CS509 Team Project Report

Bash

Purpose

The purpose of this document is to provide evidence for the different roles of the team members.

Introduction

The Design of Software Systems final project, called 'Auction House,' required collaboration among all team members to create a robust and functional application. Our team developed a full-stack auction house application using Amazon Web Services (AWS) and Next.js (built on React). This project aimed to simulate a large-scale webpage with accounts (buyers, sellers, and admins) and general users (customers) that can all interact with an auction to add items, search for items, bid on items, and fulfill sold items. An AWS Relational Database Service (RDS) stores important data such as accounts with encrypted passwords, items with their information, and bids on those items. The AWS Virtual Private Cloud (VPC) connects our RDS with the AWS Lambda functions to give our application access to the database. AWS API triggers call these Lambda functions from our built React application stored on an AWS S3 Bucket. The following sections detail the team's organization, process, accomplishments, a reflection on what went well, and the lessons learned.

Team organization, members, and responsibilities

The team was well-organized, efficient, and effective throughout the entire project. Two main subteams accomplished tasks assigned at full team meetings. Full team meetings occurred

at least once a week, typically with a secondary meeting during dedicated group work time in class. Each subteam would meet as necessary throughout the week to accomplish the goals at full team meetings. The broken-down organization of the team meant every team member had a partner they could directly work with while programming their use cases or divide up work further to balance out the team's contribution. The table below outlines the primary responsibilities of each teammate. In addition to the listed responsibilities, each teammate had the shared responsibilities of iteration stress testing, pull request reviewing, and contributing to the final report.

Team Member	**Main Responsibilities**
Alex Beck	Repository setup, Account verification, Tokens, Layer helper functions, Seller Login, Buyer Login, Seller Edit Item, Type creation, Error handling, README creation
Brent Weiffenbach	Setting up AWS services, Database Creation and Management, Seller Create Account, Seller Close Account, Buyer Create Account, Buyer Close Account, Publish Item, Unpublish Item, Fulfill Item, Remove inactive item, Archive Item
Emilia Krum	Seller Add Item, Seller Review Items, Seller Request Unfreeze Item, Customer Search Items, Customer Sort Items, Buyer Search Recently Sold, Customer Sort Items, Buyer Search Recently Sold, Buyer Place Bid, Buyer Review Active Bids, Buyer Review Purchases, Admin Freeze/Unfreeze, Admin Generate Forensics Report
Nathaniel Prickitt	Image storage/display, Customer View Item, Buyer View Item, Buyer Review Active Bids, Buyer Review Purchases, Admin Generate Auction Report

Process

The first step in designing a software system is defining the use cases the stakeholder needs to have implemented in the application. For CS509, these use cases were provided to us initially with an additional use case being added mid way through the project. The team then performed analysis to establish the Entity-Boundary-Controller design of the project, as well as the expected REST API methods and responses. Based on this analysis, the team used Canva to storyboard what the website should look like and how it should function from a user perspective. While there were many adjustments to the top level design of the project throughout development, these tasks helped guide what the final vision of the project should look like. Mainly, we modified the REST API design heavily for better error handling and to adjust for changes in our database schema. The database also changed throughout the project as it became clear what entities required what attributes and what foreign keys needed to exist on different tables to create the desired system. As an example, we thought each account type would be its own table in the database, but decided that a single Account database table was more efficient.

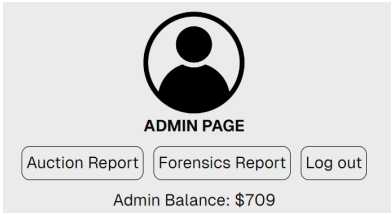
As the team began development, we employed many techniques to guarantee effective and robust work. Each team member programmed in pairs with their subteam on the assigned use cases to guarantee correctness. The team constantly referenced our API design and analysis document to keep up with naming conventions and what use cases would be interacting with one another to assure a smooth systems integration between subteams. No team member was allowed to push directly to main, requiring pull requests to be approved by another member of the team after reviewing the code to promise quality. We utilized Discord to update team members on progress, issues, or new pull requests. Discord allowed easy communication and collaboration to solve issues across multiple use cases.

Tools

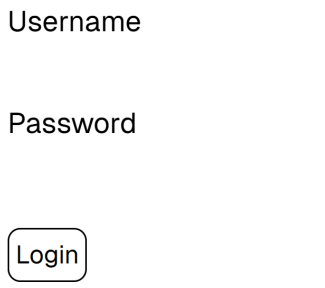
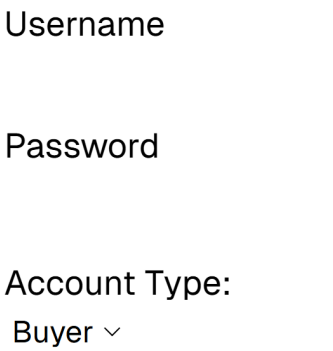
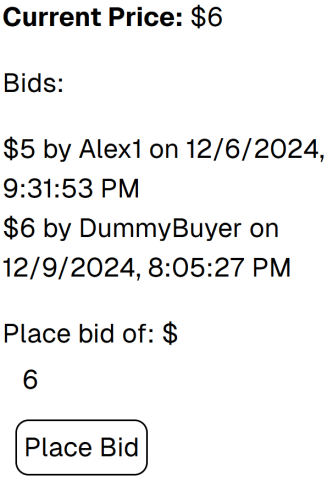
To begin the project it was first important to set up and gain experience with AWS services. After creating a fresh root user, the team created Identity and Access Management (IAM) Users so that each teammate could access the same account with the same permissions. We then created an AWS API gateway to create GET and POST methods that could be deployed onto a stage. Each method would be linked to an AWS Lambda function and could be called by HTML or React components. The project still needed to have static website hosting, so an AWS S3 Bucket was created to store the React application. Finally, we needed to set up Amazon's RDS to store data that would be used on the application. By creating an RDS with master credentials and proper permissions, the team could connect through MySQL workbench to create the database schema. With our AWS environment setup, the project was developed and managed through a React application, with version control facilitated through GitHub. We organized GitHub to have a public access folder which stored images used on the webpage, testing folder for test cases, and source folder for the main code base. To guarantee version control on all aspects of the project, the source folder contains the app, the backend Lambda functions, and useful database information. The backend folder contains folders for each use case's Lambda function(s) with the respective index.mjs and package.json. These folders can be zipped and uploaded to AWS Lambda to update any lambda functions currently deployed. The source folder also contains database information for the team to use such as the schema and sample SQL scripts to create or delete parts of the schema. Additionally, some Lambda functions used exclusively by admin for viewing and modifying the database are stored in the database folder. Finally, the actual React components contain barrelled folders for the Admin and Item page and the rest of the pages and components needed to build the website. Each team member can use the

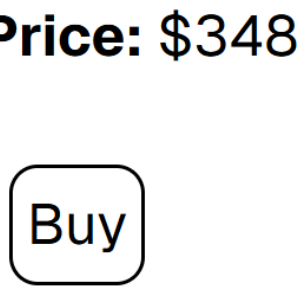
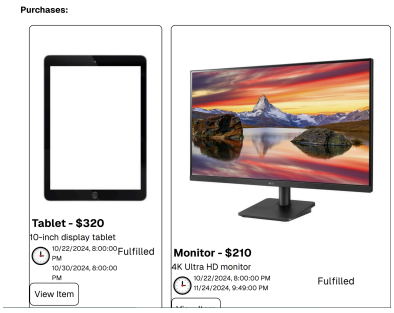
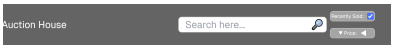
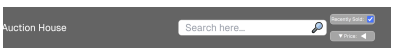
Next.js framework to test their components and backend function, and when an iteration is ready, the application can be built and uploaded to the S3 bucket for static website hosting.

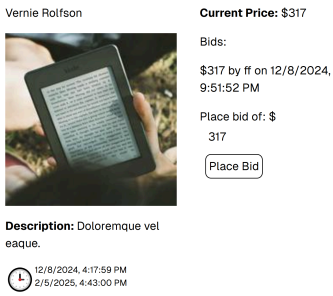
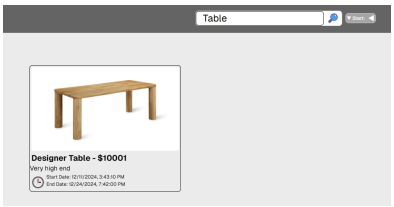
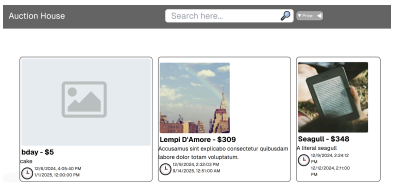
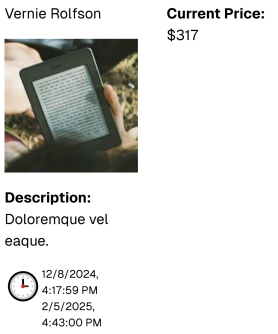
Accomplishments:

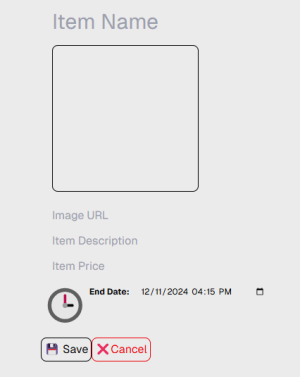
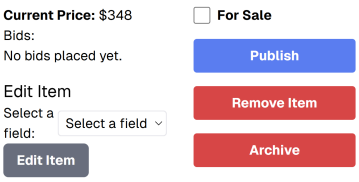
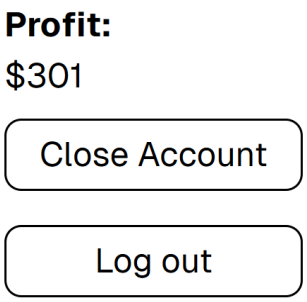
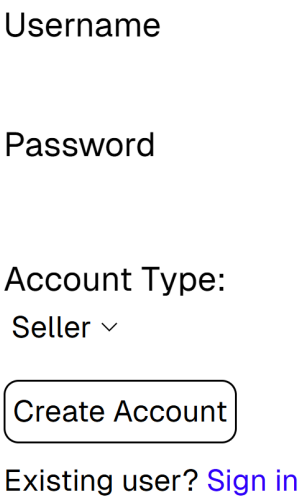
Use Case	Description of Use Case	Screenshot on Webpage
Admin: Freeze Item	An Admin can freeze items. Frozen items are unable to be bid on, and unless unfrozen, will fail. Sellers are able to request their frozen items be unfrozen.	
Admin: Generate Auction Report	An Admin can generate an auction report. The auction report includes the amount of funds that the auction house has.	
Admin: Generate Forensics Report	An Admin can generate a forensics report. The forensic report looks for items that have been completed but not fulfilled for 3 days, if bidding is open for 10 or more days, the price is higher than 10,000, and if there are more than 5 bids on an item. The report also displays accounts that have 2 or more warnings	

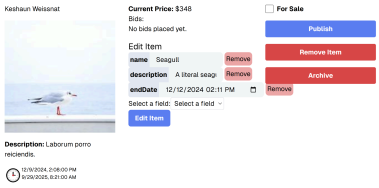
Use Case	Description of Use Case	Screenshot on Webpage
	on the accounts items	
Admin: Unfreeze Item	An Admin can unfreeze items. An unfrozen item will return to being Active, with all bids in place.	 **Andreanne Hickle - $545** Numquam blanditiis error ducimus inventore quod laudantium error adipisci quo. 12/9/2024, 2:33:40 PM 10/26/2025, 11:09:00 PM Frozen Unfreeze
Buyer: Add Funds	A buyer can add funds to their account on their account page. They can add a positive integer to the add funds entry box, click add funds, and the funds will be added to their account. Funds must be positive integers.	**Funds:** $1849 Funds # Add Funds
Buyer: Close Account	A buyer is able to close their account if it does not have any active bids. The buyer cannot reopen their account once closed. The username will be reserved, and an account cannot be created with the buyer's username again.	**Funds:** $1849 Funds # Add Funds Close Account Log out

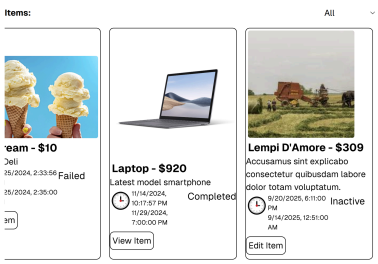
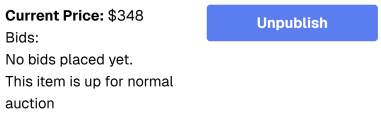
Use Case	Description of Use Case	Screenshot on Webpage
Buyer: Login	A buyer can login to their account through the login page. The buyer cannot log into an account that has already been closed.	
Buyer: Open Account	A customer is able to create a buyer account by going to create an account. Alternatively, there is a link provided within the login page that redirects the customer to the register page. A customer cannot create an account with the username of an already existing account or an already closed account. They will select buyers from a dropdown to create a buyer account.	
Buyer: Place Bid	A buyer can place a bid on an item that is active. The buyer must specify the bid amount, in whole positive numbers. The buyer must have enough funds within their account to place the bid. The funds will only be withdrawn from the buyer when the seller fulfills the item.	

Use Case	Description of Use Case	Screenshot on Webpage
Buyer: Purchase Item (For Sale use case)	After a seller places an item as For Sale, it is able to be purchased by the buyer. It is a one time purchase, with the buyer needing the required amount of funds to purchase the item.	
Buyer: Review Active Bids	A buyer can review their own bids that are active.	
Buyer: Review Purchases	A buyer can review any purchases/bids that they have made in the past.	
Buyer: Search Recently Sold	A buyer can search for items that were recently sold (within 24 hours). A sold item is one that has been fulfilled.	
Buyer: Sort Recently Sold	A buyer can sort the items that were recently sold based on criteria. Such criteria	

Use Case	Description of Use Case	Screenshot on Webpage
	include name, price, and time sold.	
Buyer: View Item	A buyer can view an item with more information than a Customer. The buyer will see all bid information for the item. The item is viewable until 24 hours has passed since the item was sold.	
Customer: Search Items	A Customer can search all active items using keywords (as a potential substring of name or description), price ranges (X-X), and dates (MM-DD-YYYY).	
Customer: Sort Items	A Customer can sort all active items by price, date (published date and expiration date).	
Customer: View Items	A Customer can view an active item (with only the highest bid shown, or initial if there are no bids) and see its description and any of its images.	

Use Case	Description of Use Case	Screenshot on Webpage
Seller: Add Item	A seller can add an item to their account if they are active. The item will initially be inactive, and the seller has to add all the information such as the item name, image, description, and initial price.	
Seller: Archive Item	An active seller can archive an inactive item. An archived item cannot be unarchived.	
Seller: Close Account	A seller is able to close an account that does not have any items that are active. The seller cannot reopen the account. The username will be reserved, and an account cannot be created with the seller's username again.	
Seller: Create Account	A customer is able to create an account by going to create an account. Alternatively, there is a link provided within the login page that redirects the customer to the register page. A customer cannot create an account with the username of an already existing account or an already closed account.	

Use Case	Description of Use Case	Screenshot on Webpage
Seller: Edit Item	An active seller can edit an inactive item.	
Seller: Fulfill Item	An active seller is responsible for fulfilling an item whose ending time has expired. The item cannot be frozen. Funds are withdrawn from the winning buyer. Fulfilling an item archives it.	**Current Price:** $650 Bids: $650 by Alex1 on 10/23/2024, 10:30:00 AM Fulfill
Seller: Login Account	A seller can login to their account through the login page. The seller cannot log into an account that has already been closed.	Username Password Login New user? Create Account
Seller: Publish Item	An active seller can publish an item that has all the required information for an item (name, description, initial price, image, end date). The item cannot be edited.	**Current Price:** $348 Bids: No bids placed yet. Edit Item Select a field: Select a field Edit Item For Sale Publish Remove Item Archive
Seller: Remove Inactive Item	An active seller can remove an inactive item, which will **permanently delete** the item.	**Current Price:** $348 Bids: No bids placed yet. Edit Item Select a field: Select a field Edit Item For Sale Publish Remove Item Archive
Seller: Request Item Unfreeze	If a seller's item is frozen, the seller can request the item be unfrozen by Admin.	**Current Price:** $40 Bids: No bids placed yet. Request Unfreeze For Sale Publish Remove Item Archive

Use Case	Description of Use Case	Screenshot on Webpage
Seller: Review Items	A seller can review their items to see which ones are inactive (not yet published), active (waiting for more bids), failed (time has expired without any bids), completed (time has expired with bids) and archived (item has been fulfilled).	
Seller: Unpublish Item	An active seller can unpublish an active item that does not have any current bids. Unpublishing an item makes the item inactive.	

Additionally, our team implemented some extra requirements that were deemed necessary for the robustness and professionalism of the project. Our accounts have encrypted passwords using Bcrypt, and when logging in the application sends users a token using JWT. Encrypted passwords is a common and necessary practice for web applications involving multiple accounts, to secure user information. Tokens allowed our team to verify what account is performing a given action, securing use cases like removing items to only be for seller users who own that specific item. With these accounts, the team felt it was important and useful for accounts to logout of the account without needing to refresh the webpage. Additionally, we created extra features for the auction house admin such as being able to view and modify the database with useful shortcut buttons or a SQL query text box. The team also implemented some features to assist in organization such as AWS layers to make shared code between lambda functions and barrelling certain components in React.

Deliverable	
Auction House Static Webpage	http://auctionhouse2024.s3-website-us-east-1.amazonaws.com/
GitHub	https://github.com/AlexanderBeck0/auction-house
Final Report	You're reading it! It is stored in github.

Reflection

Our strategies for developing this project generally worked very well and resulted in a solid software system for an auction house web application. However, several issues arose from the team's initial general lack of understanding of SQL and database structure. This caused the project's analysis to be heavily flawed where it would not have been possible for our API design to have the payloads and responses we desired. Once the database was created, the schema had to be modified multiple times throughout the first few iterations, meaning the team also had to go back and fix previously implemented Lambda functions to work with the new schema.

Our biggest mistake was with the creation of the VPS and RDS, which resulted in the AWS account being charged for the use of the application despite being on the free tier. We were able to fix this mistake by changing the RDS storage type and if it was available for public access. However, this change resulted in the added difficulty of modifying the database schema and tables, which in turn led the team to create more features for the admin.

The main change we would make for the project would be a better front-end UI. Having a better formatted and visually appealing UI would make the webpage look more professional. We would also have appreciated having better component organization in our GitHub for better recreation and documentation of the project. With GitHub, it could also have been beneficial to use ESLint to make sure pushes to GitHub were without error. Another tool, Prettier, could have

been utilized to make sure everyone's implementations have the same code structure and naming conventions.

Lessons Learned

Our team learned the importance of time management and team organization to assist with project efficiency. Dividing up tasks among the team and further into subteams was a great decision that allowed each member to work on their own time, schedule pair programming meetings, and assure quality work was completed on time. Each member or subteam being responsible for the full implementation of a use case helped all members understand the entire project system including both the AWS Services back-end and React Application front-end. The most important lesson we learned as a team was effective communication and coordination, which was crucial for the project's success. Sharing updates, resolving issues, and reviewing code collaboratively is what resulted in the final robust application. Minor miscommunications can cause significant delays or mismatched contributions which ultimately negatively affects the quality of the project.

Our advice for future teams would be to make sure each team member contributes to the full stack development of individual use cases. This reinforces every teammate's ability to debug and assist others with issues no matter if it is a front-end or back-end issue. Taking charge of the entire implementation of a use case gives all individuals on the team knowledge on all levels of designing a software system.