the response.

Create Account



Description

This sequence diagram displays the create user endpoint. When a request is received by the express server, it will be forwarded to the AuthAPI router. This will first check if there is already an account with the username then return the results. If there is **no** account that matches, the AuthAPI router will create a user. When this is complete, JWT will sign the new account and return a token. This token is returned and bundled with the response.