Eight:18 Capstone Project

Cameron Wilcox, Tanis Hadwin, Jared Weiss

Gianforte School of Computing, Montana State University

CSCI 483R: Interdisciplinary Project

Dr. Clemente Izurieta

Spring 2023

I. Introduction

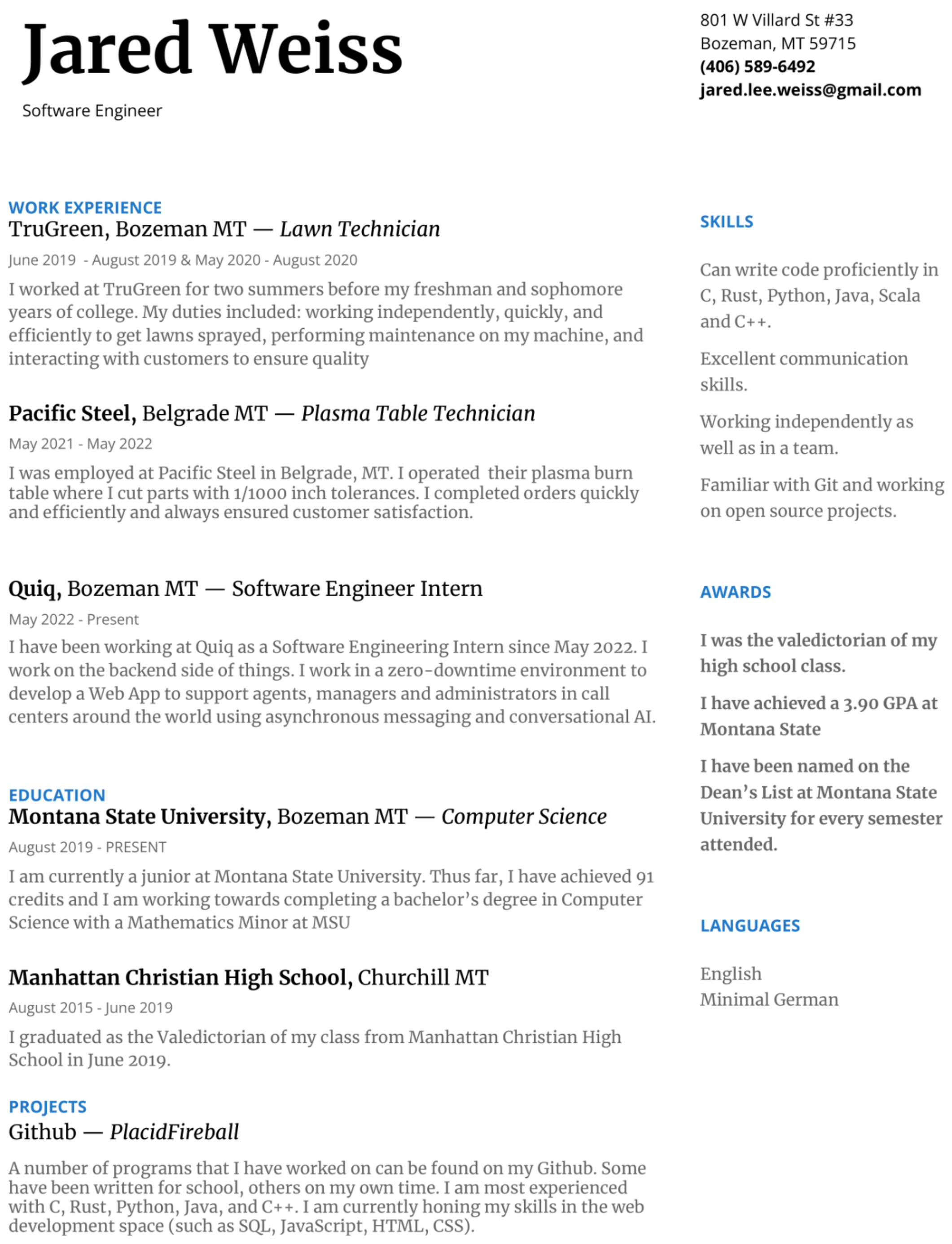
Eight:18 is a sponsorship solutions company that looks to assist potential and current events with the tedious task of selling, activating and maintaining sponsorships for any kind of event. Eight:18 also does consulting for sponsors, to coach and guide them through making decisions so that they are always up to date on the best strategies for their brand. Our goal for this capstone project is to assist Eight:18 in their endeavor to support events by developing a piece of software that tracks activation benefits. This process consists of improving efficiencies, making tracking sponsors less tedious, transitioning the task to digital and automated storage, automating report generation, and presenting all of this in an attractive and cohesive manner.

The first and most important goal that we look to achieve with this capstone project was to take the current way Eight:18 tracks information about sponsors, which is the tedious task of inputting information by hand, i.e. manually filling out spreadsheets and writing information on whiteboards and papers, and move it to a digital web application that will make the process much smoother and save Eight:18 precious time. We will accomplish this through the use of a MySQL database and Redis. The backend of our app will be written in Scala, using DropWizard.

Another issue that Eight:18 faces is the organization of photos and other media that is not quite suitable for a structure like a database. Another issue here is that it requires vastly more storage space than a database might. This required the use of AWS's storage services. We will use Amazon Web Services' S3 to store and retrieve these pieces of information so that things like this can be later used to track what sponsorship mediums were used where, and have concrete evidence of it.

Eight:18 also offers consultation and coaching for sponsors that are currently clients or for sponsors that are looking to get into events such as rodeos, fairs, and sporting events. To accomplish this, they need to create and analyze reports that will give them an idea of what the road map will look like for a sponsor, and, to track sponsors progress and revenue to guide them through these events. We will use a dashboard creator to create live reports on this web app. This allows for up-to-date reports on all clients so Eight:18 never misses a beat.

II. Qualifications



Jared Weiss

Software Engineer

801 W Villard St #33
Bozeman, MT 59715
(406) 589-6492
jared.lee.weiss@gmail.com

WORK EXPERIENCE

TruGreen, Bozeman MT — *Lawn Technician*

June 2019 - August 2019 & May 2020 - August 2020

I worked at TruGreen for two summers before my freshman and sophomore years of college. My duties included: working independently, quickly, and efficiently to get lawns sprayed, performing maintenance on my machine, and interacting with customers to ensure quality

Pacific Steel, Belgrade MT — *Plasma Table Technician*

May 2021 - May 2022

I was employed at Pacific Steel in Belgrade, MT. I operated their plasma burn table where I cut parts with 1/1000 inch tolerances. I completed orders quickly and efficiently and always ensured customer satisfaction.

Quiq, Bozeman MT — Software Engineer Intern

May 2022 - Present

I have been working at Quiq as a Software Engineering Intern since May 2022. I work on the backend side of things. I work in a zero-downtime environment to develop a Web App to support agents, managers and administrators in call centers around the world using asynchronous messaging and conversational AI.

EDUCATION

Montana State University, Bozeman MT — *Computer Science*

August 2019 - PRESENT

I am currently a junior at Montana State University. Thus far, I have achieved 91 credits and I am working towards completing a bachelor's degree in Computer Science with a Mathematics Minor at MSU

Manhattan Christian High School, Churchill MT

August 2015 - June 2019

I graduated as the Valedictorian of my class from Manhattan Christian High School in June 2019.

PROJECTS

Github — *PlacidFireball*

A number of programs that I have worked on can be found on my Github. Some have been written for school, others on my own time. I am most experienced with C, Rust, Python, Java, and C++. I am currently honing my skills in the web development space (such as SQL, JavaScript, HTML, CSS).

SKILLS

Can write code proficiently in C, Rust, Python, Java, Scala and C++.

Excellent communication skills.

Working independently as well as in a team.

Familiar with Git and working on open source projects.

AWARDS

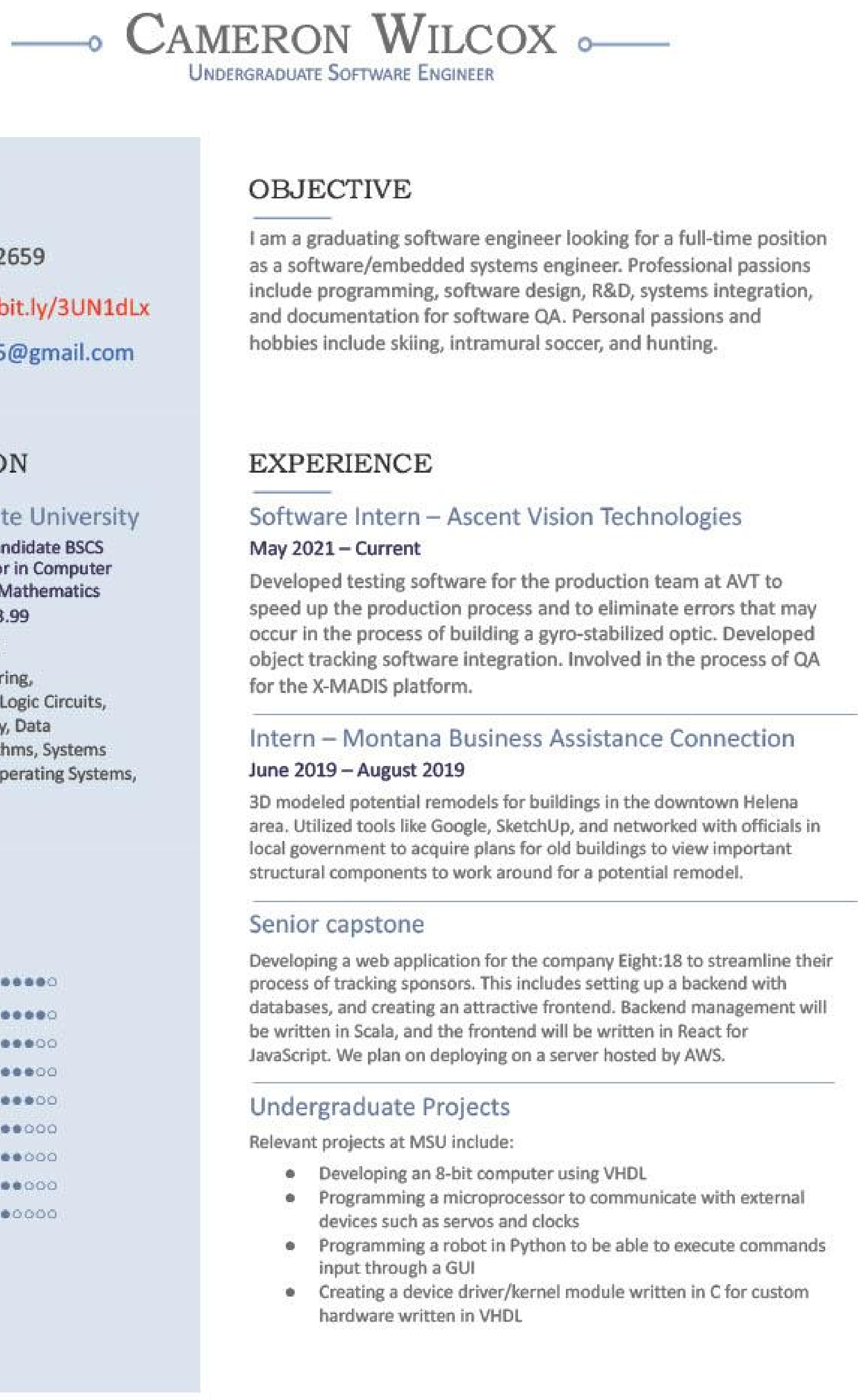
I was the valedictorian of my high school class.

I have achieved a 3.90 GPA at Montana State

I have been named on the Dean's List at Montana State University for every semester attended.

LANGUAGES

English
Minimal German



CAMERON WILCOX

UNDERGRADUATE SOFTWARE ENGINEER

CONTACT

307-851-2659

LinkedIn: bit.ly/3UN1dLx

cjwilcox15@gmail.com

EDUCATION

Montana State University

Undergraduate candidate BSCS Spring 2023, minor in Computer Engineering, and Mathematics

Cumulative GPA: 3.99

Relevant Courses:

Software Engineering, Microprocessors, Logic Circuits, Computer Security, Data Structures/Algorithms, Systems Administration, Operating Systems, Networks

SKILLS

C++	●●●●○
LINUX	●●●●○
PYTHON	●●●○○
JAVA	●●●○○
C	●●●○○
LUA	●●○○○
VHDL	●●○○○
ASSEMBLY	●●○○○
MATLAB	●○○○○

OBJECTIVE

I am a graduating software engineer looking for a full-time position as a software/embedded systems engineer. Professional passions include programming, software design, R&D, systems integration, and documentation for software QA. Personal passions and hobbies include skiing, intramural soccer, and hunting.

EXPERIENCE

Software Intern – Ascent Vision Technologies

May 2021 – Current

Developed testing software for the production team at AVT to speed up the production process and to eliminate errors that may occur in the process of building a gyro-stabilized optic. Developed object tracking software integration. Involved in the process of QA for the X-MADIS platform.

Intern – Montana Business Assistance Connection

June 2019 – August 2019

3D modeled potential remodels for buildings in the downtown Helena area. Utilized tools like Google, SketchUp, and networked with officials in local government to acquire plans for old buildings to view important structural components to work around for a potential remodel.

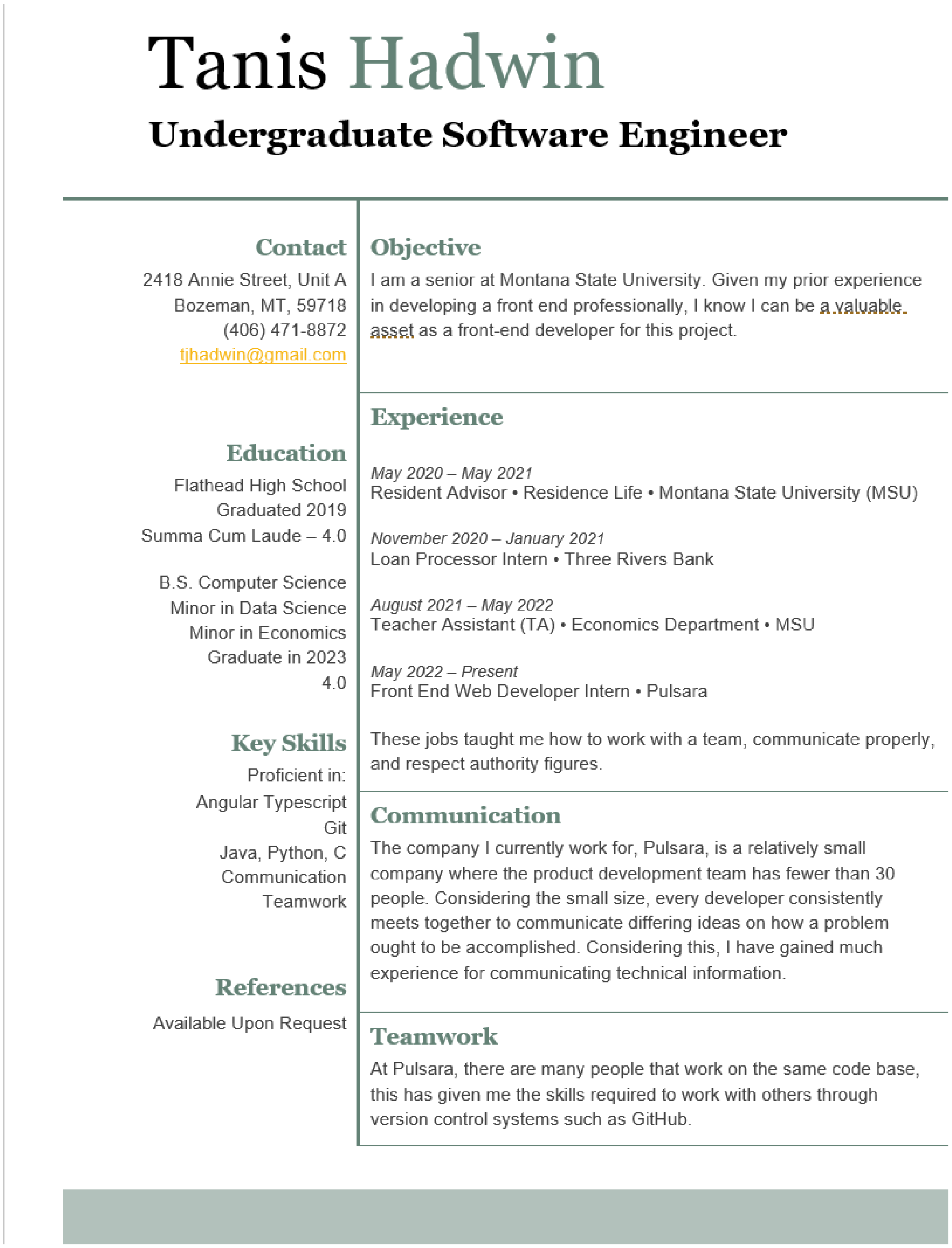
Senior capstone

Developing a web application for the company Eight:18 to streamline their process of tracking sponsors. This includes setting up a backend with databases, and creating an attractive frontend. Backend management will be written in Scala, and the frontend will be written in React for JavaScript. We plan on deploying on a server hosted by AWS.

Undergraduate Projects

Relevant projects at MSU include:

- Developing an 8-bit computer using VHDL
- Programming a microprocessor to communicate with external devices such as servos and clocks
- Programming a robot in Python to be able to execute commands input through a GUI
- Creating a device driver/kernel module written in C for custom hardware written in VHDL



Tanis Hadwin

Undergraduate Software Engineer

Contact

2418 Annie Street, Unit A
Bozeman, MT, 59718
(406) 471-8872
tjhadwin@gmail.com

Education

Flathead High School
Graduated 2019
Summa Cum Laude – 4.0

B.S. Computer Science
Minor in Data Science
Minor in Economics
Graduate in 2023
4.0

Key Skills

Proficient in:
Angular Typescript
Git
Java, Python, C
Communication
Teamwork

References

Available Upon Request

Objective

I am a senior at Montana State University. Given my prior experience in developing a front end professionally, I know I can be a valuable asset as a front-end developer for this project.

Experience

May 2020 – May 2021
Resident Advisor • Residence Life • Montana State University (MSU)

November 2020 – January 2021
Loan Processor Intern • Three Rivers Bank

August 2021 – May 2022
Teacher Assistant (TA) • Economics Department • MSU

May 2022 – Present
Front End Web Developer Intern • Pulsara

These jobs taught me how to work with a team, communicate properly, and respect authority figures.

Communication

The company I currently work for, Pulsara, is a relatively small company where the product development team has fewer than 30 people. Considering the small size, every developer consistently meets together to communicate differing ideas on how a problem ought to be accomplished. Considering this, I have gained much experience for communicating technical information.

Teamwork

At Pulsara, there are many people that work on the same code base, this has given me the skills required to work with others through version control systems such as GitHub.

III. Background

For the Eight:18 capstone project we were given the goal of moving the current system that Eight:18 uses of tracking sponsors, onto an online platform to streamline the process and make organization easier. The first step in this process was learning how Eight:18 currently organizes the information that they collect on sponsors. They currently use the business spreadsheet program Excel (Gillis, 2021) to organize all of the sponsor data. This data includes: how many people visited an event, what sponsors were present, what mediums were used for what sponsors, for certain mediums like radio spots or social media videos, how long were those slots. For mediums like posters, banners and billboards, the dimensions of the medium are recorded. Data on what items sponsors give out is also recorded.

Once we had gathered the spreadsheets that Eight:18 had given us, we set out to figure out how to organize this data within a database, using MySQL as our database technology of choice. A database is a set of structured data, based on tables with fields ("MySQL 8.0 Reference Manual", 2022). A backend management system may query against the database for entries in certain tables, and the database will return information on the requested entry.

We will also be using Redis on the backend to store smaller packets of data and non-permanent data stores. Redis is much faster as it is an in-memory data-store, so we will use it to keep "hot" data in Redis and keep "cold" data in the MySQL Database.

Once the database is set up, a backend management system is needed to handle requests that come in from the frontend. Our backend will be written in Scala, a functional programming language built on the JVM. The main libraries that we will be using are Dropwizard ("Dropwizard Core") which includes a range of libraries consisting of Jackson, Jetty, Logback and a few others. Jackson is used for JSON serialization and deserialization. Jetty is used for managing servlets on the backend so that we can provide a multi-threaded environment that will scale as Eight:18 begins to transfer all their data onto the web app. Jersey is used to manage a full-featured RESTful environment. Jackson is Java's most popular and best library for JSON management. Logback is the successor for Log4j, and is the JVM's best logging framework. ("Dropwizard Core")

JSON stands for JavaScript Object Notation. It is a method of transferring and storing information in the form of human-readable text. This allows for efficient transfer of information between the frontend and the backend. We will use this standard for storing and transferring our information.

All of these programs will be hosted in the cloud using Amazon Web Services. Amazon Web Services is a popular and reliable cloud based hosting service that we will be using to host our backend, MySQL database, Redis Server and also a web server for our frontend. While we could host it on our own servers, AWS offers a more reliable and faster alternative than using local machines. We will host Eight18's servers in the 'us-west' cluster in AWS (What is AWS, 2022).

Of course, all of this data storage and processing means nothing without clear and concise ways to both collect and represent the data to users. For this purpose we will also be developing a frontend web application for users to easily interface with the backend system. To do this, we will be utilizing the React JS web framework, along with various other React native libraries to assist in data presentation, routing, and several other relevant tasks. The React JS framework provides a user-friendly way to develop an easily scalable web application.

IV. Work Schedule

Figure 1

1st Semester CSCI 482R Work Schedule

Monday	**Tuesday**	**Wednesday**	**Thursday**	**Friday**
Virtual Team Meeting from 5:30 pm - 7:00 pm		Virtual Team Meeting from 5:30 pm - 7:00 pm	In Person Team Meeting and Meet with Eight:18 Team from 4:30 pm - 6:30 pm	

Figure 2

2nd Semester CSCI 483R Work Schedule

Monday	**Tuesday**	**Wednesday**	**Thursday**	**Friday**
	In Person Team Meeting in Software Factory from 10:50 pm - 12:00 pm		In Person Team Meeting in Software Factory from 10:50 pm - 12:00 pm Virtual Meeting with Eight:18 Team During In Person Meeting	Virtual Team Meeting from 4:00 pm - 5:00 pm (Optional)

V. Timelines and Deliverables

Figure 3

1st Semester Timeline and Deliverables

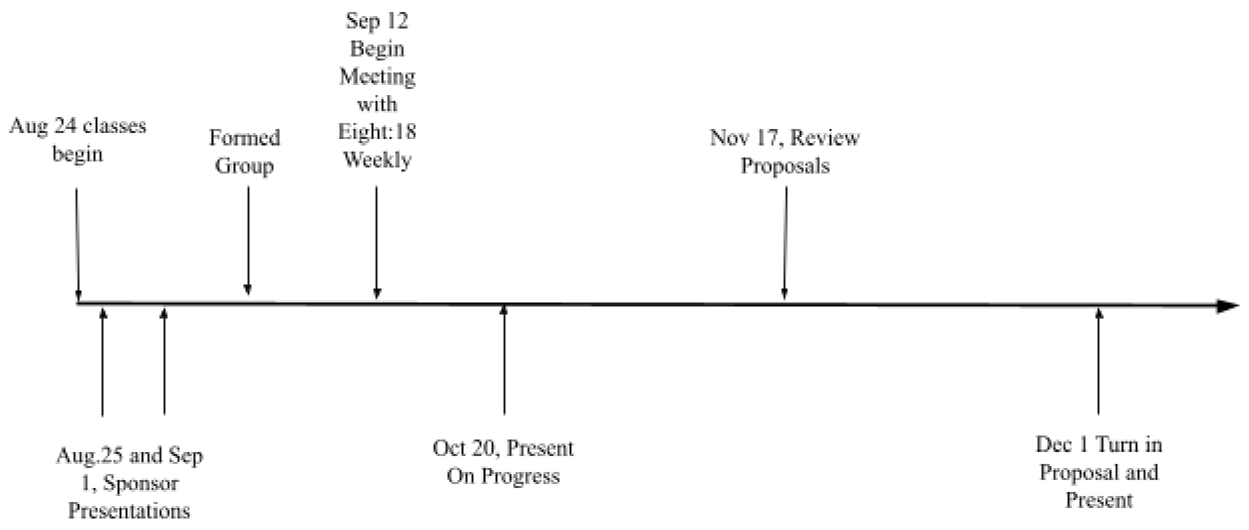
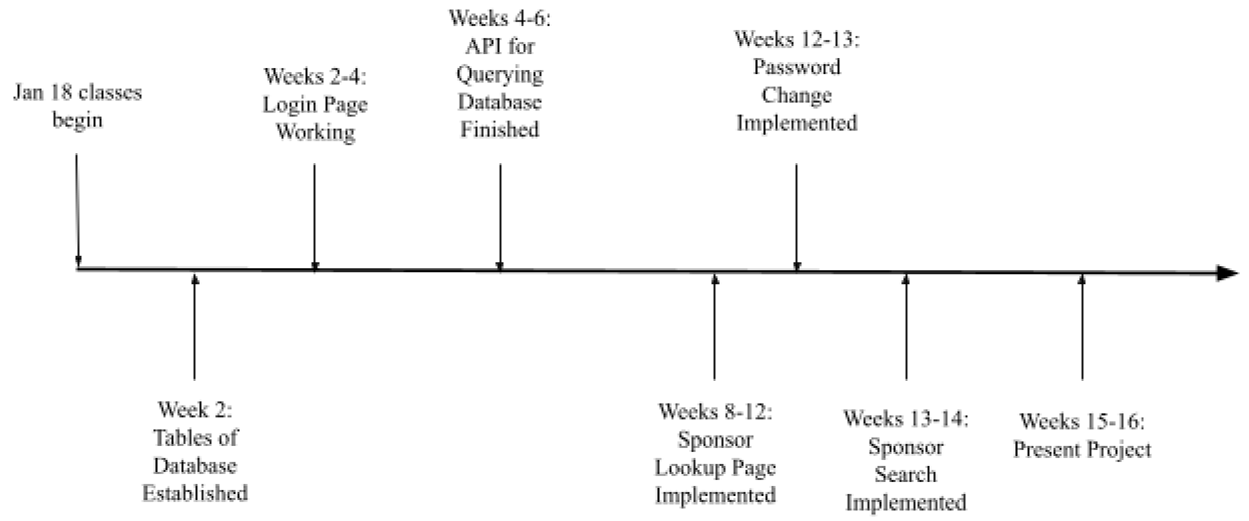

Aug 24 classes begin
Formed Group
Sep 12 Begin Meeting with Eight:18 Weekly
Nov 17, Review Proposals
Aug.25 and Sep 1, Sponsor Presentations
Oct 20, Present On Progress
Dec 1 Turn in Proposal and Present

Figure 4

2nd Semester Timeline and Deliverables


Jan 18 classes begin
Weeks 2-4: Login Page Working
Weeks 4-6: API for Querying Database Finished
Weeks 12-13: Password Change Implemented
Week 2: Tables of Database Established
Weeks 8-12: Sponsor Lookup Page Implemented
Weeks 13-14: Sponsor Search Implemented
Weeks 15-16: Present Project

Discussion of Timelines and Deliverables

For this capstone project we broke the task of supporting Eight:18 into different phases: the first phase is getting the foundation down for future development to take place, this is also the phase that is necessary to consider the project a success. The second phase is to expand on the interface and to begin working on the formulas for proper report generation. This will include feeding these reports to the front end. The third and final phase would include adding the ability to add sponsors to the app, so that they can communicate directly with Eight:18, as well as adding features like adding financial objectives for sponsor, adding alerts for upcoming deadlines, producing "snapshots" of the sponsors profile for the sponsor to look at.

Phase two and three are not currently included on the timeline due to the fact that they are not required for the project to be considered a success. However, in the event that we are ahead in the first phase, we will move on to the second when it becomes complete. Deliverables would include: creation of formulas for report generation, implementation of those formulas and-feeding the data from the formulas to the front end.

VI. Agile Life Cycle Approach

For maximum flexibility, and to ensure consistent progress on this project, our group decided to use the agile life cycle for our project cycles. Agile is designed for an iterative process that allows for high flexibility of week to week checkups, where the trajectory of the work can be changed if needed. We will use Trello for our progress tracker to monitor what tasks we have planned, what tasks are in the works, what tasks are complete, what bugs need to be fixed and even what tasks need to be pushed to a later phase. We started by meeting with Eight:18 and established what tasks should go into what phases, and then broke the tasks down into milestones that we can deliver on throughout the semester.

Roles and Responsibilities

Cameron Wilcox	Tanis Hadwin	Jared Weiss
● Develop for both frontend and backend ● Quality assurance ● Schedule meetings ● Cloud deployment	● Lead on frontend development ● Quality assurance ● Frontend/UI design ● Design API	● Lead on backend development ● Quality assurance ● Database manager ● Design API

VII. **Proposal Statement**

Through this design process we have implemented the necessary functional, non-functional, interface, and performance requirements. Additionally we referenced architectural design documents and used development standards throughout the process of development. Our goal for the Eight:18 Capstone Project was to create a solid foundation for tracking sponsor information, as well as translating the process from a tedious manual one to a more automatic and efficient one.

Functional/Non-Functional Requirements

Functional	**Non-Functional**
Simple interface for entering/storing sponsor information	Create scalable infrastructure
Create dashboard for displaying information	Create reliable infrastructure
Create actionable reports	Create codebase that is easy for future devs to pick up

These non-functional requirements are met through the use of a service like AWS. This way, in the event that something goes down, the app will experience little to no delays in use or loss of information.

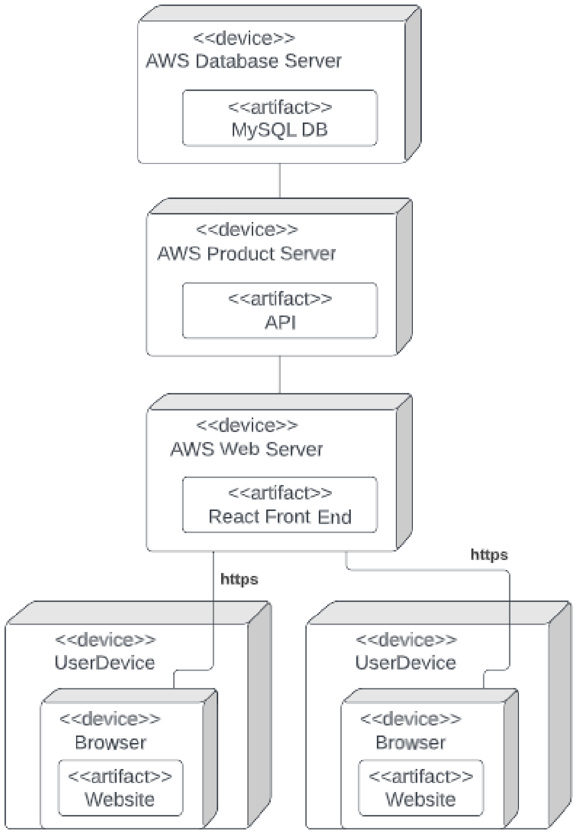
We will use the following UML diagrams: deployment, component, and class diagrams; to portray the architectural designs built into the code. Using these designs, we foresee that this software will be able to perform at a certain level to meet Eight:18's requirements. Once development on this project is picked up again post-graduation, we expect that it will streamline the Eight:18's workflow and potential to an even greater degree.

We are using an agile approach to get our work done while keeping the needs of Eight:18 in mind during development. Within this approach we will be using a myriad of tools to accomplish this: Visual Studio Code, IntelliJ, Discord, Zoom, GitHub, and email. Visual Studio Code and IntelliJ will both be used for programming and for debugging. Visual Studio will be used more for the frontend side, while IntelliJ will be used more for the backend. This is because IntelliJ has plugins and a more efficient environment for writing backend code, where VSCode has access to plugins that IntelliJ does not for frontend development. Discord, Zoom, email and GitHub will be used to collaborate during our weekly meetings with Eight:18, as well as share code and share thoughts that may not need an entire face-to-face meeting.

VIII. Methods

Architectural Design Documents Discussion

Figure 5

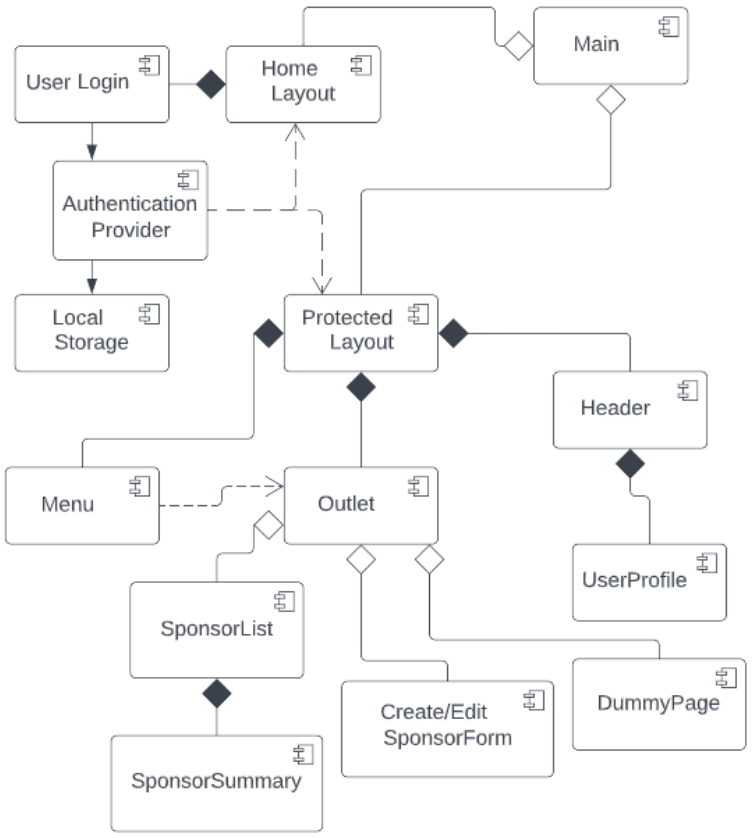


Top Level Architecture:

The very top layer of the design of this website can be shown by the UML Deployment diagram in Figure 5. The first notable aspect of this diagram is the host servers. The current objective of the application is to deploy it on Amazon Web Services, AWS, servers. The specifics of this decision will be discussed further in the Trade-Off Discussion below. The other notable attribute is how this diagram is indicative of the design pattern that we have selected for our application.

The Adapter Pattern is a design pattern that involves an entity intended to function as a translator between two, otherwise incompatible entities, so as to be able to communicate with each other. In our case, the Adapter is the API that allows for communication between the front end web application and the database. Front end web frameworks are only intended to temporarily store data with the intent of presenting it. Databases are designed to store data and nothing else. As a result, these two things need each other in order to function, but the API is needed as an intermediary to ensure that the front end only receives data that it actually needs so as to not negatively affect the speed and performance of the application.

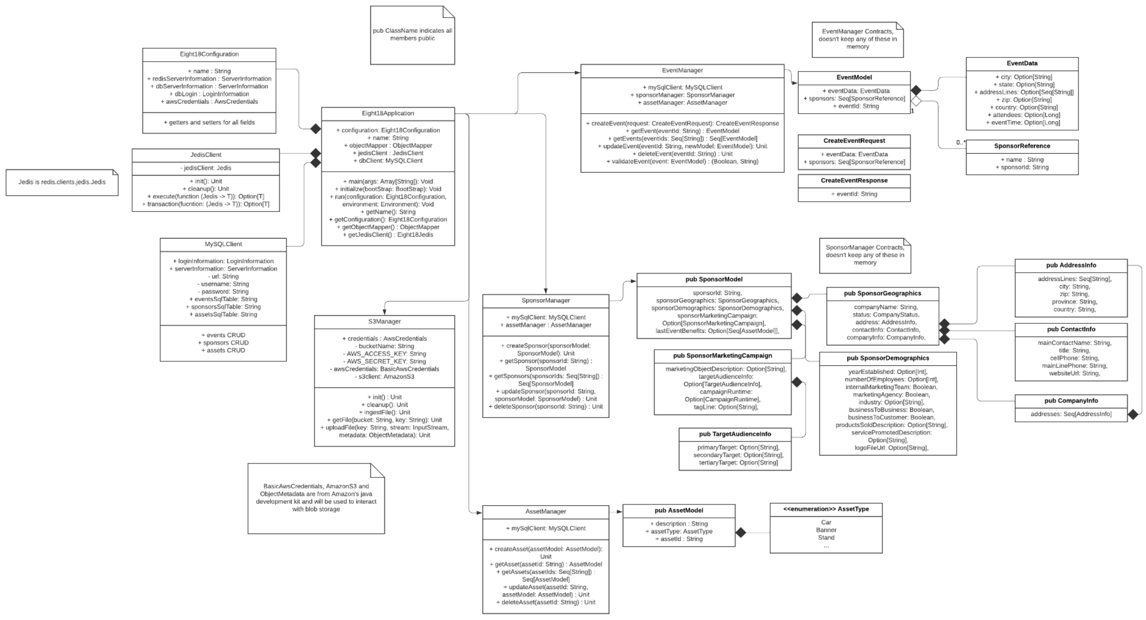
Front End Architecture: **Figure 6**



The architecture of the front end can most easily be explained by the UML Component Diagram shown in Figure 6. In the context of front end web development, the idea of a Class Diagram doesn't fit, so it is fully explained by what is present in the Component Diagram. The app starts from the main root component. The first thing that happens is dividing the app between the Protected Layout and Home Layout authentication checks, which only serve to ensure that urls are only accessible to certain authentication statuses. For example, in order to access the login component, there cannot be a user currently authenticated in local storage, but to view the sponsor list, authentication is required. The act of logging in adds an authenticated user to local storage, and logging out removes it, so these layouts are always ensured to hold the proper state.

The outlet component describes the process of choosing which page component to show. In this case the only options are the sponsor list which is composed of individual sponsor summaries, the create and edit sponsor form, or the dummy page, which is a stand in for every page that will eventually exist in the web app.

<u>Backend Architecture:</u> **Figure 7**


pub ClassName indicates all members public
Eight18Configuration
+ name : String
+ redisServerInformation : ServerInformation
+ dbServerInformation : ServerInformation
+ dbLogin : LoginInformation
+ awsCredentials : AwsCredentials
+ getters and setters for all fields
Jedis is redis.clients.jedis.Jedis
JedisClient
- jedisClient: Jedis
+ init(): Unit
+ cleanup(): Unit
+ execute(function (Jedis -> T)): Option[T]
+ transaction(fucntion: (Jedis -> T)): Option[T]
MySQLClient
+ loginInformation: LoginInformation
+ serverInformation: ServerInformation
- url: String
- username: String
- password: String
+ eventsSqlTable: String
+ sponsorsSqlTable: String
+ assetsSqlTable: String
+ events CRUD
+ sponsors CRUD
+ assets CRUD
Eight18Application
+ configuration: Eight18Configuration
+ name: String
+ objectMapper : ObjectMapper
+ jedisClient : JedisClient
+ dbClient: MySQLClient
+ main(args: Array[String]): Void
+ initialize(bootStrap: BootStrap): Void
+ run(configuration: Eight18Configuration, environment: Environment): Void
+ getName(): String
+ getConfiguration(): Eight18Configuration
+ getObjectMapper() : ObjectMapper
+ getJedisClient() : Eight18Jedis
S3Manager
+ credentials : AwsCredentials
- bucketName: String
- AWS_ACCESS_KEY: String
- AWS_SECRET_KEY: String
- awsCredentials: BasicAwsCredentials
- s3client: AmazonS3
+ init() : Unit
+ cleanup(): Unit
+ ingestFile(): Unit
+ getFile(bucket: String, key: String): Unit
+ uploadFile(key: String, stream: InputStream, metadata: ObjectMetadata): Unit
BasicAwsCredentials, AmazonS3 and ObjectMetadata are from Amazon's java development kit and will be used to interact with blob storage
EventManager
+ mySqlClient: MySQLClient
+ sponsorManager: SponsorManager
+ assetManager: AssetManager
+ createEvent(request: CreateEventRequest): CreateEventResponse
+ getEvent(eventId: String) : EventModel
+ getEvents(eventIds: Seq[String]) : Seq[EventModel]
+ updateEvent(eventId: String, newModel: EventModel): Unit
+ deleteEvent(eventId: String) : Unit
+ validateEvent(event: EventModel) : (Boolean, String)
EventManager Contracts, doesn't keep any of these in memory
EventModel
+ eventData: EventData
+ sponsors: Seq[SponsorReference]
+ eventId: String
1
EventData
+ city: Option[String]
+ state: Option[String]
+ addressLines: Option[Seq[String]]
+ zip: Option[String]
+ country: Option[String]
+ attendees: Option[Long]
+ eventTime: Option[Long]
0..*
SponsorReference
+ name : String
+ sponsorId: String
CreateEventRequest
+ eventData: EventData
+ sponsors: Seq[SponsorReference]
CreateEventResponse
+ eventId: String
SponsorManager Contracts, doesn't keep any of these in memory
SponsorManager
+ mySqlClient: MySQLClient
+ assetManager : AssetManager
+ createSponsor(sponsorModel: SponsorModel): Unit
+ getSponsor(sponsorId: String) : SponsorModel
+ getSponsors(sponsorIds: Seq[String]) : Seq[SponsorModel]
+ updateSponsor(sponsorId: String, sponsorModel: SponsorModel) : Unit
+ deleteSponsor(sponsorId: String) : Unit
pub SponsorModel
sponsorId: String,
sponsorGeographics: SponsorGeographics,
sponsorDemographics: SponsorDemographics,
sponsorMarketingCampaign: Option[SponsorMarketingCampaign],
lastEventBenefits: Option[Seq[AssetModel]],
pub SponsorGeographics
companyName: String,
status: CompanyStatus,
address: AddressInfo,
contactInfo: ContactInfo,
companyInfo: CompanyInfo,
pub AddressInfo
addressLines: Seq[String],
city: String,
zip: String,
province: String,
country: String,
pub ContactInfo
mainContactName: String,
title: String,
cellPhone: String,
mainLinePhone: String,
websiteUrl: String,
pub CompanyInfo
addresses: Seq[AddressInfo]
pub SponsorMarketingCampaign
marketingObjectDescription: Option[String],
targetAudienceInfo: Option[TargetAudienceInfo],
campaignRuntime: Option[CampaignRuntime],
tagLine: Option[String],
pub TargetAudienceInfo
primaryTarget: Option[String],
secondaryTarget: Option[String],
tertiaryTarget: Option[String]
pub SponsorDemographics
yearEstablished: Option[Int],
numberOfEmployees: Option[Int],
internalMarketingTeam: Boolean,
marketingAgency: Boolean,
industry: Option[String],
businessToBusiness: Boolean,
businessToCustomer: Boolean,
productsSoldDescription: Option[String],
servicePromotedDescription: Option[String],
logoFileUrl: Option[String],
AssetManager
+ mySqlClient: MySQLClient
+ createAsset(assetModel: AssetModel): Unit
+ getAsset(assetId: String) : AssetModel
+ getAssets(assetIds: Seq[String]) : Seq[AssetModel]
+ updateAsset(assetId: String, assetModel: AssetModel) : Unit
+ deleteAsset(assetId: String) : Unit
pub AssetModel
+ description : String
+ assetType: AssetType
+ assetId : String
<<enumeration>> AssetType
Car
Banner
Stand
...

The architecture for the backend can be laid out as this UML Class diagram. The structure is best described as somewhat of a tree, with Eight18Application creating a number of managers that all deal with their own smaller resources.

Link to diagram for closer inspection:

https://lucid.app/lucidchart/2f4fa569-79ee-44b3-9681-0f16e6f156b1/edit?view_items=7Yr-Dsnw-We0&invitationId=inv_72d26291-b746-481f-bb24-54f444f605db

Wireframes

Figure 8

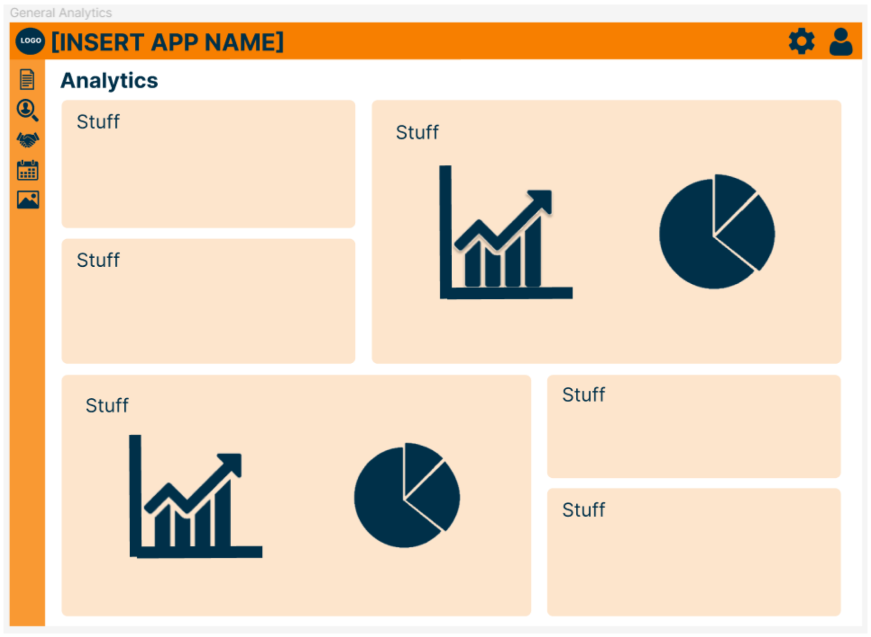
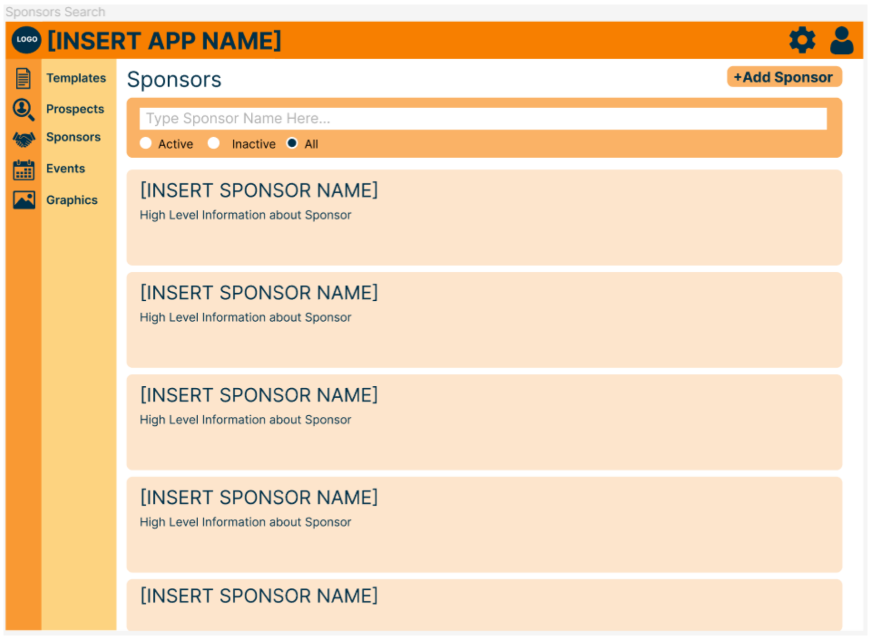


Figure 8 will be our main analytics page layout, where Eight:18 can see some general insights into current sponsors and events.

Figure 9



Active Inactive [✓ All]

[INSERT SPONSOR NAME]
High Level Information about Sponsor

[INSERT SPONSOR NAME]
High Level Information about Sponsor

[INSERT SPONSOR NAME]
High Level Information about Sponsor

[INSERT SPONSOR NAME]
High Level Information about Sponsor

[INSERT SPONSOR NAME]

Figure 9 will be our main sponsors list page, where Eight:18 will be able to search and inspect specific sponsors, or view the group as a whole.

Figure 10

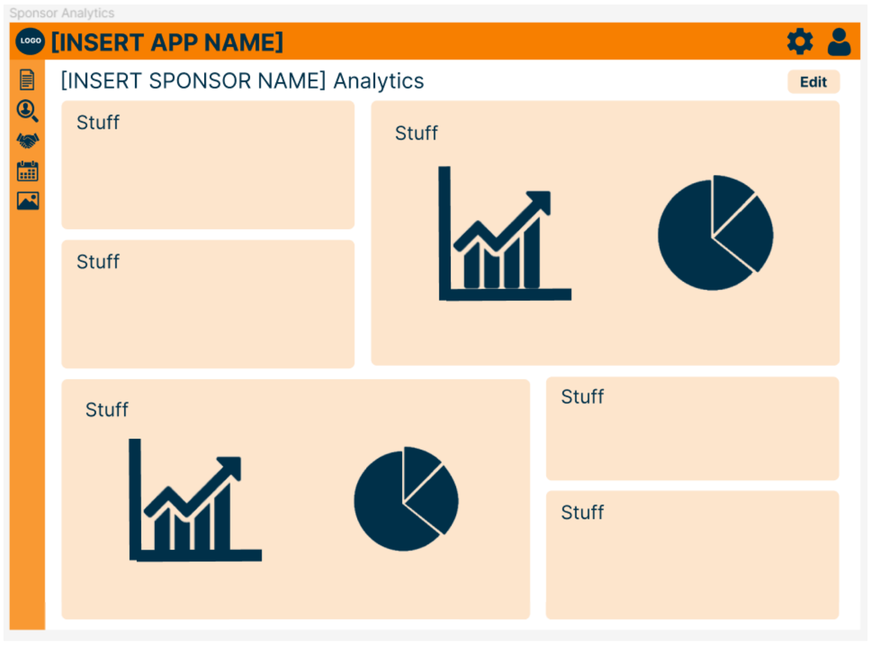


Figure 10 will be our main sponsor analytics page, where Eight:18 will be able to view general analytics and performance of this sponsor.

Figure 11

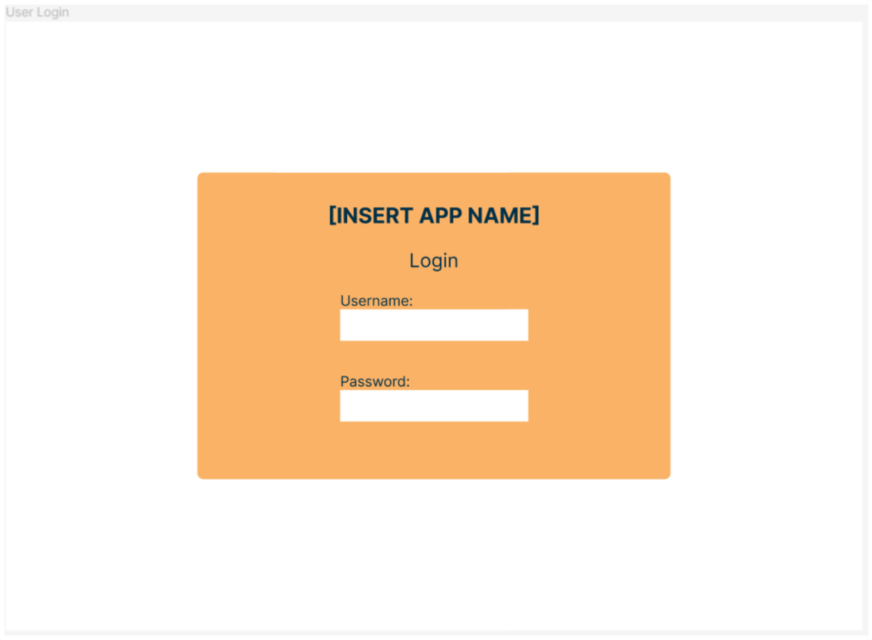


Figure 11 will be our main login page, where Eight:18 will be able to login to view sponsor data and analytics.

Figure 12

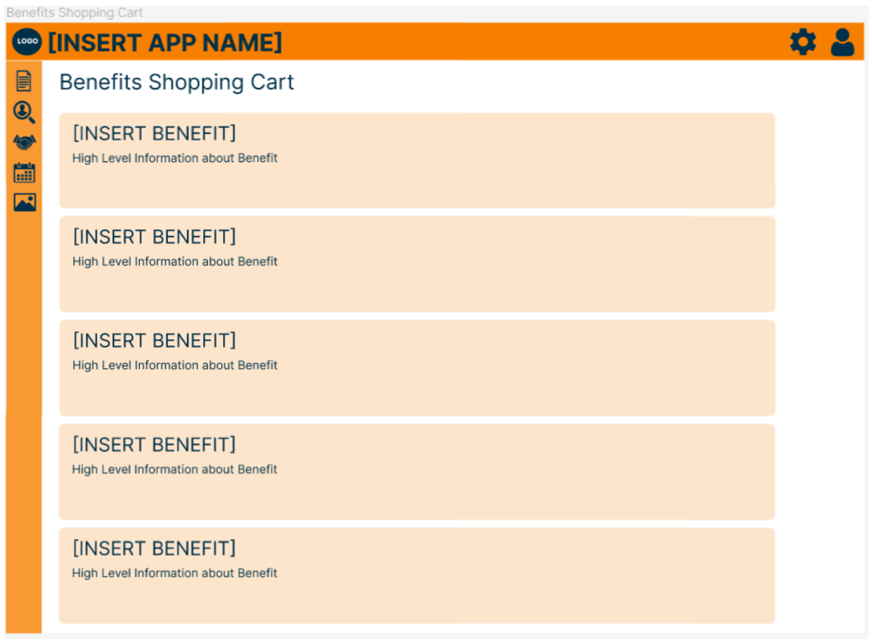

Benefits Shopping Cart
LOGO
[INSERT APP NAME]
Benefits Shopping Cart
[INSERT BENEFIT]
High Level Information about Benefit
[INSERT BENEFIT]
High Level Information about Benefit
[INSERT BENEFIT]
High Level Information about Benefit
[INSERT BENEFIT]
High Level Information about Benefit
[INSERT BENEFIT]
High Level Information about Benefit

Figure 12 will be the area that Eight:18 will be able to add benefits for sponsors on events.

Trade-Off Discussion

During the designing of this project, we had to make many decisions regarding implementation details and had to decide what tools would work better over other tools. This led us to comparing many different products and languages, here are the main trade-offs that we had to make:

1. **Scala vs. Java**

Every backend needs a programming language. During initial discussions we talked about a number of different programming languages that we could use to write our backend. Our discussions mostly bounced between Scala and Java.

Scala

Scala is a strongly typed general purpose programming language built on the JVM. It was designed by Martin Odersky as an alternative to Java, with one of its biggest features being language interoperability with Java, which allows us to use Java libraries in Scala code. Scala is object-oriented, like Java, but supports many more functional programming paradigms than Java does. These include, immutability, lazy evaluation, currying and pattern matching. On top of this Scala has one of the most intricate type systems around, meaning it is strongly typed. This allows for algebraic data types (great for revenue tracking), operator overloading, higher-order types and optional parameters. All of these points make Scala an attractive language to use in a backend environment, because immutability allows for multithreaded applications, Java has some amazing libraries (like Dropwizard mentioned earlier), and a strong type system is great for ensuring bugs are caught at compile time, not runtime. (Scala, 2022)

Java

Java is a high-level, object-oriented programming language built on the JVM. It was designed by James Gosling at Sun Microsystems in 1995. Java has similar syntax to C and C++, which made it attractive to developers in the 2000's, but now it is known for being verbose and requires a lot of boilerplate code before anything gets done. Java was designed with the "Write Once, Run Anywhere" philosophy, so code can be written on a Windows machine, and run on a Linux machine later. Java has a complicated type system, and its implementation of generics leaves a lot to be desired. Java is ~20% slower than Scala, and a history of security vulnerabilities make Java a hard sell for a backend that's written from scratch. (Java, 2022)

Language of Choice

In the end, we chose Scala. Scala makes many improvements on Java, including an 'Option' type (to avoid the dreaded NullPointerException), speed increases, and a host of other improvements while also allowing for use of amazing Java libraries. Scala is an improved version of Java, and we will be using it to write a great backend for Eight:18.

2. AWS vs. Microsoft Azure vs. Google Cloud Services

At the very beginning of this project it became clear that we would need a platform to deploy the web application for Eight:18 because eventually in later phases, they will be allowing users into the system and will need something to be running all the time. However, they are not interested in buying or maintaining their own server, as that has major setup and maintenance costs. We researched three different major services offered to see what platform would be the best to deploy on.

AWS

Amazon Web Services is currently the leader in cloud platforms, with around 200 plus services offered (Rana, 2022). They also currently have the largest number of availability zones at 66 (Rana, 2022), and are still expanding. While Eight:18's target consumers are currently all within the western United States, this leaves room for an expansion of clientele. Specifically, we would be using AWS S3 (Simple Storage Service) along with EC2 (Elastic Compute Cloud) to deploy our database and our frontend on. These services are easy to set up and use and are widely used, making AWS an attractive solution.

Microsoft Azure

Microsoft's Azure offers similar services to AWS, however at a significantly reduced amount, which is about 100 plus services (Rana, 2022). For the purposes of the project however, they do offer the same services as AWS. We would be using their Azure Virtual Machines and Azure Blob Storage for our web application and storage. It is worth mentioning that Azure only has 54 availability zones worldwide. They do still cover the entire United States, so this would not be a major issue.

Google Cloud

Google Cloud is the youngest platform out of these three, with only 20 availability zones throughout the world (Rana, 2022), and only about 60 services. Specifically, we would be using the Google Compute Engine and Google Cloud Storage services for storage and deployment. Google is currently expanding all of their services to catch up to Amazon and Microsoft.

Platform of Choice

We chose to go with AWS for a number of reasons. Firstly, they are known for being the most reliable platform of the three, due to the time that they have had to mature as a platform and the experience the company has gained as a result. Secondly, it has a much larger user base than Azure or Google cloud, which helps to achieve our non-functional goal of making this app easy to start development on again because Eight:18 will be more likely to find developers who have knowledge of AWS over Google Cloud or Azure. Lastly, AWS offers more services to meet specific needs, so if at any point in the project we need a new and niche service, there is a better chance that AWS will offer that solution over Google Cloud or Azure.

3. MySQL vs. Other Databases

Since we will be storing data about sponsors and assets associated with those sponsors for Eight:18, we will need to manage all their data and manage it well. There were a number of ways that we could have stored data. We researched MySQL and MongoDB.

MySQL

MySQL is a relational database system owned by Oracle. It is one of the most popular systems for managing relational databases, and has a host of libraries that allow it to be used by our backend using JDBC (Java Database Controller). It is free to install, with simple syntax and can be hosted in Amazon web services, which makes it extremely attractive for our purposes. Downsides of MySQL include scalability, limited open-source-ability and limited compliance with SQL standards. None of the downsides are really relevant to us at this moment, so MySQL makes a lot of sense for our purposes. (Altexsoft, 2019)

MongoDB

Another attractive solution that we researched was MongoDB. MongoDB is a free and open source non-relational database management system. Mongo relies heavily on RAM to get and store data quickly. Since Mongo isn't explicitly intended for structured data processing, it supports storing just about anything in the database. One of the benefits of MongoDB is horizontal scalability, because data is stored across a number of distributed servers. Another benefit is simple data storage and retrieval, because of its reliance on system memory. Some downsides of MongoDB include data insecurity and extensive memory consumption. (Altexsoft, 2019)

Database of choice

Ultimately we landed on MySQL for our "cold" data store. We are familiar with SQL syntax from previous classes and work, which makes this an easy pick up and setup for us. Also, MongoDB doesn't have a focus on security (user login is not required), and MongoDB has no guarantees that their updates won't break your existing implementation. For these reasons, we will be using MySQL as our data store for Eight:18.

4. **Redis vs. Other In-Memory Datastores**

We also need a form of "hot" storage, where we will be storing smaller packets of information, like API credentials, login session information, and other bits of data that don't require a full fledged database. We examined a number of in-memory data stores but the two that were most attractive to us were Redis and Memcached.

Memcached

Memcached is a high-performance distributed memory object caching system. It is generic in nature, allowing us to store data in key-value pairs (memcached). For our purposes, we would use it to store a JSON object in memory and then retrieve it from Memcached when we need it. Advantages to using Memcached are blazing speed, simplicity and efficient use of memory. Disadvantages of Memcached include a lack of advanced data structures, no transactions and no replicas of data (Redis vs. Memcached).

Redis

Redis is an open source, in-memory, key-value data store that can be used as a primary database, cache, message broker, and queue. Redis is extremely fast as data is stored as in-memory, and can be used to store "hot" data that will be accessed often. Benefits of using Redis include advanced data structures, modules that can be installed alongside redis for support of custom data structures (like JSON, XML and others), and high scalability because of data replication (Redis : In Memory Database). One of the downsides of Redis include its reliance on system memory. If our data doesn't all fit in memory, Redis won't work, or at least, will be very slow. Another downside of Redis is its lack-luster security. Redis doesn't encrypt data at rest, and so we need to do that ourselves for sensitive data.

In-memory data store of choice

After a bit of deliberation, Redis was included to act as our "hot" data store. Redis just has way more features and use cases than Memcached, and is supported by AWS pretty much out of the box, which makes it an easy choice for our use case.

5. **Agile vs. Waterfall**

To assist with organization we had to decide what our design life cycle was going to be. There are a myriad of options that we could have gone with but narrowed it down to the Agile approach and the Waterfall approach. We researched what each method was about and then made a decision that best fit our needs.

Agile

Agile is an iterative approach in which a team breaks up a project into its features which the team tackles during time periods called “sprints” (Kavlakoglu, 2020). During a sprint, the team develops a portion of the software and at the end, will reconvene to evaluate progress, and possibly demo the current state of the product to the stakeholder. Any feedback is then used in the next sprint to change the direction of the project. This method is very flexible and allows for modification of the road map for a more customized product that truly fits the stakeholders needs.

Waterfall

The Waterfall method is a linear sequential method where everything is generally planned and tracked from the start (Kavlakoglu, 2020). This tracking is usually done through a Gant chart and the entire method can be roughly broken up into the following phases: gathering of requirements, design, implementation, verification, and maintenance. Any involvement from the stakeholder is usually at the very end, during verification and maintenance which may be costly if some requirement is misinterpreted.

Method of Choice

The Agile approach better fits our needs for this project. We have Eight:18 heavily involved in this process, so having a system where we can show them progress consistently works better. This allows for that constant feedback to always be developing the app towards the vision of Eight:18. We also liked having the ability to be flexible in our development, which is at the core of the Agile philosophy.

6. Angular vs. React JS vs. Vue JS

In the modern era of web development, web frameworks have become increasingly popular. Considering all of the benefits they provide, it is no surprise that they have become the industry standard. The question was not whether we should leverage a framework's power, but which one we should choose to utilize. We quickly narrowed our options down to three of the most popular frameworks that exist: Angular, React JS, and Vue JS.

Angular

The Angular web framework is a Google product specifically designed for large web applications. From an architectural standpoint, it is the most rigid of the three frameworks that we are analyzing, with strict requirements for component design and navigation. Due to this rigidity, Angular is primarily intended to develop large, architecturally complex applications that must have lots of room for growth. Angular also plays host to the largest core functionality out of the three frameworks, this results in less need for external libraries, longer build times and a much steeper learning curve.

React JS

React JS, developed by Facebook, is largely considered to be the most popular web framework available to use. It is much less strict in its architecture, leading to more flexibility in component design, and it hosts a much smaller core library than Angular does. This smaller core library means heighted need for external packages, fast build times, and a much smaller learning curve. React JS also has the most flexibility in terms of application size. The small learning curve and less strict architectural requirements make it much more suited to developing smaller applications, while still offering room for growth.

Vue JS

Vue JS, developed by Evan You, offers the most unique layout of the three frameworks considered. It is primarily driven by a Model-View-Viewmodel, MVVM, design. The model represents the information, the view is what the user sees, and the viewmodel is the tool to translate between the view and the model. Overall, this is fairly similar to our top level architecture but restricted to only the front end. This design choice, that fully separates each aspect of GUI development, results in a slightly steeper learning curve than React JS and less flexibility for application size. The MVVM architecture is considered too complicated for a simple application, but grows in complexity substantially once the application has become too large.

Framework of Choice

Considering the low learning curve, flexibility of design, and potential for growth, the React JS framework became the clear choice. Considering we are beginners, the easy learning curve is desirable, the flexibility will allow us to be able to rethink our design later on if we need to, and the potential for growth will mean that even after we have finished development for the semester, the application still has plenty of room left to expand to what Eight:18 ultimately desires.

VII. Expected Results

Upon completion of the Eight:18 capstone project, we anticipate completing three primary objectives. The first objective is to move organization of data into databases, and make that data accessible. This data can be queried by sponsor, and will be delivered via JSON to the frontend. Second, this data needs to be displayed on an intuitive and attractive front end. The data will be broken up into widgets to better visually organize the data, and this should allow for much better access to a sponsor and their information. Finally, the app will be capable of adding new sponsors and editing pre-existing ones. If a sponsor is no longer with Eight:18, they need to be able to archive or remove them from the system.

VIII. Actual Results

Actual Screenshots

Figure 13

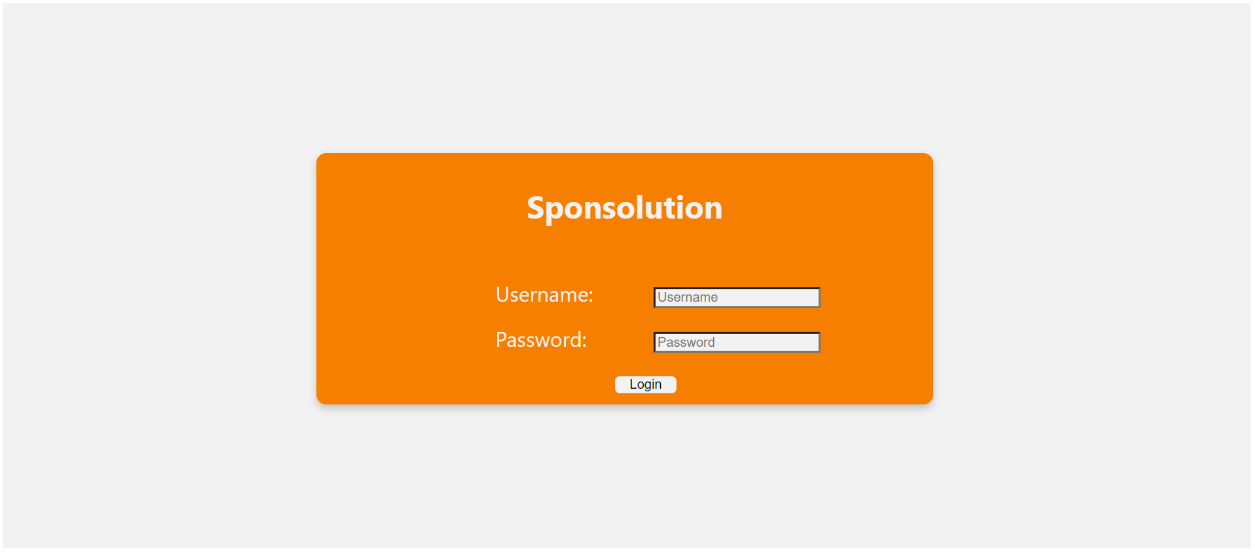
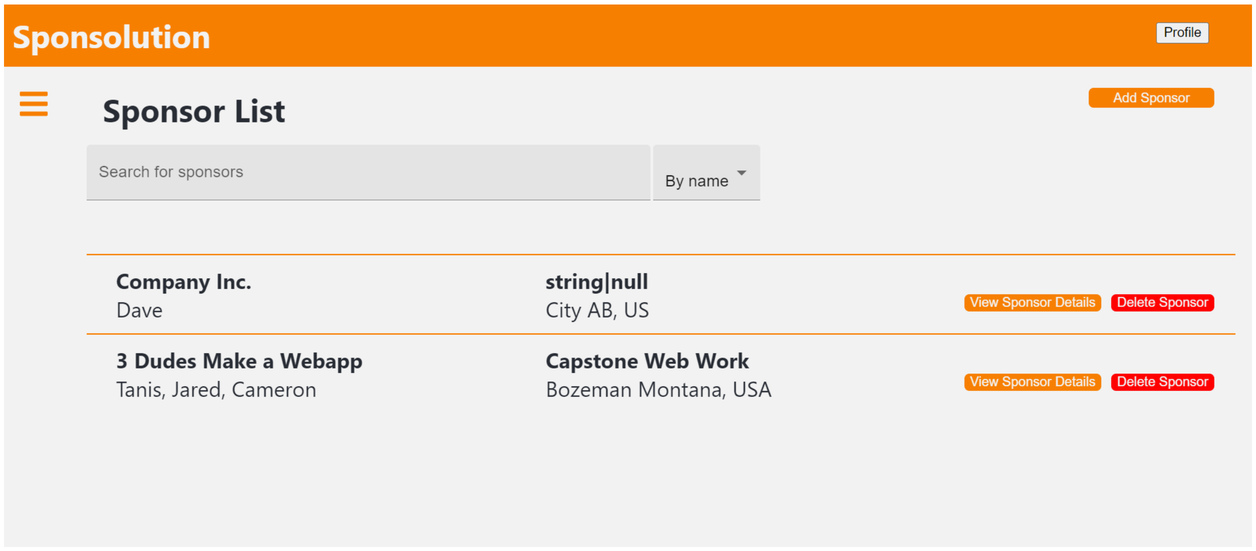


Figure 13 is the main login page, where Eight:18 will be able to login to view sponsor data and analytics.

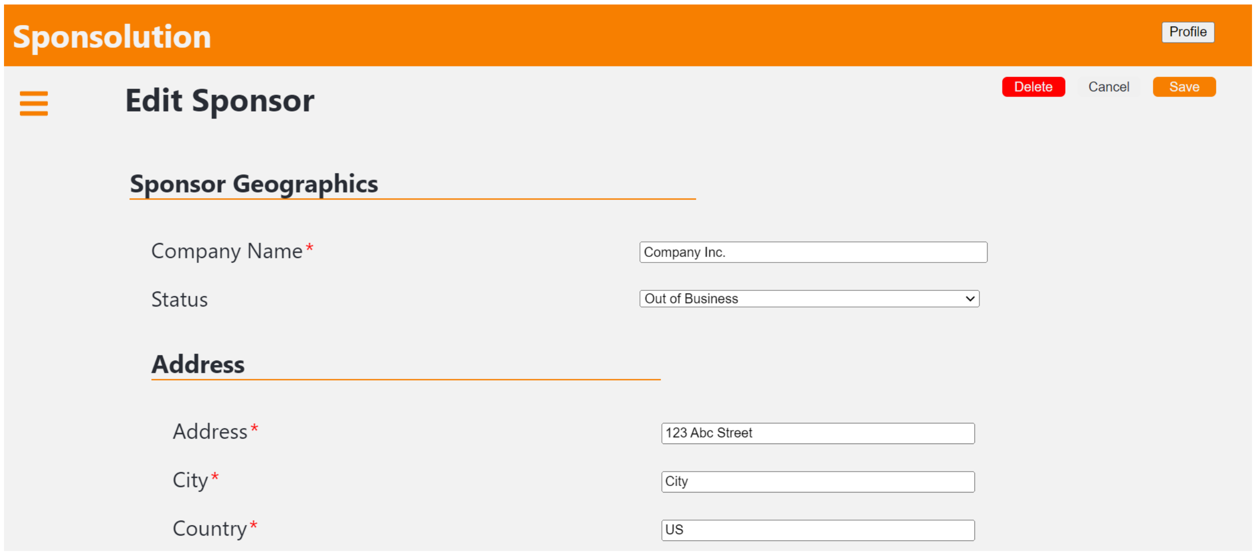
Figure 14



Company Inc. Dave	**string|null** City AB, US
3 Dudes Make a Webapp Tanis, Jared, Cameron	**Capstone Web Work** Bozeman Montana, USA

Figure 14 is the main sponsors list page, where Eight:18 will be able to search and inspect specific sponsors, or view the group as a whole.

Figure 15



Edit Sponsor

Sponsor Geographics

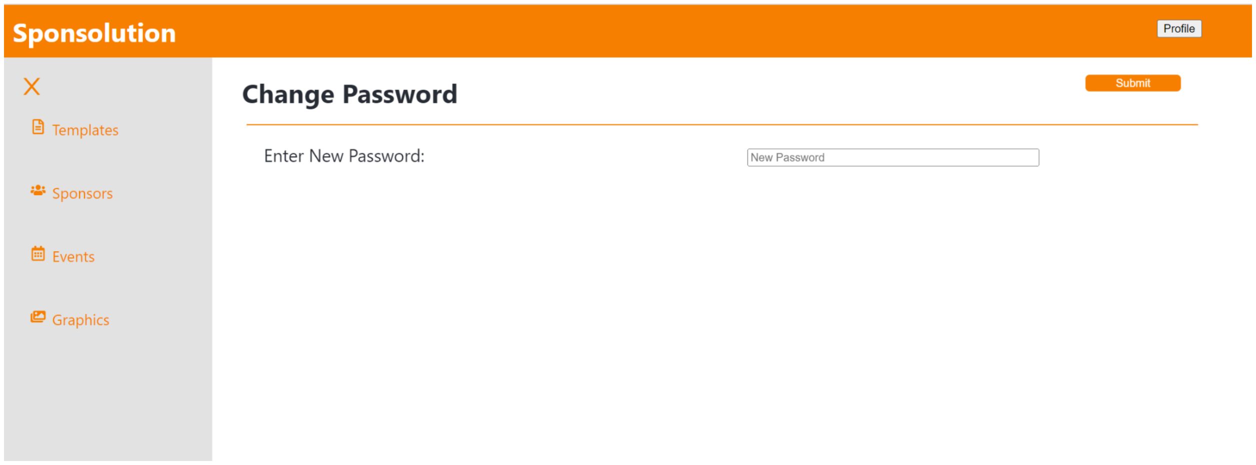
Company Name*	Company Inc.
Status	Out of Business

Address

Address*	123 Abc Street
City*	City
Country*	US

Figure 15 is the main sponsor edit page. This can also function as a create sponsor page. This form, which is much longer than what can fit in the screen, contains every valuable field a sponsor could have.

Figure 16

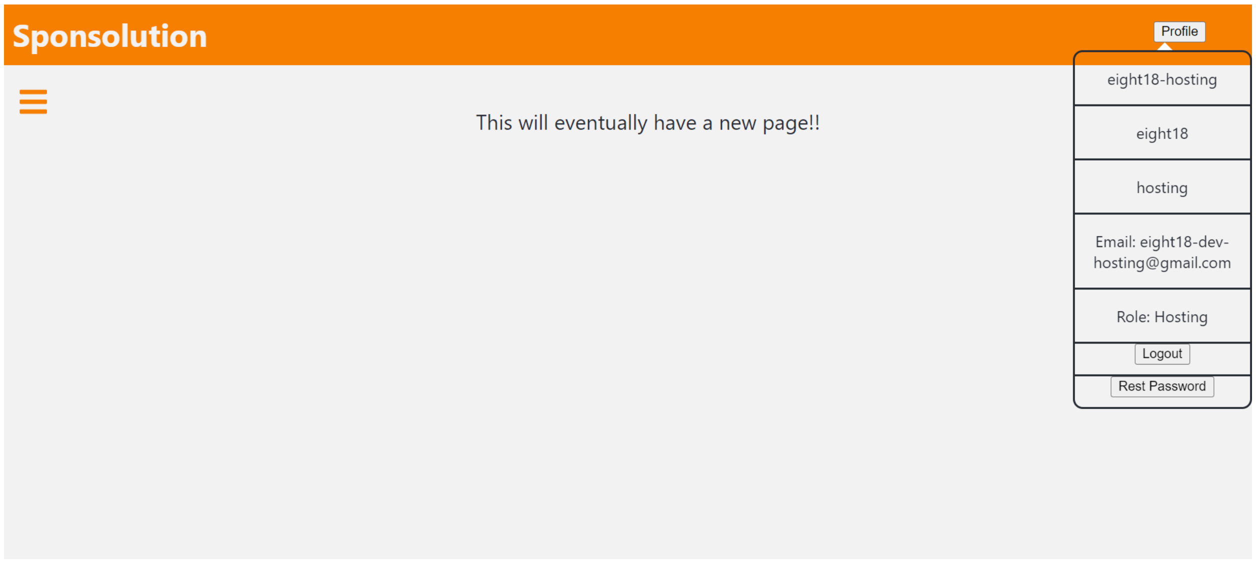


Change Password

Enter New Password: New Password

Figure 16 is the reset password page, which is accessible through the profile menu. Additionally, this frame shows the expanded version of the side navigation bar.

Figure 17



This will eventually have a new page!!

Figure 17 shows the dummy page that currently functions as a stand in for the page components that don't exist yet. It also displays the expanded profile menu.

We believe that we have fulfilled the expected results for this capstone project. The web application has the ability to track and edit sponsors, and storage is now digital, sitting in a MySQL database for cold storage. The application also has an interface that is consistent with what is common practice for today's applications. The interface is also very clean and attractive, fulfilling that non-functional requirement as well. While we unfortunately could not get around to phases two and three, we believe that we have established a very solid prototype for a product that Eight:18 may like to use in the future.

Code

This final section shows an example of the adapter pattern and how we built our abstraction of Eight:18's data in a MySQL database. Here, we are adapting data given from forms/raw json data, and storing it in a MySQL database, and also fetching that data out in ways that we know how to deal with and understand on the backend.

```
package eight18.core.database

import eight18.core.json.{JsonMappers, Jsonable}
import eight18.core.logging.Logging
import eight18.managers.SponsorManager.{SponsorDemographics, SponsorGeographics,
SponsorMarketingCampaign, SponsorModel, SponsorReference}
import eight18.{LoginInformation, ServerInformation}

import scala.collection.mutable
import scalikejdbc.*
import com.fasterxml.jackson.databind.JsonNode
import eight18.api.CoreResource.SetMySQLCredsRequest
import eight18.core.data.encryption.InlineEncryption.DecryptionException
import eight18.core.data.encryption.{EncryptedStringDeserializer, EncryptedStringSerializer,
InlineEncryption, KeyResolver}
import eight18.core.data.strings.StringUtils
import eight18.core.redis.JedisClient
import eight18.managers.AssetManager.AssetModel
import eight18.managers.assets.AssetType
import eight18.managers.UserManager.{User, UserRole}
import io.dropwizard.lifecycle.Managed
import eight18.managers.EventManager.*
import eight18.managers.UserManager

import scala.reflect.ClassTag
import scala.util.{Failure, Success, Try}

object MySQLClient {
  private class MySqlError(message: String) extends RuntimeException(message)

  object MySqlErrors {
    def MISSING_FIELD(field: String, db: String) = throw new MySqlError(s"No such field:
$field in MySQL database $db")
  }
}

class MySQLClient(serverInformation: ServerInformation, jedisClient: JedisClient) extends
Logging with Managed {
  import MySQLClient._
  import MySQLClient.MySqlErrors._

  /*
  * As of 1/16/23 we have support for connecting to an AWS RDC MySQL instance from the
backend
  * Got this working, but don't feel like paying Amazon a buttload of money for storing my
bogus sponsor
```

```
  * data.
  * */

  private var shouldLogSuccessfulQueries : Boolean = false
  def toggleQueryLogging(): Unit = {
    shouldLogSuccessfulQueries = !shouldLogSuccessfulQueries
  }

  GlobalSettings.queryCompletionListener = (sql: String, params: collection.Seq[Any], millis:
Long) => {
    if (shouldLogSuccessfulQueries) {
      logger.info(s"Statement: [$sql] params: [$params] executed successfully in ${millis}ms")
    } else {
      logger.trace(s"Statement: [$sql] params: [$params] executed successfully in ${millis}ms")
    }
  }

  GlobalSettings.queryFailureListener = (sql: String, params: collection.Seq[Any], e:
Throwable) => {
    logger.error(s"Statement: [$sql] params: [$params] failed to execute", e)
  }

  // database credentials
  private final val dbLoginInfoKey: String = Seq("eight18", "database", "login").mkString(":")
  private var loginInformation: LoginInformation = try {
    jedisClient.execute { jedis =>
      JsonMappers.encryptedMapper.readValue(jedis.get(dbLoginInfoKey),
classOf[LoginInformation])
    }
  } catch {
    case d: DecryptionException if !StringUtils.isNullOrEmpty(d.unencryptedString) =>
      logger.warn(s"Found unencrypted value for DB login info, setting it again so we don't
encounter this in the future")
      jedisClient.execute { jedis =>
        val unencrypted = Option(jedis.get(dbLoginInfoKey))
          .map(json => JsonMappers.coreMapper.readValue(json, classOf[LoginInformation]))
          .getOrElse(throw new RuntimeException(s"Expected that dbLoginInfo key was set
[$dbLoginInfoKey]"))
        val encrypted = JsonMappers.encryptedMapper.writeValueAsString(unencrypted)
        jedis.set(dbLoginInfoKey, encrypted)
        unencrypted
      }
    case t: Throwable =>
      logger.error(s"Unexpected throwable during get DB credentials", t)
      null
  }
  if (loginInformation == null) {
    logger.error("-" * 100 + "*" * 10 + "-" * 100)
    logger.error(s"Tried to get MySQL login credentials from redis on service startup but
credentials were not in redis. They are blank for now.")
    logger.error(s"SET THE CREDENTIALS AND THEN RESTART THE BACKEND,
MySQL INTEGRATION WILL NOT WORK WITHOUT CREDENTIALS SET")
    logger.error("-" * 100 + "*" * 10 + "-" * 100, new Exception("here"))

    loginInformation = LoginInformation("root", "eight184ever") // this is what the login info to
your local database should be, but this won't work when our DB is hosted in AWS
  }

  private val awsMySql: Boolean = serverInformation.host.endsWith("amazonaws.com")

  private val prefix: String = if (awsMySql) "jdbc:mysql:aws" else "jdbc:mysql"
  private val databaseName: String = "mysql"
  private val url : String =
s"$prefix://${serverInformation.host}:${serverInformation.port}/$databaseName"

  logger.debug(s"Credentials: $loginInformation")
  private val username: String = loginInformation.username
  private val password: String = loginInformation.password

  // database connection
  if (awsMySql)
    Class.forName("software.aws.rds.jdbc.mysql.Driver")
  else
    Class.forName("org.h2.Driver")
  ConnectionPool.singleton(url, username, password)
  implicit val session: AutoSession = AutoSession

  logger.debug(s"Connected to MySQL database on host: [${serverInformation.host}] port:
[${serverInformation.port}]") // in theory, seems this log isn't always correct >.>

  private val internalIdField = sqls"`internal_id`"

  // REQUIRED TO INITIALIZE ALL OBJECTS AND PERFORM NECESSARY TABLE
UPDATES
  // PLEASE DO NOT DELETE
  // Also, if you add more objects under mysqlclient, make sure to stick them here so your
tables pop up :)
  try {
    events.init()
    assets.init()
    sponsors.init()
    users.init()
  } catch {
    case t: Throwable =>
      logger.error("-" * 100 + "*" * 10 + "-" * 100)
      logger.error(s"ERROR INITIALIZING DATABASE TABLES, GUESSING THIS HAS
SOMETHING TO DO WITH LOGIN CREDENTIALS, CATCHING THIS SO YOU CAN
FIX, THE BACKEND IS BROKEN RIGHT NOW")
      logger.error("-" * 100 + "*" * 10 + "-" * 100, t)
  }


  /**
   * Actions to be run when MySQLClient starts and stops,
   * these are executed by the dropwizard managed lifecycle
   * */
  override def start(): Unit = {
    super.start()
  }
  override def stop(): Unit = {
    super.stop()
  }

  def setMySQLCredentials(request: SetMySQLCredsRequest): Unit = {
    jedisClient.execute { jedis =>
      val payload = LoginInformation(request.username,
request.password).toJsonEncryptedString
      logger.debug(s"Setting database credentials: [$payload]")
      jedis.set(dbLoginInfoKey, payload)
    }
  }

  /**
   * Get a field as a primitive scala type, no object casting here as
   * we are writing jsons into the db, suggest using `getObject` for that use case
   * */
  private def getField[T](rawMap: Map[String, Any], field: String, db: String) : T = {
    rawMap.getOrElse(field, MISSING_FIELD(field, db)).asInstanceOf[T]
  }

  /**
   * Get an object out of a json in the db, here we do some fancy casting to just
   * to make life easier, as opposed to doing .asInstanceOf[T] all the time
   * if you just want a string or something like that, use `getField`
   * */
  private def getObject[T: ClassTag](rawMap: Map[String, Any], field: String, db: String,
isOptional: Boolean = false)(implicit tag: ClassTag[T]) : T = {
    val rawJson = getField[String](rawMap, field, db)
    // just did try/catch here to get more readable error messages, we still just
    // rethrow the exception anyway
    try {
      JsonMappers.coreMapper.readValue(rawJson, tag.runtimeClass).asInstanceOf[T]
    } catch {
      case t: Throwable =>
        logger.error(s"Unable to cast [$rawJson] to class ${tag.runtimeClass}, rethrowing
exception", t)
        throw t
    }
  }

  object events {
    /**
     * Event related SQL, if you end up making changes to
     * DB table, make sure to update the table definition in code
     * TAG: EVENT_TABLE
     * */
    private val eventsSqlTable = sqls"`eight18`.`events`"
    private val eventIdField = sqls"`event_id`"
    private val eventDataField = sqls"`event_data`"
    private val sponsorReferenceField = sqls"`sponsor_reference`"
    sql"""
      CREATE TABLE IF NOT EXISTS $eventsSqlTable (
        $internalIdField INT NOT NULL AUTO_INCREMENT,
        $eventIdField VARCHAR(60) NOT NULL,
        $eventDataField JSON NOT NULL,
        $sponsorReferenceField JSON NOT NULL,
      PRIMARY KEY ($internalIdField, $eventIdField));
    """.execute.apply()

    def init(): Unit = {}

    def insertEventData(event: EventModel) : Unit = {
      val eventDataJson = JsonMappers.coreMapper.writeValueAsString(event.eventData)
      val sponsorReferences = JsonMappers.coreMapper.writeValueAsString(event.sponsors)

      sql"""
      INSERT INTO $eventsSqlTable (`event_id`, `event_data`, `sponsor_reference`)
      VALUES (${event.eventId}, $eventDataJson, $sponsorReferences)
      """.update.apply()
    }

    def getEvents(eventIds: Seq[String]) : Seq[EventModel] = {
```

```
    val rawMaps: Seq[Map[String, Any]] = sql"""
    SELECT event_id, event_data, sponsor_reference
    FROM $eventsSqlTable
    WHERE event_id IN ($eventIds)
    """.map(_.toMap()).list.apply()

    rawMaps.map(eventModelRaw => {
      val eventId = getField[String](eventModelRaw, "event_id", "events")
      val eventData = getObject[EventData](eventModelRaw, "event_data", "events")
      val sponsorReference = getObject[Seq[SponsorReference]](eventModelRaw,
"sponsor_reference", "events")

      EventModel(eventData, sponsorReference, eventId)
    })
  }

  def updateEvent(newEventModel: EventModel) : Unit = {
    val eventDataJsonString =
JsonMappers.coreMapper.writeValueAsString(newEventModel.eventData)
    val sponsorReferenceJsonString =
JsonMappers.coreMapper.writeValueAsString(newEventModel.sponsors)
    val eventId = newEventModel.eventId

    sql"""
    UPDATE $eventsSqlTable
    SET event_data=$eventDataJsonString, sponsor_reference=$sponsorReferenceJsonString
    WHERE event_id=$eventId
    """.update.apply()
  }

  def deleteEvent(eventId: String) : Unit = {
    sql"""
    DELETE FROM $eventsSqlTable WHERE event_id=$eventId
    """.execute.apply()
  }

  def dangerOnlyUseWhenYouKnowWhatYouAreDoingEvents(): Unit = {
    sql"""
      TRUNCATE $eventsSqlTable
    """.execute.apply()
  }
 }

 object sponsors {
  /**
   * Sponsor related SQL, if you end up making changes to
```

```
    * DB table, make sure to update the table definition in code
    * TAG: SPONSOR_TABLE
    * */
   private val sponsorsSqlTable = sqls"eight18.sponsors"
   private val sponsorGeographicsField = sqls"`sponsor_geographics`"
   private val sponsorDemographicsField = sqls"`sponsor_demographics`"
   private val sponsorMarketingCampaignField = sqls"`sponsor_marketing_campaign`"
   private val lastEventBenefitsField = sqls"`last_event_benefits`"
   private val sponsorIdField = sqls"`sponsor_id`"
   private val allSponsorFields = sqls"$sponsorGeographicsField,
$sponsorDemographicsField, $sponsorMarketingCampaignField, $lastEventBenefitsField,
$sponsorIdField"

   sql"""
     CREATE TABLE IF NOT EXISTS $sponsorsSqlTable (
       $internalIdField INT UNSIGNED NOT NULL AUTO_INCREMENT,
       $sponsorGeographicsField JSON NOT NULL,
       $sponsorDemographicsField JSON NOT NULL,
       $sponsorMarketingCampaignField JSON NULL,
       $lastEventBenefitsField JSON NULL,
       $sponsorIdField VARCHAR(60) NOT NULL,
     PRIMARY KEY ($internalIdField, $sponsorIdField),
     UNIQUE INDEX `internal_id_UNIQUE` ($internalIdField ASC) VISIBLE);
   """.execute.apply()

   def init(): Unit = {}

   def insertSponsorModel(sponsor: SponsorModel) : Boolean = {
     val sponsorGeographics = sponsor.sponsorGeographics.toJsonString
     val sponsorDemographics = sponsor.sponsorDemographics.toJsonString
     val sponsorMarketingCampaign =
sponsor.sponsorMarketingCampaign.map(_.toJsonString).orNull
     val lastSponsorBenefits =
JsonMappers.coreMapper.writeValueAsString(sponsor.lastEventBenefits.orNull)

     sql"""
     INSERT INTO $sponsorsSqlTable ($allSponsorFields)
     VALUES ($sponsorGeographics, $sponsorDemographics, $sponsorMarketingCampaign,
$lastSponsorBenefits, ${sponsor.sponsorId})
     """.execute.apply()
   }

   def getSponsors(sponsorIds : Seq[String]) : Seq[SponsorModel] = {

     val rawMaps: Seq[Map[String, Any]] = sql"""
     SELECT $allSponsorFields
    FROM $sponsorsSqlTable
    WHERE $sponsorIdField IN ($sponsorIds)
    """.map(_.toMap()).list.apply()

    val results = rawMaps.map(sponsorModelRaw => {
      val model = SponsorModel(
        sponsorId = getField[String](sponsorModelRaw, "sponsor_id" ,"sponsors"),
        sponsorGeographics = getObject[SponsorGeographics](sponsorModelRaw,
"sponsor_geographics", "sponsors"),
        sponsorDemographics = getObject[SponsorDemographics](sponsorModelRaw,
"sponsor_demographics", "sponsors"),
        // NOTE: we do `Try` here since when the field is NULL it isn't included in the map, and
this field is optional so it is null-able
        sponsorMarketingCampaign =
Try(getObject[SponsorMarketingCampaign](sponsorModelRaw,
"sponsor_marketing_campaign", "sponsors")).toOption, brandAttitudes = None,
        lastEventBenefits = Option(getObject[Any](sponsorModelRaw, "last_event_benefits",
"sponsors"))
      )
      model
    })

    results
  }

  def updateSponsor(sponsorId: String, newSponsorModel: SponsorModel): Boolean = {
    val sponsorGeographics = newSponsorModel.sponsorGeographics.toJsonString
    val sponsorDemographics = newSponsorModel.sponsorDemographics.toJsonString
    val sponsorMarketingCampaign =
newSponsorModel.sponsorMarketingCampaign.map(_.toJsonString).orNull
    val lastSponsorBenefits =
JsonMappers.coreMapper.writeValueAsString(newSponsorModel.lastEventBenefits.orNull)

    sql"""
    UPDATE $sponsorsSqlTable
    SET $sponsorGeographicsField=$sponsorGeographics,
$sponsorDemographicsField=$sponsorDemographics,
$sponsorMarketingCampaignField=$sponsorMarketingCampaign,
$lastEventBenefitsField=$lastSponsorBenefits
    WHERE $sponsorIdField=$sponsorId
    """.execute.apply()
  }

  def deleteSponsor(sponsorId: String): Boolean = {
    sql"""
    DELETE FROM $sponsorsSqlTable WHERE $sponsorIdField=$sponsorId
    """.execute.apply()
  }

  def getAllSponsors: Seq[SponsorModel] = {
    val rawMaps: Seq[Map[String, Any]] =sql"""
      SELECT $allSponsorFields
      FROM $sponsorsSqlTable
    """.map(_.toMap()).list.apply()

    val results = rawMaps.map(sponsorModelRaw => {
      val model = SponsorModel(
        sponsorId = getField[String](sponsorModelRaw, "sponsor_id", "sponsors"),
        sponsorGeographics = getObject[SponsorGeographics](sponsorModelRaw,
"sponsor_geographics", "sponsors"),
        sponsorDemographics = getObject[SponsorDemographics](sponsorModelRaw,
"sponsor_demographics", "sponsors"),
        // NOTE: we do `Try` here since when the field is NULL it isn't included in the map, and
this field is optional so it is null-able
        sponsorMarketingCampaign =
Try(getObject[SponsorMarketingCampaign](sponsorModelRaw,
"sponsor_marketing_campaign", "sponsors")).toOption,
        brandAttitudes = None,
        lastEventBenefits = Option(getObject[Any](sponsorModelRaw, "last_event_benefits",
"sponsors"))
      )
      model
    })

    results
  }

  def getAllSponsorIds: Seq[String] = {
    val rawMaps: Seq[Map[String, Any]] =
    sql"""
      SELECT $sponsorIdField FROM $sponsorsSqlTable
    """.map(_.toMap()).list.apply()

    rawMaps.map(getField[String](_, "sponsor_id", "sponsors"))
  }

  def dangerOnlyUseWhenYouKnowWhatYouAreDoingSponsors(): Unit = {
    logger.warn("!!! WARNING !!! We are about to delete all sponsors from the DB, we really
hope you meant to do that!!!")
    sql"""
      DELETE FROM $sponsorsSqlTable;
    """.execute.apply()
  }
 }

 object users {
  /**
   * User related SQL, if you end up making changes to
   * DB table, make sure to update the table definition in code
   * TAG: USER_TABLE
   * */
  private val usersSqlTable = sqls"`eight18`.`users`"
  sql"""
    CREATE TABLE IF NOT EXISTS $usersSqlTable (
      `id` INT NOT NULL,
      `first_name` VARCHAR(45) NULL,
      `last_name` VARCHAR(45) NULL,
      `email` VARCHAR(100) NOT NULL,
      `password` VARCHAR(50) NOT NULL,
      `role` VARCHAR(7) NOT NULL,
      `locked` BIT(1) NOT NULL,
      `disabled` BIT(1) NOT NULL,
      `username` VARCHAR(45) NOT NULL,
    PRIMARY KEY (`id`, `email`, `username`),
    UNIQUE INDEX `id_UNIQUE` (`id` ASC) VISIBLE,
    UNIQUE INDEX `email_UNIQUE` (`email` ASC) VISIBLE,
    UNIQUE INDEX `username_UNIQUE` (`username` ASC) VISIBLE);
  """.execute.apply()

  private val firstName= sqls"`first_name`"
  private val lastName = sqls"`last_name`"
  private val email = sqls"`email`"
  private val password = sqls"`password`"
  private val role = sqls"`role`"
  private val locked = sqls"`locked`"
  private val disabled = sqls"`disabled`"
  private val username = sqls"`username`"

  def init(): Unit = {}

  private def getUserFromRawMap(rawMap: Map[String, Any]) : User = {
    val id = getField[Int](rawMap, "id", "users")
    val firstName = getField[String](rawMap, "first_name", "users")
    val lastName = getField[String](rawMap, "last_name", "users")
    val userName = getField[String](rawMap, "username", "users")
    val email = getField[String](rawMap, "email", "users")
    val password = EncryptedStringDeserializer.decrypt(getField[String](rawMap, "password",
"users"))
```

```
    val role : UserRole = UserManager.userRoleFromString(getField[String](rawMap, "role",
"users"))
    val locked = getField[Boolean](rawMap, "locked", "users")
    val disabled = getField[Boolean](rawMap, "disabled", "users")
    User(
      firstName = firstName,
      lastName = lastName,
      userName = userName,
      email = email,
      password = password,
      role = role,
      locked = locked,
      disabled = disabled,
      id = id
    )
  }

  def createUser(user: User): Unit = {
    val id = user.id
    val firstName = user.firstName
    val lastName = user.lastName
    val userName = user.userName
    val email = user.email
    val password = EncryptedStringSerializer.encrypt(user.password)
    val role = user.role.toString
    val locked = if (user.locked) 1 else 0
    val disabled = if (user.disabled) 1 else 0
    sql"""
      INSERT INTO $usersSqlTable (`id`, `first_name`, `last_name`, `email`, `password`,
`role`, `locked`, `disabled`, `username`)
      VALUES ($id, $firstName, $lastName, $email, $password, $role, $locked, $disabled,
$userName)
    """.update.apply()
  }

  def findUserByEmail(email: String): Option[User] = {
    val mapOpt = sql"""
      SELECT * FROM $usersSqlTable WHERE email = $email
    """.map(_.toMap()).list.apply().headOption
    val ret = mapOpt.map(getUserFromRawMap)
    ret
  }

  def findUserByUsername(userName: String) : Option[User] = {
    val mapOpt = sql"""
      SELECT * FROM `eight18`.`users` WHERE username = $userName
    """.map(_.toMap()).list.apply().headOption
    val ret = mapOpt.map(getUserFromRawMap)
    ret
  }

  def updateUser(user: User): Unit = {
    sql"""
    UPDATE $usersSqlTable
    SET $firstName=${user.firstName}, $lastName=${user.lastName}, $email=${user.email},
$password=${EncryptedStringSerializer.encrypt(user.password)}, $role=${user.role.toString},
$locked=${if(user.locked) 1 else 0}, $disabled=${if(user.disabled) 1 else 0}
    WHERE $username=${user.userName}
    """.update.apply()
  }

  def dangerOnlyUseWhenYouKnowWhatYouAreDoingUsers(): Unit = {
    sql"""
      DELETE FROM $usersSqlTable
      WHERE $username != ${"eight18-hosting"}
    """.execute.apply()
  }
 }

 object assets {
  private val assetsSqlTable = sqls"`eight18`.`assets`"
  private val descriptionField = sqls"`description`"
  private val assetTypeField = sqls"`asset_type`"
  private val assetIdField = sqls"`asset_id`"
  private val sponsorReferenceField = sqls"`sponsor_reference`"
  private val dollarAmountField = sqls"`dollar_amount`"

  private val allAssetFields = sqls"$descriptionField, $assetTypeField, $assetIdField,
$sponsorReferenceField, $dollarAmountField"

  sql"""
    CREATE TABLE IF NOT EXISTS $assetsSqlTable (
      $internalIdField INT NOT NULL AUTO_INCREMENT,
      $assetIdField VARCHAR(60) NOT NULL,
      $assetTypeField VARCHAR(60) NOT NULL,
      $sponsorReferenceField JSON NOT NULL,
      $dollarAmountField INT NOT NULL,
      $descriptionField VARCHAR(1000) NOT NULL,
    PRIMARY KEY ($internalIdField, $assetIdField));
  """.execute.apply()

  def init(): Unit = {}
  private def getAssetFromRawMap(rawMap: Map[String, Any]): AssetModel = {
    val description = getField[String](rawMap, "description", "assets")
    val assetType = AssetType.fromString(getObject[String](rawMap, "asset_type", "assets"))
    val assetId = getField[String](rawMap, "asset_id", "assets")
    val sponsorReference = getObject[SponsorReference](rawMap, "sponsor_reference",
"assets")
    val dollarAmount = getField[Int](rawMap, "dollar_amount", "assets")

    AssetModel(
      description,
      assetType,
      assetId,
      sponsorReference,
      dollarAmount
    )
  }

  private def extractAssetFields(model: AssetModel): (String, AssetType, String, String, Int)
=
    (model.description, model.assetType, model.assetId,
model.sponsorReference.toJsonString, model.dollarAmount)

  def createAsset(assetModel: AssetModel): Unit = {
    val (description, assetType, assetId, sponsorReference, dollarAmount) =
extractAssetFields(assetModel)

    sql"""
      INSERT INTO $assetsSqlTable ($allAssetFields)
      VALUES ($description, $assetType, $assetId, $sponsorReference, $dollarAmount)
    """.update.apply()
  }

  def getAsset(assetId: String) : Option[AssetModel] = {
    val mapOpt = sql"""
      SELECT * FROM $assetsSqlTable WHERE $assetIdField = $assetId
    """.map(_.toMap()).list.apply().headOption
    val ret = mapOpt.map(getAssetFromRawMap)
    ret
  }

  def updateAsset(assetModel: AssetModel) : Unit = {
    val (description, assetType, assetId, sponsorReference, dollarAmount) =
extractAssetFields(assetModel)

    sql"""
    UPDATE $assetsSqlTable
    SET $descriptionField=$description, $assetTypeField=$assetType,
$sponsorReferenceField=$sponsorReference, $dollarAmountField=$dollarAmount
    WHERE $assetIdField=$assetId
    """.update.apply()
   }

   def deleteAsset(assetId: String) : Unit = {
    sql"""
    DELETE FROM $assetsSqlTable WHERE $assetIdField=$assetId
    """.execute.apply()
   }

   def dangerOnlyUseWhenYouKnowWhatYouAreDoingAssets(): Unit = {
    sql"""
     TRUNCATE $assetsSqlTable
    """.execute.apply()
   }
  }
}
```

IX. References

Gillis, A. S. (2021, November 15). *Excel*. SearchEnterpriseDesktop. https://www.techtarget.com/searchenterprisedesktop/definition/Excel

Kavlakoglu, E. K. (2020, November 4). *Agile vs. Waterfall*. IBM. https://www.ibm.com/cloud/blog/agile-vs-waterfall

MySQL :: MySQL 8.0 Reference Manual :: 1.2.1 What is MySQL? (n.d.). Retrieved October 26, 2022, from https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html

Pattakos, A. (2022, January 4). *Angular vs React vs Vue 2022*. aThemes. https://athemes.com/guides/angular-vs-react-vs-vue/

Rana, A. (2022, October 27). *AWS vs Azure vs Google Cloud: Choosing the Right Cloud Platform*. Intellipaat Blog. https://intellipaat.com/blog/aws-vs-azure-vs-google-cloud/

What is AWS. (2022, November 12). Amazon Web Services, Inc. https://aws.amazon.com/what-is-aws/?trk=78b916d7-7c94-4cab-98d9-0ce5e648dd5f

Dropwizard Core — Dropwizard. (2022, November 12). https://www.dropwizard.io/en/latest/manual/core.html

Wikipedia contributors. (2022, October 9). *Scala (programming language)*. Wikipedia. https://en.wikipedia.org/wiki/Scala_(programming_language)

Wikipedia contributors. (2022, November 9). *Java (programming language)*. Wikipedia. https://en.wikipedia.org/wiki/Java_(programming_language)

memcached - a distributed memory object caching system. (n.d.). https://memcached.org/about

Redis vs. Memcached | AWS. (2022). *Amazon Web Services, Inc. https://aws.amazon.com/elasticache/redis-vs-memcached/*

Redis: In-memory database. How it works and Why you should use it | The Home of Redis Developers. (2022, November 10). https://developer.redis.com/explore/what-is-redis/

Altexsoft. (2019, October 15). *Comparing Database Management Systems: MySQL, PostgreSQL, MSSQL Server, MongoDB, Elasticsearch and others*. AltexSoft. https://www.altexsoft.com/blog/business/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/