Online Photo Gallery with E-Commerce Support

A Manuscript

Submitted to

the Department of Computer Science

and the Faculty of the

University of Wisconsin-La Crosse

La Crosse, Wisconsin

by

Tyler Durr

in Partial Fulfillment of the

Requirements for the Degree of

Master of Software Engineering

December 2020

Online Photo Gallery with E-Commerce Support

By Tyler Durr

We recommend acceptance of this manuscript in partial fulfillment of this candidate's requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

Dr. Kenny Hunt Examination Committee Chairperson	Date
Dr. William Petullo Examination Committee Member	Date
Dr. Elliott Forbes Examination Committee Member	Date

Abstract

T. Durr, “Online Photo Gallery with E-Commerce Support,” University of Wisconsin-La Crosse, La Crosse, Wisconsin, Master’s capstone, Dec. 2020.

This manuscript serves as a description of the software engineering process for the online gallery of a local independent photographer, with a focus on the design and architecture of the system. The online gallery solution this photographer was using prior to this project did not seem geared toward smaller photographers. Online storage was limited to three gigabytes on the free tier, and higher tiers were so expensive as to be prohibitively so. Additionally, the service would take a fifteen percent commission on any sales made, further restricting the kinds of photographers who could afford it. With this in mind, the measures of success for this project were that, compared to the photographer’s previous solution, this site would be cheaper and allow for more storage while remaining efficient in its use of bandwidth and keeping personal images and transaction records secure.

Acknowledgements

To the hundreds of people in my hometown who have positively impacted me throughout my life: I hope you can each share in the pride of this accomplishment with me.

To each of my teachers throughout my time at UWL, UWMC, and in Edgar: Your genuine interest in my future has been integral in helping me reach this point, the pinnacle of my formal education. This is as much your success as it is mine.

To my project advisor, Dr. Kenny Hunt: Thank you immensely for your feedback and guidance throughout this project. It would not have been possible without your support.

To my parents, Kevin and Dawn: I would not be here without your unending love and care for me.

To my fiancée and project sponsor, Monica Bergs: Your encouragement, tolerance, and love have not gone unnoticed during this project. I can only hope that I will be able to reciprocate the love you have shown me throughout this endeavor.

Table of Contents

List of Figures

Glossary

Agile

A project management methodology which prioritizes quick and regular delivery of results to the customer, the ability to adapt to changing requirements during project execution, and frequent feedback to the customer. This is in opposition to the waterfall methodology, which is seen as inflexible and slow to generate return on investment.

Amazon Web Services (AWS)

A set of robust and responsive cloud services provided by Amazon, Inc., including compute, databases, machine learning, storage, etc.

Cookie

A small piece of data which is stored by a client browser on behalf of the server. The data is sent back to the server on every subsequent request. This enables stateful browsing, which in turn enables a server to know who you are, what preferences you have set, or anything else the server deems worthy of remembering.

Content Security Policy (CSP)

An HTTP header which specifies what resources a client should allow the current site to do, access, and/or execute.

Cross-Site Request Forgery (CSRF)

A CSRF attack involves a malicious third-party site making requests to another site from the user's browser. Because the request comes from the user's browser, it will include any authentication the user has given. Notably, this means the malicious site can perform any action the user would be able to.

Cross-Site Scripting (XSS)

An attack vector which involves causing a user to unwittingly execute scripts in their browser. The scripts are "injected" into otherwise trustworthy websites in one of two ways: persistent attacks, which involve storing the script on the server so that the user unknowingly executes it whenever they navigate to the resource which contains it, and reflected attacks, which are so named because the script is never stored on the server, but rather reflected, by the server back to the user via an error message, search result, or some other page which displays user input.

DomainKeys Identified Mail (DKIM)

Using DKIM on outbound emails allows recipients to verify that an email claiming to be sent on behalf of a specific domain, e.g., `monicajean.photography`, came from an authorized email sender for that domain. It achieves this by employing a cryptographic hash of the email. It also ensures that the contents of the email have not changed while in transit.

Domain Name System (DNS)

A network of servers on the Internet which acts as the definitive database of mappings from domain names to IP addresses. It can also store other information about domains, such as who can send the domain's emails, who can issue the domain's HTTPS certificate, ownership verification keys for external services, and even arbitrary data.

Elastic Compute Cloud (EC2)

An AWS cloud service used to provide computation capacity via a remote virtual machine running Linux.

Express

A Node.js web server which enables the use of middleware on each request it receives, provides facilities for routing, and wraps Node.js's http module with utility methods for HTTP methods and error handling, among other things.

Less

A CSS preprocessor. Less transpiles into CSS with the use of an included transpiler. The language adds features to CSS like the ability to nest blocks of CSS within each other.

Let's Encrypt

An HTTPS certificate authority which provides certificates for free. Certificates requests are authenticated using a challenge which proves that the requester has control over the domain they are requesting a certificate for.

Node Package Manager (NPM)

A software registry which hosts open-source packages which can be downloaded and installed for use in projects.

Node.js

A runtime for JavaScript code which operates independent of a browser. The standard library for this runtime enables file access, network communication, and multithreading, among many other things.

Pug

An HTML templating language available as an NPM package which can be used in any Node.js project.

Scrum

An agile methodology which is characterized by splitting a project into sprints. Each sprint takes in a subset of project requirements, implements them, demonstrates them to the customer, and then holds a retrospective to facilitate frequent reflection and improvement.

Sender Policy Framework (SPF)

A DNS TXT record which lists the IP addresses authorized to send a domain's outbound emails.

Session

A sequence of HTTP requests made by one user using one client. Sessions are managed using a variety of techniques and are often (although not exclusively) used for authentication. (Note: In the photography domain, the word session is also the name for a meeting between a photographer and their client for the purpose of getting pictures taken. For the sake of clarity, in this report, HTTP sessions will be explicitly referred to as *HTTP* sessions.)

Simple Email Service (SES)

An AWS cloud service which provides email sending and receiving, as well as DKIM signing.

Simple Storage Service (S3)

An AWS cloud service used to store “objects” in “buckets”, essentially the cloud names for files and folders.

Sprint

One phase of an Agile Scrum-managed project which lasts between one and four weeks. At the end of each sprint, one or more parts of the project should be fully implemented and tested, such that they can be provided to the customer for immediate use.

WebP

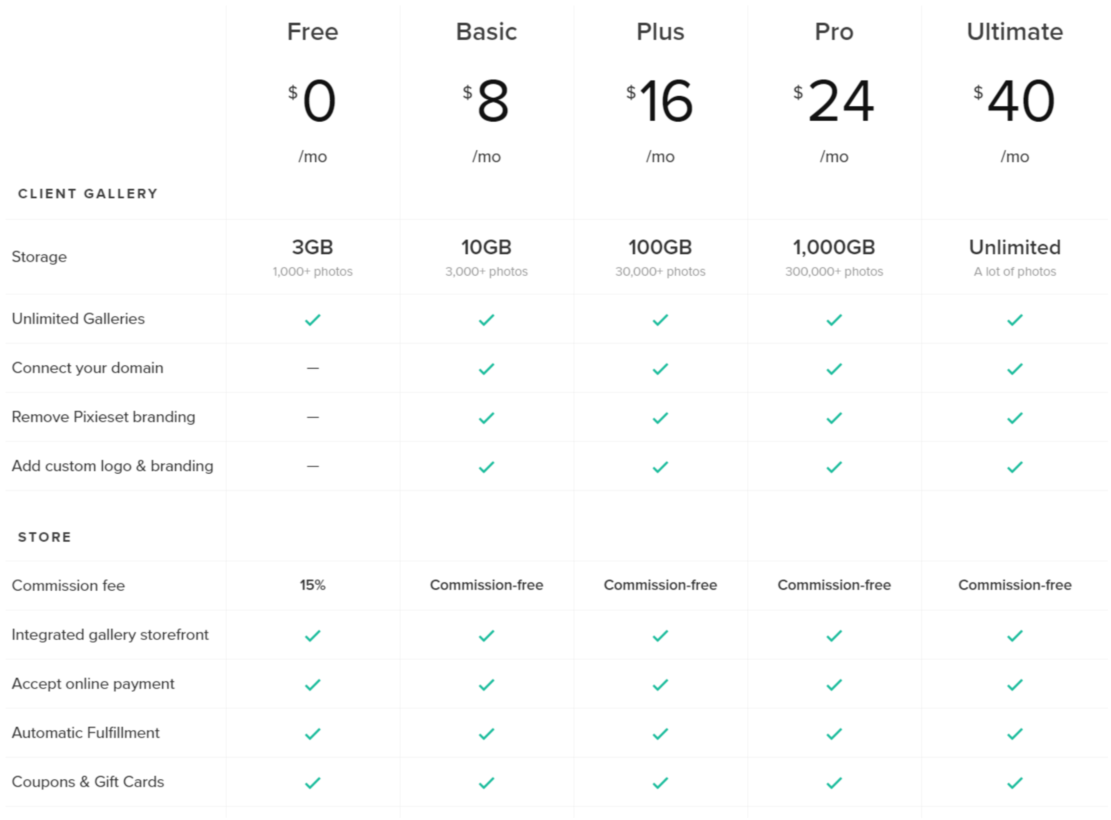
A lossy image format released by Google in 2010. Rabbat’s stated goal for the format was to “further compress lossy images like JPEG to make them load faster, while still preserving quality and resolution." [1]

1. Introduction

Monica Jean Photography is a local, independent photography business in La Crosse, Wisconsin. The proprietor most commonly takes pictures for events at the local YMCAs in La Crosse and Onalaska. She also does staged photography sessions, including engagements, couples, and families. In total, this amounts to about two dozen sessions per year, each session having about 100 pictures each. Once a session has been completed, the unedited images are culled down to the best-quality ones, i.e., properly lit, in focus, visually appealing, etc. Those images are edited by the photographer and then uploaded to an online gallery for the client to view and, optionally, download.

1.1. Impetus

Prior to this project, Monica Jean Photography was using an online photo gallery service called Pixieset. Pixieset offers photographers a way to share their finalized, edited images with clients through a web-based interface. Clients are then able to purchase prints of their images from the website. The site also offers a choice of fonts, themes, and image layout styles for the store and galleries [2].



	Free	Basic	Plus	Pro	Ultimate
	$0 /mo	$8 /mo	$16 /mo	$24 /mo	$40 /mo
CLIENT GALLERY					
Storage	3GB 1,000+ photos	10GB 3,000+ photos	100GB 30,000+ photos	1,000GB 300,000+ photos	Unlimited A lot of photos
Unlimited Galleries	✓	✓	✓	✓	✓
Connect your domain	–	✓	✓	✓	✓
Remove Pixieset branding	–	✓	✓	✓	✓
Add custom logo & branding	–	✓	✓	✓	✓
STORE					
Commission fee	15%	Commission-free	Commission-free	Commission-free	Commission-free
Integrated gallery storefront	✓	✓	✓	✓	✓
Accept online payment	✓	✓	✓	✓	✓
Automatic Fulfillment	✓	✓	✓	✓	✓
Coupons & Gift Cards	✓	✓	✓	✓	✓

Figure 1. Pixieset's pricing options

Unfortunately, even Pixieset's free pricing option is not very conducive to a small business like Monica Jean Photography. The free version offers three gigabytes of storage. Pixieset [3] suggests that this amount of storage would hold 1,000 images. With the image counts of some sessions reaching into the hundreds, this storage would be filled very quickly, which would mean needing to choose which images to save and which to delete. An option with unlimited storage narrowly misses $500 per year.

1.2. Project Goals

The goal of this project was to provide Monica Jean Photography with a cheaper alternative to Pixieset. This project would offer reduced functionality while maintaining security, reliability, and efficiency. Some features were altered from Pixieset as desired by the customer. One change made was to omit the user-configurable parts of the site's style. Pixieset offers design options because each of its users may have their own preferred styles; here, the site was directly designed for one photographer. The primary change to the functionality of the site was to require all users to log into the site with an email address and password. Other changes were derived from this change, such as associating each session to a client, who would be the only authorized viewer for the session.

2. Requirements

2.1. Overview

This section discusses the development methodology chosen for this project, why it was chosen, and details how this methodology was followed throughout the project's lifecycle.

2.2. Development Methodology

The stakeholders in this project are the project owner and sponsor, photographer Monica Bergs; project advisor Dr. Kenny Hunt; Tyler Durr, the developer; and the end users of the product, the clients of Monica Jean Photography.

In choosing a methodology, an important consideration was that this website, like most others, will continue to be developed long after the initial implementation during this project. Therefore, it was important that there would not be a final transition from implementation/testing to a maintenance-only phase.

A second consideration was that the methodology would need to be amenable to changing requirements. As noted previously, the sponsor did not want all of Pixieset's features precisely mirrored. By default, a photographic session on Pixieset can be viewed with only an email address and no password. The session and all images in it are essentially public, available to anyone who can fake an email address. To restrict access to a typical email address and password login experience, one must provide a list of authorized email addresses, and even then, all email addresses share the same password, which can only be set or changed by the photographer. Image downloads employ a separate PIN which must also be emailed to any authorized viewers. Many differences between Pixieset and this project stem from the decision to use conventional email address and password accounts and to restrict each session to access by only one user. Although Pixieset provided a core framework for describing the requirements of this project, both the developer and sponsor recognized that certain features would undergo significant change and, hence, adaptability was a key consideration in the selection of a lifecycle model.

According to Digital.ai Software, Inc. [4], 95 percent of people involved in software development utilize some agile methodology in whole or part at their workplaces. Of those 95 percent, 58 percent said the specific version of agile that they use is Scrum. This shows that many current workplaces are using Scrum, a methodology which has the two properties listed above.

Scrum was therefore chosen for this project's development methodology because it is amenable to changing requirements, can be continued indefinitely, and has wide use in the industry already.

On large Scrum-managed projects with multiple developers, the developers break into teams for each sprint. These teams should be self-managing groups of no more than ten people. Each team should be able to work independently, meaning that every team must have all the necessary knowledge to turn a backlog item into an implemented and tested product feature [5]. Since this project only had one developer, the Scrum methodology was modified as described in the following paragraphs.

In general, the Scrum model specifies that daily meetings be held among team members. The common understanding of these meetings is that at the beginning of each day, each team would hold a meeting called a daily scrum, where each team member discusses what they completed in the previous day, what they plan to complete that day, and anything which potentially obstructs them. This meeting is sometimes called a daily standup, an allusion to the idea that the meeting should be short enough that attendees could or should remain standing for it [6]. Daily scrums were omitted during this project and replaced by a weekly scrum with the project advisor held via an online videoconferencing site.

The Scrum model also specifies that at the end of the sprint, the combined teams conduct a sprint review to review their progress and present any new features to the customer for immediate release. In this way, the project very quickly begins to create value for the customer because parts of the project will be completed and ready to use even though the entire project is not. Sprint reviews were conducted as an in-person meeting with the project sponsor at the end of each sprint to discuss the new features implemented in that sprint.

Furthermore, Scrum specifies that at the end of each sprint, each team holds a meeting called a retrospective. This meeting is an invitation to reflect on the sprint's productive and counterproductive moments and brainstorm ways to increase productivity in future sprints [5]. For this project, retrospectives were entered into a digital notebook so they could be reviewed at later times.

To implement Scrum in this project, first, an initial product backlog was generated. The product backlog is a prioritized list of all known things which have yet to be implemented in the project, including requirements, changes in requirements, bug fixes, etc. Each item on this list is called a backlog item. Each backlog was assigned a number which estimates the amount of work it will take to complete that item, called storypointing. Finally, the top of the project backlog was assigned to a timeboxed section of work called a sprint. This is done using the storypoints assigned to each backlog item to estimate velocity, or how many storypoints can be completed in a sprint [5]. Each sprint was two weeks long. The set of backlog items implemented in a sprint is called the sprint backlog.

Throughout nine sprints in this project, 51 backlog items totaling 198 storypoints were completed. The average velocity across all sprints was twenty-two storypoints per sprint. Figure 2 is a graph of velocity by sprint, showing a general gradual increase in the number of storypoints taken in per sprint. The large spike and subsequent dip in sprints four and five demonstrate the flexibility of Scrum.

A foreseen project-external time commitment occurred during sprint five. This commitment significantly reduced the amount of effort that the developer could devote toward the project in that sprint. Because all backlog items were storypointed when they were created, it was possible to accurately predict the amount of work that could be completed with the reduced amount of work time.

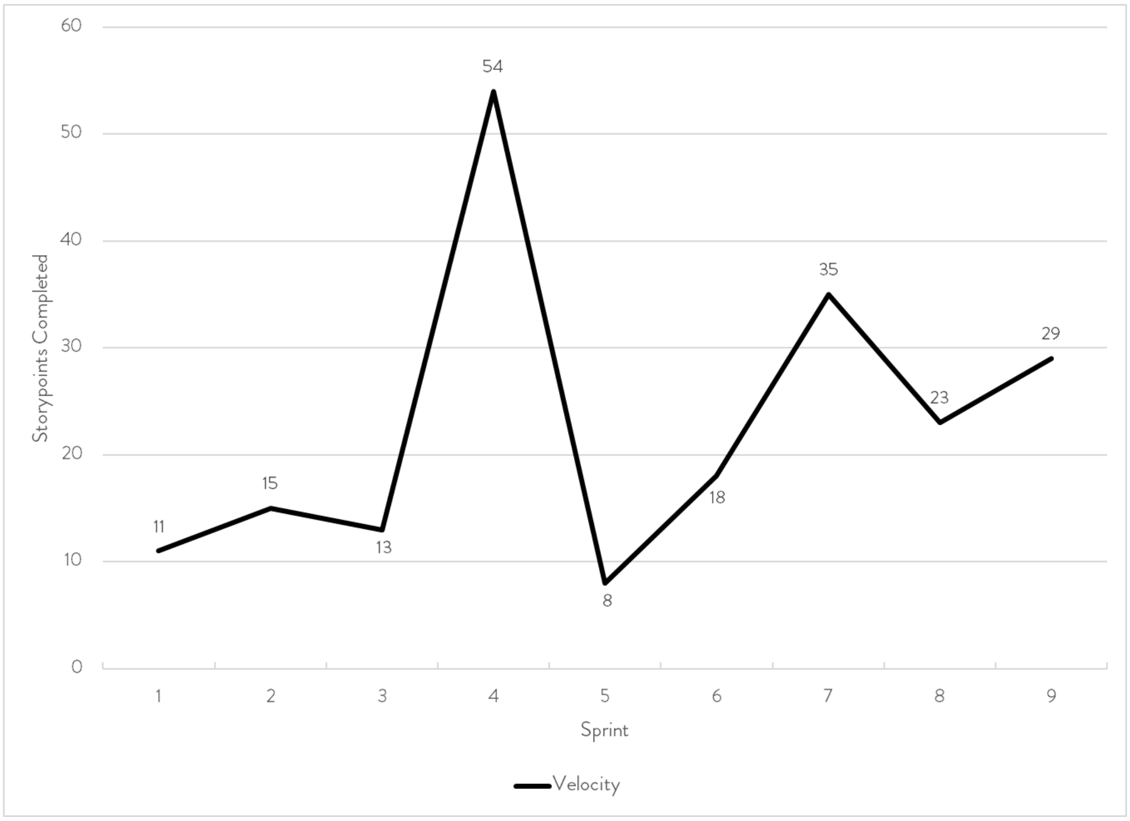


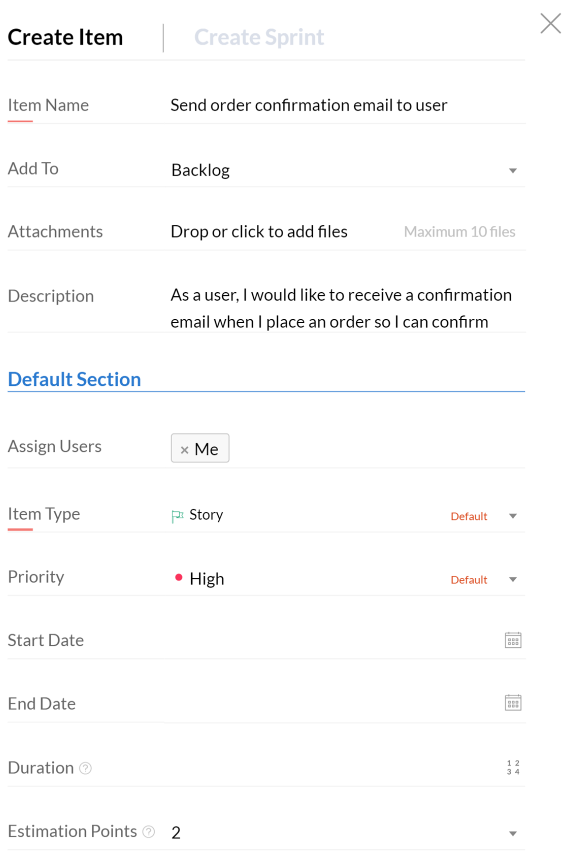
Figure 2. Velocity per sprint.

2.2.1. Project Management

Zoho Sprints is the web-based agile management tool used to document and track backlog items throughout this project. By entering storypoints for each backlog item as they are added, Sprints was able to estimate how many backlog items could be completed in future sprints and therefore estimate how long it would take to complete all backlog items, i.e., complete the project. An example of creating a new backlog item can be seen in Figure 3.

Zoho Sprints also includes a kanban board which was used to track the progress of each individual backlog item. The kanban board is divided into columns representing different states of a task's development workflow [7]. This project classified tasks as “To do”, “In progress”, or “Completed”. At the beginning of each sprint, a card was automatically generated for each item in the sprint backlog. As development began on each task, the task was flagged as being “In progress” and moved to “Completed” once finished. This kanban board provided a quick overview of progress in a sprint.

Zoho Sprints also helped determine whether work was lagging behind within a sprint by employing a burndown chart. A burndown chart shows the amount of incomplete storypoints in a sprint compared to the time in that sprint. Ideally, with a consistent amount of work being completed every workday, the burndown chart would be a straight line from 100 percent incomplete at the beginning of the sprint to 0 percent complete at the end. These burndown charts were useful in identifying that work often lagged in the beginning of each sprint and



Item Name	Send order confirmation email to user
Add To	Backlog
Attachments	Drop or click to add files Maximum 10 files
Description	As a user, I would like to receive a confirmation email when I place an order so I can confirm

Default Section

Assign Users	× Me
Item Type	Story Default
Priority	● High Default
Start Date	
End Date	
Duration	
Estimation Points	2

Figure 3. Adding a new backlog item in Zoho Sprints

caught up near the end.

In Figure 4, all nine sprint burndowns have been superimposed into one graph, along with an ideal burndown in dots and the actual average burndown in dashes. From this chart, a clear pattern emerges: Work lagged at the beginning of each sprint and then rapidly caught up at the end.

A shortcoming of Zoho Sprints is the inability to change backlog items after a sprint has begun. This lack of flexibility led to inconsistencies in the recorded project state since some backlog items in Zoho Sprints do not exactly reflect the feature implemented. For example, a feature in sprint 4, “Downsize images to max dimension of 600 px when uploading”, was revised to to change the size of the picture, but the backlog item remains as initially titled.

2.3. Roles

There are two types of user: administrator and client.

An administrator will create all client accounts. They cannot elevate a client account to be an administrator. The admin will also create all products, and can view, edit, and delete existing ones. Once a client account has been created, an admin can create a new

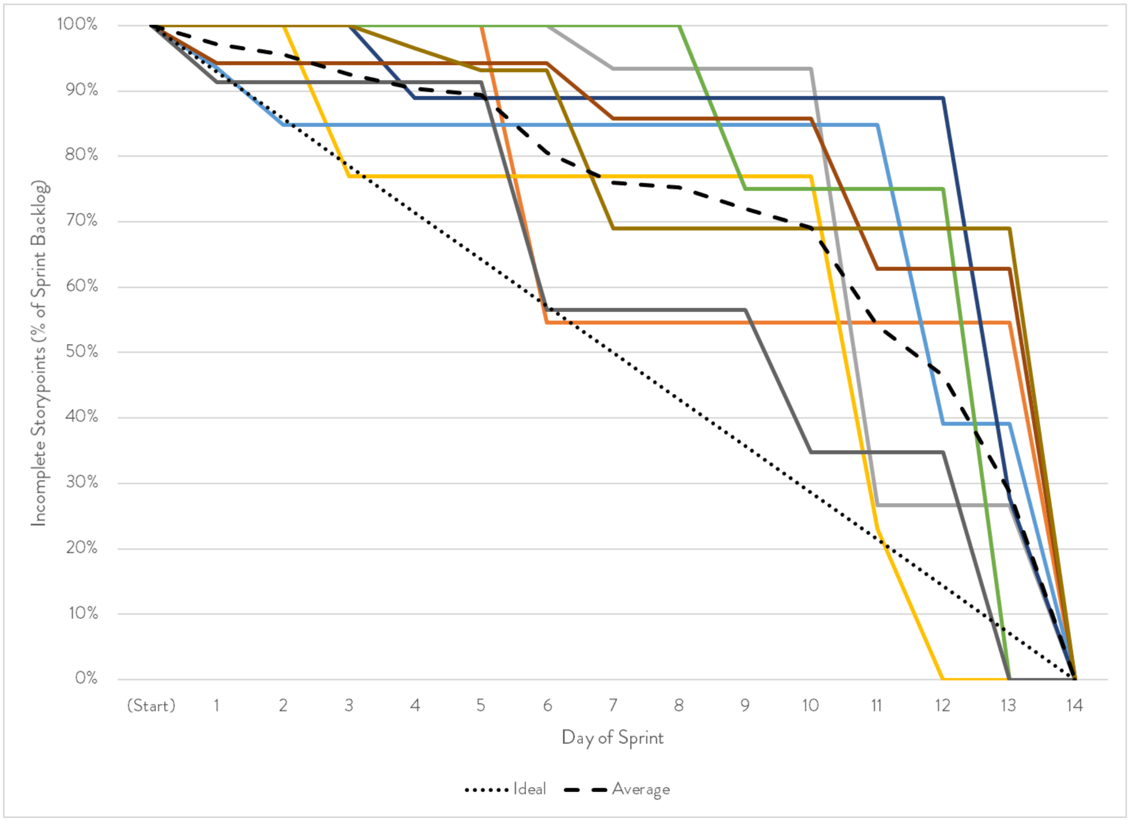


Figure 4. Each sprint burndown overlaid into one graph. Each solid line represents The dotted line represents the ideal burndown. The dashed line represents the average of the actual burndowns.

photographic session for the client and edit any photographic sessions which have been previously created. After a client makes a purchase through the site, the admin will be able to view the contents of that order so that it can be fulfilled, but they cannot delete or edit it.

Clients can view only the sessions which an administrator has created for them and only the orders which they have made. They can only edit their own account information. Clients may also make purchases from the site; this is the only way an order can be created.

2.4. Gathering

The initial set of requirements were gathered from a meeting with the project sponsor where we reviewed Pixieset's features and discussed which of those features she would like to see in this project, as well as additions or changes to those features. After the initial gathering, requirements and backlog items were added as they were encountered.

As progress was made and meetings were held with the sponsor and advisor, requirements (and, therefore, backlog items) were added as necessary. The sprint in which each requirement was implemented is listed in parentheses below. Note that this project stemmed from a prototype made in an independent study class prior to the beginning of the capstone project, so some requirements were already implemented before the project officially began and have no implementation sprint.

2.5. Functional

Functional requirements are those that describe what a system does—in other words, what *functions* the system can perform [8].

The following high-level functional requirements were gathered throughout the duration of the project:

- As any user, I would like to...
  - log in and log out of my account.
  - reset my own password via email if I forget it. (7)
  - edit my own information, including my name, email, and phone number.
- As a client, I would like to...
  - view a list of sessions I've had. Only sessions which haven't expired will be shown.
  - view the images in my sessions.
  - download the images in my sessions, if allowed by the administrator. (9)
  - view a list of products available for purchase.
  - select one or more of these products to purchase per image.
  - view my cart, including shipping, tax, and a grand total.
  - check out via PayPal, where shipping and payment information would be entered.
  - receive a confirmation email when my order completes. (5)
  - view a list of orders I've made in the past. (4)
- As an administrator, I would like to...
  - view a list of my clients.
  - search clients by name. (7)
  - create new clients by entering their information.
  - send newly-created clients their login info via email. (5)
  - edit my clients' information.
  - send my clients a password reset link via email. (7)
  - view the sessions and orders linked with each client. (7)
  - view a list of all orders.
  - view the contents of any order, complete with shipping address. (9)

- create a new session tied to a client, with a name, expiration date, a toggle to enable downloads, a cover image, and the images in the session.
- send a client an email when a new session is added to their account. (7)
- edit a session's metadata.
- download the images in any session. (4)
- view a list of all products. (6)
- search products by name. (6)
- create new products, including a name, price, and an image of the product. (6)
- edit products. (6)
- delete products. If a client has a deleted product in their cart, it will be removed, but they will be notified as such. If they previously purchased that product, it should still show in their order history. (6)
- send and receive email the `monicajean.photography` domain from within the Gmail account I already have.

2.6. Non-Functional

In contrast with functional requirements, non-functional requirements describe *how* a system performs its functions [8]. They essentially focus on *qualities* the system should have, including security, performance, maintainability, etc.

The following high-level non-functional requirements were gathered throughout the duration of the project:

The system should...

- support strong authorization and authentication. (HTTPS in sprint 1; CSRF added in sprint 5; CORS restricted in sprint 9)
- be automatically deployable upon code push to the repository. (3)
- remain efficient on low-powered mobile devices or those with a limited data plan. (Image thumbnails introduced in sprint 4; HTTP/2 originally introduced in sprint 2; other image optimizations made in sprint 4)
- store no payment information.
- be cheaper to operate than Pixieset.
- recover from errors gracefully, without crashing the entire site.

2.7. Changes

Throughout the project, changing customer demands warranted changing requirements. At one point, it was possible to add a single session to multiple users. It was only after implementation and presentation at sprint review that the sponsor determined that the feature was unnecessary. Another example is image download functionality. Early on in the project, downloads were to be restricted to a certain number of images rather than the full session. This was later changed to an all-or-nothing configuration. Such changing demands were a source of many backlog items. Fortunately, Scrum is amenable to such changes, and simply adding a new backlog item was generally sufficient to incorporate the change.

3. Design

3.1. Overview

This section discusses the architecture of the project, its classes, database schemata, and the user interface.

3.2. Architecture

DNS services for the project's domain, `monicajean.photography`, are provided by Google Domains, the registry through which the domain was originally registered.

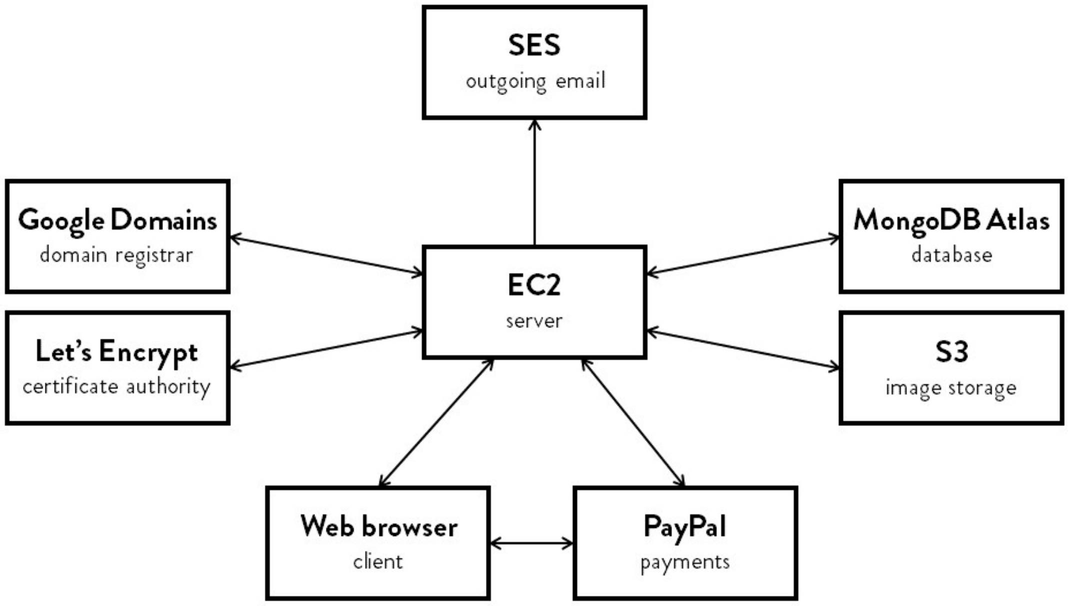


Figure 5. General application architecture overview

The project utilizes multiple Amazon Web Services (AWS) services to store and serve its content.

AWS Simple Storage Service (S3) provides cloud-based storage for networked applications. This project used S3 to store all administrator-uploaded images and the thumbnails of those images. Images are saved in an S3 *bucket*, analogous to a file directory. Each image being stored in the bucket is uniquely identified by its ID from the database. S3 offers multiple storage classes, which provide different access speeds and resiliency levels depending on the way in which the data being stored will be used. This project uses the Standard - Infrequent Access class, which is for "long lived but infrequently accessed data that needs millisecond access" [9].

AWS Elastic Compute 2 (EC2) is an cloud computing service which provides the ability to create remote virtual machines, called instances. An EC2 instance running Ubuntu is used as the server. Connections to the server are secured with a 2048-bit RSA key pair. This server instance can be easily cloned or upgraded if necessary to handle larger amounts of traffic in the future. (AWS also provides a load balancer to distribute traffic between multiple servers.)

Node.js is a JavaScript runtime which runs independently of any browser. The server runs as a Node.js application on the EC2 instance, utilizing fifty-eight different Node Package Manager (NPM) dependencies in its efforts. The server is also in charge of renewing the domain's HTTPS certificate with certificate authority Let's Encrypt and keeping its own IP address up to date in Google Domains' DNS records.

The EC2 server is the hub through which almost all communications flow: S3 uploads and downloads are streamed through it; outgoing emails are sent using SES by it; database connections are established by it; and all pages are rendered and served by it. Only incoming email and PayPal payment information don't pass through this server.

3.2.1. Class Diagram

Figure 6 shows the final class diagram for the project.

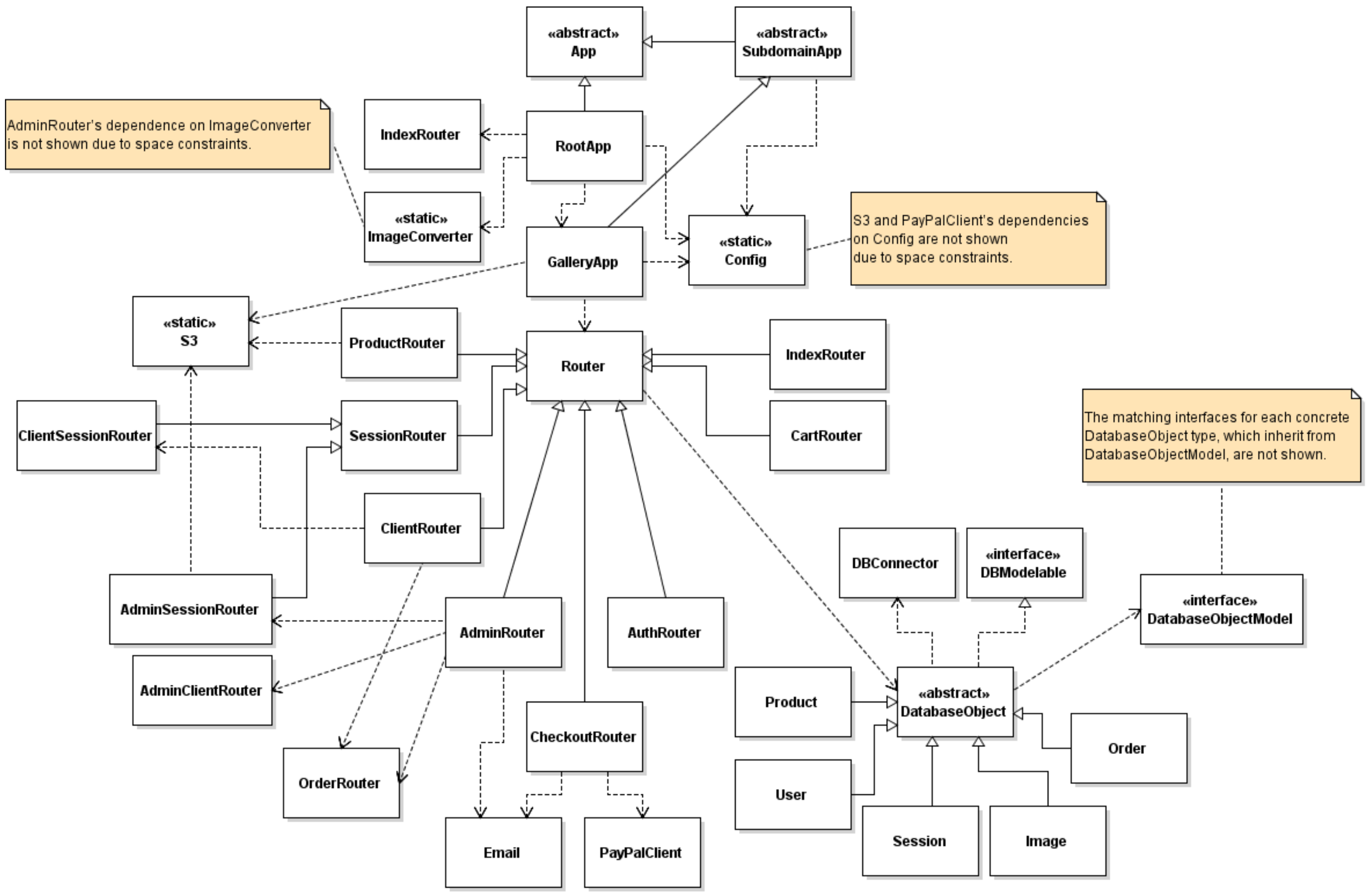


Figure 6. Class diagram for the entire project

All `-Router` classes ultimately descend from the abstract `Router` class. These classes contain the various endpoints of the project and the logic which is triggered by them. Routers' names describe the kind of endpoints they handle, e.g., `AuthRouter` contain endpoints related to authenticating users.

All `-App` classes ultimately descend from the abstract `App` class. Both of the concrete `-App` classes each handle a subdomain or set of subdomains, with `GalleryApp` handling `gallery` and `RootApp` handling the base domain. They contain handlers which are universal to all routes under their control and provide a mounting point for those routes.

`DatabaseObjectModel` and `DatabaseObject` are the parents of the interfaces and concrete classes, respectively, which represent each collection in the database. They contain methods to get and set properties from documents in these collections. All `DatabaseObject`s implement the `DBModelable` interface, which contains properties necessary to properly store documents back in the database. Each of the children of `DatabaseObject` and `DatabaseObjectModel` is discussed in further detail in the Implementation section.

`S3`, `PayPalClient`, `Email`, and `DBConnector` contain methods the application uses to connect to S3, PayPal, email service Simple Email Service (SES), and MongoDB, respectively. These classes do not fit into the routing or database hierarchy, but nevertheless contain conceptually-related methods. The `Config` class also connects to an “external” system, but that system is just a locally-stored JSON file containing API keys and secrets.

3.3. Database

The database is hosted by MongoDB Atlas, a cloud-based interface for MongoDB databases operated by MongoDB, Inc. MongoDB Atlas is also hosted on AWS servers.

MongoDB was chosen for its wide support, ease of use, and ability to accommodate changing requirements.

According to solid IT [10], MongoDB is the most popular non-SQL database engine in the world. This popularity naturally results in a litany of packages which have been developed for use with MongoDB. It also means that plenty of resources for usage, learning, and troubleshooting exist, easing the development process.

Unlike SQL databases, which store records as rows in a table, MongoDB stores data as documents in a collection. These documents have no prescribed structure, so they do not have to conform to any predefined schema, though such a schema can be enforced in the implementation.

Throughout the project, the fields needed on each of the different entities has changed often:

1. `User.phoneNumber`: added to store clients' phone numbers.
2. `User.fullName`: divided into `User.givenName` and `User.familyName` to enable more friendly-sounding emails (“Hello, Jane” versus “Hello, Jane Smith”).

3. `User.isABusiness`: added to support business clients, like the YMCA, specifically with respect to their names. If this value is true, the full name is just the given name; if false, the full name is the given and family name.
4. `User.passwordReset`: holds the token which is verified during password resets.
5. `Session.shortDescription` and `Session.longDescription`: removed by customer request due to lack of usefulness.
6. `Session.coverImage`: added to identify the cover image for display in session lists.
7. `Session.expirationDate`: added to enable sessions to expire, hiding them from client view.
8. `Product.longDescription` and `Product.shortDescription`: removed by customer request due to lack of usefulness.
9. `Image.originalName`: added to be displayed on admin order view so that the administrator can more easily find the image purchased on their own computer.

With the changing set of fields needed from the database, MongoDB's simple flexibility has not gone unappreciated.

3.3.1. Collections

Multiple schemata may exist within a single collection. For each collection, the comprehensive schema will be shown (from the TypeScript code which represents it), and any variety in that schema will be explicitly mentioned.

All documents in all collections are guaranteed to have two keys: `_id`, a uniquely-identifying 24-character hexadecimal string mandated by MongoDB; and `active`, a boolean which is set to `false` when the document is soft deleted, mandated by `DatabaseObject`.

The ER diagram is shown in Figure 7.

`User` A `User` represents one administrator or client. Each `User` contains personal data such as `phoneNumber` and a `given`- and `familyName`. The `familyName` of a `User` would be empty if `isABusiness` is true. It also contains information about the account itself: `admin` represents whether it is an administrator or client account and `passwordReset` holds the token for the currently-outstanding password reset request, if there is one. `sessions` and `orders` are arrays of `ObjectID`s, the type of the unique identifiers mentioned above. While the user is going through the checkout process, `outstandingOrder` holds the ID of the order they are completing; otherwise, it is empty.

Login credentials are specified by `emailAddress` and `passwordHash`. The email address must be unique among users; this is enforced by having an index on the `emailAddress` field which requires as such and checking for duplicate addresses before every update of an email

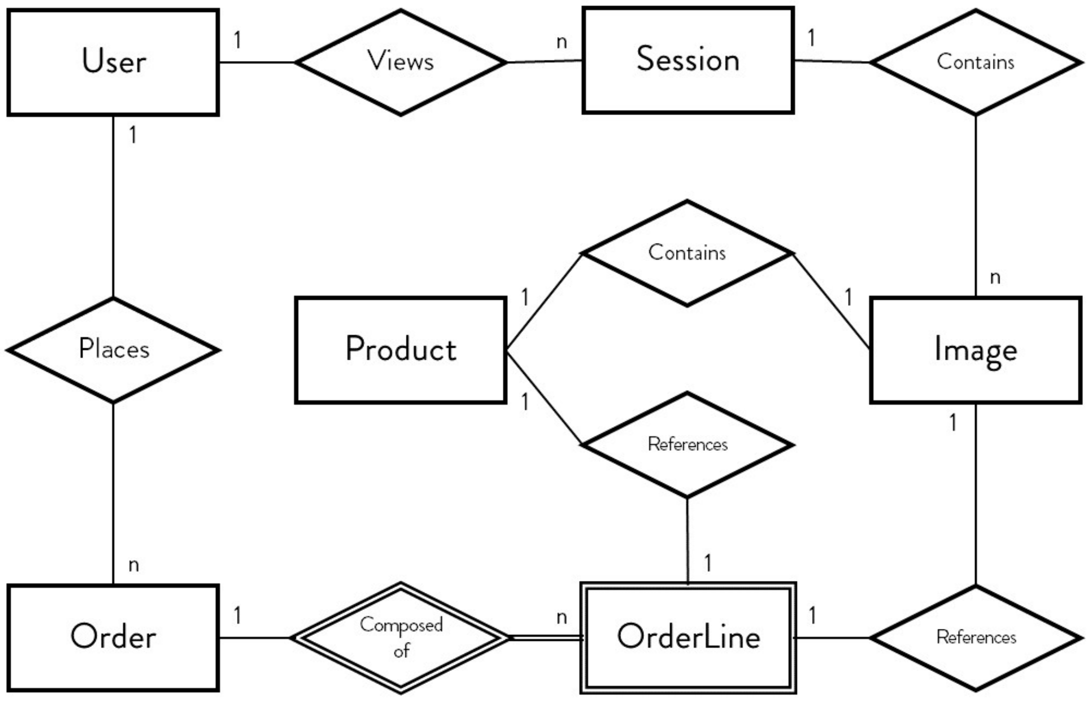


Figure 7. ER diagram

address. To ensure the security of users' passwords, only the salted hash, and never the password itself, is stored. The hash function is a cryptographic function which is extremely difficult to reverse, analogous to trying to unmix two colors of paint. Whenever a login is attempted for an account that exists, the candidate password entered is hashed by the same algorithm as the one which generated `passwordHash`, and the two are compared for equality, which indicates that the correct password was entered.

The `cart` property is a JSON object unto itself, which contains a key for each item in the cart. Each of these items, in turn, contains a JSON object, containing the products being ordered for that item. That is also a JSON object listing the quantity of each item. To represent this complex object in a relational database, one would need to split it up into separate rows for each user-item-product tuple. Then, whenever the cart needs to be retrieved, it would have to be reconstructed from that list of rows. And all of this would be stored in a table separate from where the user exists, despite it being inextricably linked to that user. In MongoDB, however, the two can be stored together, and the object will be serialized and deserialized automatically. This reduces the amount of boilerplate code needed to use such a value. An example of a cart is shown in Figure 8.

`Session` A `Session` is a set of `Image`s, where the images share a common subject matter. That subject can be an event, like a wedding, awards ceremony, or birthday party; people, like a family, couple, or high school senior; or even an object, like a newly-released product or remodeled building. The `coverImage` of a `Session` is the image which is displayed when the session is viewed as part of a list of `Sessions`. The `expirationDate` is the date on which the

```
cart: {
   items: {
      "5fb99650c641f83d3802d671":{
         products: {
            "5fa9fd904df77f2a90ef0627":{
               quantity: 4
            },
            "5fd25272f957277160b8dec0":{
               quantity: 2
            }
         }
      },
      "5fb9964fc641f83d3802d667":{
         products: {
            "5fa9fd904df77f2a90ef0627":{
               quantity: 5
            }
         }
      }
   }
}
```

Figure 8. This cart contains some products which are being ordered for two separate images.

client may no longer view or order products from this `Session`. `downloadable` determines whether the client is allowed to download the original, full-resolution JPEG images from the `Session`.

`Image` An `Image` is the database representation of all the different sizes and formats which the original file exists in. `fileType` is the type of the original file, and is a value from an enum containing the valid file types which can be uploaded to the website by an administrator (at this time, only JPEG and WebP are supported). `originalName` is the name of the uploaded file as it was on the administrator's computer, and `md5Hash` is the hash of that file, required when uploading to AWS S3. `webMD5Hashes` is an object with a key for each file type the server downsizes the original image into. Those optimized thumbnails are hashed and the results are stored here.

`Product` A `Product` is the database representation of a type of photographic print that a client can order. It includes the `name`, `description`, and `price` of the product, as well as an `image` of it (the `ObjectID` of an `Image`). To avoid the rounding risks associated with performing arithmetic on floating-point values, `price` is the cost of the product in cents, which is always a whole number. For example, a price of $9.95 would be stored as 995.

```
{
  admin: boolean;
  emailAddress: string;
  phoneNumber: string;
  passwordHash: string;
  givenName: string;
  familyName: string;
  isABusiness: boolean;
  sessions: ObjectID[];
  cart: Cart;
  orders: ObjectID[];
  outstandingOrder: ObjectID | null;
  passwordReset: string | null;
}
```

Figure 9. Comprehensive `User` schema

```
{
  date: Date;
  title: string;
  images: ObjectID[];
  coverImage: ObjectID;
  expirationDate: Date;
  downloadable: boolean;
}
```

Figure 10. Comprehensive `Session` schema

`Order` and `OrderLine` An `Order` is a list of products which a `User` has already or is about to purchase. `completionDate` is the datetime when the `Order` payment completed. The `transactionProcessor` is a value from an enum representing the company through which the payment was processed (currently, the only value is `PayPal`). `processorOrder` holds the entire JSON object which is transmitted by the payment processor when the payment is made, containing all available information about the transaction. These three values are `null` until the checkout process is completed and payment made. The `tax-` and `shippingTotal` are stored on the completed `Order` rather than being calculated each time because the calculations which result in these values can change over time.

`OrderLine`s cannot exist independent of `Order`s, hence their representation as a dependent entity in the ER diagram for this project. They contain links to the item and product they correspond to, the quantity that was ordered, and the cost per unit for that item at the time the checkout process was initiated. Cost per unit needs to be stored on the order in case this value would change in the future.

```
{
  fileType: FileType;
  originalName: string;
  md5Hash: string;
  webMD5Hashes: { [key in FileType]: string };
}
```

Figure 11. Comprehensive `Image` schema

```
{
  active: boolean;
  name: string;
  price: number;
  image: ObjectID;
  description: string;
}
```

Figure 12. Comprehensive `Product` schema

3.4. User Interface

One of the most important parts of designing a pleasant user experience is making the interface design consistent throughout [11], [12], [13]. This consistency is achieved through the use of HTML templating language Pug and CSS preprocessing language Less.

3.4.1. Pug

The purpose of Pug is to enable the server to render dynamically-generated HTML and then transmit the rendered output to the client, rather than the more common method of using client-side JavaScript to retrieve that content, parse it, and insert it into an empty page. The way Pug achieves this is by essentially extending HTML syntax to include loops, functions,

```
{
  completionDate: null | Date;
  orderLines: OrderLine[];
  transactionProcessor: TransactionProcessor | null;
  processorOrder: any | null;
  taxTotal: number;
  shippingTotal: number;
}
```

Figure 13. Comprehensive `Order` schema

```
{
    itemID: ObjectID;
    productID: ObjectID;
    quantity: number;
    costPerUnit: number;
}
```

Figure 14. Comprehensive `OrderLine` schema

variables, and even JavaScript itself. For each request of a page written in Pug, the server will gather the dynamic content needed and pass it to Pug's `render` function along with the page to be rendered. Pug will then process the template file, inserting content as instructed, and generate a string of HTML output which is transmitted to the client. This process of generating a plain HTML file to send to the client is called server-side rendering.

An example of this language is shown in Figure 15, which shows the template for the page of a single session. First, `extends layout` references the "parent" template file, which describes content which is common to all pages on the site, such as the top navigation, footer, basic styles, etc. Then, lines 3–5 append to a block of code named `head` in the `layout` template, adding a page-specific stylesheet and script. Next, `block content` represents the page's main material. If the session passed to this template is downloadable (line 9), include a link to that download. Lastly, for each image in an array of images, add a list item (`li`) to the page which shows the image and links to the products page for that image. Lines 18–20 show the syntax for adding an attribute to an HTML element; in this case, adding `href` to each link to represent where the link should lead.

Extending templates in Pug enables consistent interface design by providing both a singular place to put sitewide design like the navigation bar & footer and separate places to write pages' unique content, without duplicating either.

3.4.2. Less

In standard CSS, styles are applied by using a selector which identifies the element(s) which should receive the styles. Continuing from the example in Figure 15, to style only list items in the unordered list (`ul`) in the `articles.image` element, the selector would be `articles.image > ul > li`. However, to add styles to the links in those list items, one would need to write out the entire selector again, `articles.image > ul > li > a`, and write them as a separate block. Despite the links residing in the list items, these ideas cannot be nested together in the way they are conceptually related.

Less enables, most prominently, nesting related blocks of CSS together. This is shown in Figure 16. When this Less file is compiled to CSS, the ampersands at the beginning of each selector (lines 4, 9, and 12) will be replaced with the parent selector at that point and the blocks will be unnested, resulting in standard CSS. All CSS files are generated when the server is deployed and are static; they do not change from request to request.

```
extends layout

append head
   +addStylesheet("/stylesheets/gallery/session.css")
   +addScript("/scripts/gallery/session.js")

block content
   - const sessionID = session.id.toHexString();
   if session.downloadable
      a(
         href=`/${ adminAsClient ? "admin" : "user" }/sessions/${ sessionID
            }/download`
      ).download Download images (.zip)
   article.images
      ul
         each image in images
            - const imageID = image.id.toHexString();
            li
               a(
                  href=(adminAsClient ? null : `/user/sessions/item/${
                     imageID }?sessionID=${ sessionID }`)
               )
                  +addPicture("/private/sessions/" + imageID)
```

Figure 15. A small example of a Pug template file.

Much like Pug does for HTML, Less eliminates redundant CSS in development by removing the need to write similar selectors over and over again. It also enables simple functions which can be used to generate blocks of CSS based on a few input values, further decreasing the amount of duplication.

```less
article.images {
    margin : 1em auto;

    & > ul {
        padding         : 0;
        list-style-type : none;
        margin-top      : 0;

        & > li {
            text-align : center;

            & > a {
                // link styles go here
            }
        }
    }
}
```

Figure 16. A shortened example of a Less file.

4. Implementation

4.1. Language

TypeScript is a superset of JavaScript which introduces that introduces strong static typing, along with classes, interfaces, generic types, inheritance, access control, and other features related to object-oriented programming [14]. TypeScript was chosen as the language for this project.

JavaScript by itself is dynamically typed, and therefore performs no static type checking. This potentially allows typing-related bugs to go unnoticed until they cause real-world errors. Bunge [15], a software engineer at Airbnb, analyzed six months of bug reports in Airbnb's JavaScript code and discovered that 38 percent of those bugs would have been captured as compile-time errors had the code been written in TypeScript. In another study, an estimated 15 percent of bugs in public GitHub JavaScript repositories were determined to be such that they would have been caught with TypeScript's static typing [16]. Being able to catch more type-related bugs before they cause problems and the enhancements TypeScript provides to JavaScript are a net positive compared to the small amount of time needed to add those types.

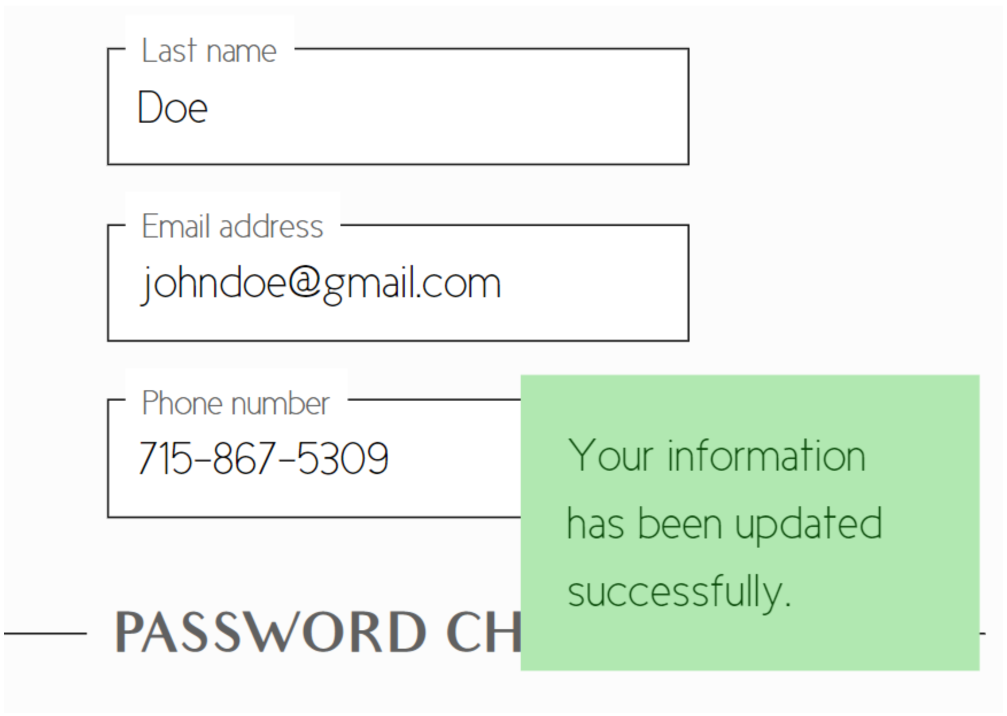
4.2. Request Flow

Both Node.js and Express operate by directing data through a series of operations which transform or output that data. For Node.js, this is manifested by the built-in `Stream` class, which can represent streams of data from and/or to any source, e.g., file IO, an HTTP request, and console IO. Express enhances streaming for HTTP requests specifically by adding support for routing, error handling, and middleware, collectively called handlers. These handlers are chained together, each transforming the request or response, until a full response is formed and transmitted.

All handlers take three arguments: the `Request` that was received, the `Response` being generated, and a `NextFunction` which, when called, triggers the next handler in the pipeline. (Error handlers prepend an `Error` to this signature.) Handlers which share common parts of their URLs are grouped into `Routers`, each group being a class with a name ending in `-Router`. Examples would be an authentication router containing all logic for requests with a path that starts with "/auth", an index router for requests to the root subdomain (specifically, `monicajean.photography` versus `gallery.monicajean.photography`), and a checkout router containing the logic which controls the checkout process, located at "/check-out". In this project, routers which handle an entire subdomain are called `Apps` for ease of reference.

4.2.1. Success and Error Handling

When a route needs to notify the user of the outcome of an action, e.g., "Your profile has been updated" or "'715-867-530' is not a valid phone number", the route will use the NPM package connect-flash to add that message to the current user's HTTP session. Upon the next page load, the message will be "flashed" on screen, colored green for success and red for error, and cleared from the session. Since all requests trigger a page to load, this means that any messages will be shown immediately when they are added. These messages float above the rest of the page's content in the bottom right corner of the viewport and disappear after five seconds. An example of a success message is shown in Figure 17.



Last name: Doe

Email address: johndoe@gmail.com

Phone number: 715-867-5309

Your information has been updated successfully.

PASSWORD CH

Figure 17. An example of a success message. This specific message is shown after a user updates their profile.

If an error is thrown in a handler, and the handler itself does not catch that error, Express will search for the next error handler in the pipeline. If one is not found, the error is logged to the console and the application halts. To ensure that all errors are caught by at least one error handler, the final handler in the entire project is an error handler. Placing this handler at the very end guarantees that it comes after whichever handler threw the error.

Since, almost by definition, the handler which would have been best equipped to handle the error failed to do so, this error handler simply responds to all errors with a 500 HTTP status code (Internal Server Error) and shows a generic error page. This prevents the website from crashing whenever an unexpected issue arises, increasing reliability.

4.2.2. An Example: A Client's Session List

The route handler for this example is shown in Figure 18, the Pug template in Figure 19, and the resulting webpage in Figure 20.

The first line denotes that this function will handle only GET requests to routes that match "/". This is the URL *relative to the router it is mounted in.* Working our way backwards up to the entry point, this route is mounted in `ClientSessionRouter`, which handles the path "/user/sessions"; `ClientSessionRouter` is mounted in `GalleryApp`, which handles all requests to the `gallery.monicajean.photography` subdomain; and `GalleryApp` is mounted in `RootApp`, which handles all requests to `monicajean.photography`. Therefore, a request to `https://gallery.monicajean.photography/user/sessions` will flow through `RootApp`, then into `GalleryApp`, then into `ClientSessionRouter`, then into this handler.

Handlers can also interrupt this flow to handle errors. `GalleryApp` contains a handler which retrieves the `User` for the current request from the database, appends it to the request as `res.locals.user`, and passes the request on. If that handler determined that no one had logged in for this HTTP session yet, it would know that no further routes could possibly handle the request being made, display an error, and redirect the user to the login page.

Assuming no previous handler throws an error, this route handler will be triggered, receiving `req`, a `Request`, and `res`, a `Response`. The `NextFunction` is ignored because this handler will send a response, and is therefore the last one for this request. Since `GalleryApp` must have found the `User` for this request and put it into `req`, that `User` is retrieved from `req`. The HTTP sessions database collection will be searched for `Session`s that have an `_id` in the `User.sessions` array of `ObjectID`s, and have not expired (have either no `expirationDate` or an `expirationDate` later than the current time). Using the sessions collection, the handler will `find` documents matching the query, `sort` them in descending order by `_id` (newest to oldest), `map` each of those raw documents to a `Session` object, and turn the mapped objects to an array. Finally, the Pug template named `sessions.pug` is `render`ed with a `pageTitle` of "My Sessions", and the array of `Session`s, and the resulting HTML is transmitted back to the client browser. A small enhancement made to the `render` function automatically saves any updated `DatabaseObject`s stored in `res.locals` back to the database before rendering the HTML. This eliminates the need to manually save those documents at the end of each route.

The Pug template receives the `pageTitle` and `sessions` values as well as any properties added to `res.locals`, including the current `User`, a nonce value used to securely include scripts and styles, etc. The template can utilize those values in all the same ways a typical JavaScript function could. Here, on line 6 of Figure 19, the content of the page is laid out. For each `Session`, a link is created which leads to that session's page, the link being the cover image of the session. The link also contains a `div` of class `text-overlay`, with the name & date of the session, the number of images in it, and the expiration date.

```
this.router.route("/").get((req: Request, res: Response) => {
   const user: User = res.locals.user;
   const query: FilterQuery<SessionModel> = {
      _id: {
         $in: user.sessions
      },
      $or: [
         { expirationDate: { $gte : new Date() } },
         { expirationDate: { $exists : false } }
      ]
   };

   Session.table.find(query)
      .sort({ _id: -1 })
      .map((sessionModel: SessionModel) => new Session(sessionModel))
      .toArray()
      .then((sessions: Session[]) => {
         res.render("sessions", {
            pageTitle: "My Sessions",
            sessions: sessions
         });
      });
});
```

Figure 18. Route handler which shows the user the list of sessions they can view

4.3. Endpoints

The endpoints in this project are listed below. Some endpoints include text preceded by a colon, such as `/:sessionID`; this indicates that that value is not a literal value, but rather a parameter with that name whose value can be any valid string. For example, the endpoint `/user/sessions/:sessionID` could represent `/user/sessions/5fa9f5614df77f2a90ef0625`, where "5fa9f5614df77f2a90ef0625" is the ID of the session being referenced.

- `/`
  - GET: Redirects to `/auth/login`.
- `/auth`
  - `/login`
    - GET: Show the login page. If a user is already logged in, redirect them the same as a successful POST to this endpoint would.

```
extends layout

append head
   +addStylesheet("/stylesheets/gallery/sessions.css")

block content
   if sessions.length > 0
      section.sessions
         each session in sessions
            a.session(
                    href=`/user/sessions/${session.id.toHexString()}`
            )
               +addPicture(
                  `/private/sessions/${session.coverImage.toHexString()}`,
                  null,
                  "cover-image",
                  "100%"
               )
               div.text-overlay
                  header
                     h3= session.title
                  p
                     | #{session.date.toLocalDateString()}&nbsp;&nbsp;|&nbsp
                       ;
                     | #{session.images.length} image#{(session.images.
                        length !== 1 ? "s" : "")}
                  p Expires #{session.expirationDate.toLocalDateString()}
```

Figure 19. Pug template which renders a list of the client's sessions

  - ∗ POST: Log the user in if the credentials sent in the request are correct and take them to either `/user/sessions` or `/admin/clients`, depending on whether the user is a client or admin, respectively.
- – `/logout`
  - ∗ GET: Log the user out and take them to the login page.
- – `/forgot-password`
  - ∗ GET: Show the page to request a password reset.
  - ∗ POST: Submit a password reset request. The user will receive an email with a link to `/auth/password-reset` to continue.
- – `/password-reset`
  - ∗ GET: If a valid reset token is supplied, show the page for the user to enter a new password.

    - POST: Verify the reset token and check that the new password meets the password requirements, and if so, change the user's password.

- `/user`
  - GET: Redirects to `/user/profile`.
  - `/profile`
    - GET: Show the current user's profile in editable form.
    - POST: Update the current user's profile.
  - `/sessions`
    - GET: Show the client the list of sessions which an administrator has assigned to them. Only sessions which have not yet expired are shown.
    - `/:sessionID`
      - GET: Show the client the images in the session.
      - `/download` GET: If the administrator has enabled downloads for the current session, initiates a download of the full size images in one session, in a zip file.
    - `/item/:itemID`
      - GET: Show a single image, and all the products for sale. Each product has an editable field showing how many of that product are in the client's cart for the current image.
  - `/orders`
    - GET: Show the paginated list of all the current user's orders, filtered by search query, if any.
    - `/:orderID`
      - GET: Show one of the current user's order's details.
  - `/cart`
    - POST: Update the client's cart with the information in the request.

- `/admin`
  - `/clients`
    - GET: Show the client list, filtered by search query, if one was entered.
    - `/new`
      - GET: Show the page to create a new client.
      - POST: Create a new client with the information contained in the request. The client will receive an email with their login information if successful.
    - `/:clientID`
      - GET: Show a client's profile information in an editable form.
      - POST: Update a client's profile.

    - `/sessions`
        - `/new`
            - GET: Show the page to create a new session. The client's ID must be specified in the URL.
            - POST: Create a new session. The images sent will be converted to thumbnails, and upon completion, the client will receive an email that they have a new session to view.
        - `/:sessionID`
            - GET: Show one session's details in an editable form.
            - POST: Update one session's details.
            - `/download` GET: Initiates a download of the full size images in one session, in a zip file.
    - `/products`
        - GET: Show the list of products, filtered by search query, if one was entered.
        - `/new`
            - GET: Show the page to create a new product.
            - POST: Create a new product with the information in the request.
        - `/:productID`
            - GET: Show one product's details in an editable form.
            - POST: Update one product's details.
            - DELETE: Soft delete one product. The product will still be shown in previously-made orders, but will no longer be available to edit or order.
    - `/orders`
        - GET: Show the list of all orders, filtered by search query, if any.
        - `/:orderID`
            - GET: Show one order's details.

- `/checkout`
    - GET: Redirects to `/checkout/cart`.
    - `/cart`
        - GET: Step 1 of the checkout process. Show the client's current cart in editable form.
        - POST: Redirects to `/checkout/payment`.
    - `/payment`
        - GET: Step 2 of the checkout process. If the client has confirmed the contents of their cart, show the page where the client will enter payment information. If they have not (e.g., they directly entered this URL into their browser), redirect them to `/checkout/cart`.
    - `/payment-approved`

        - POST: Receives PayPal's record of the client's payment and shipping address from the client's browser. If the payment is verified by PayPal and the amount matches the expected amount, the client will be redirected to `/checkout/complete`. They will also receive a confirmation email, and the admin will receive an email notifying them that an order has just been completed.
    - `/complete`
        - GET: If the client successfully completed their order on the `/checkout/payment` page, show a confirmation page. If not, redirect them to `/checkout/cart`, where they can begin the checkout process.

- `/private/sessions/:imageID`
    - GET: If the image belongs to a session which the user is allowed to view, loads the image. If the URL specifies an extension (.jpeg or .webp), that type of file will be sent; if no extension is specified, the type will be determined based on the request's `Accept` header.

4.4. `DatabaseObject` and `DatabaseObjectModel` Hierarchies

`DatabaseObject` is the parent class of `User`, `Session`, `Image`, `Product`, and `Order`. Likewise, `DatabaseObjectModel` is the parent interface of `UserModel`, `SessionModel`, `ImageModel`, `ProductModel`, and `OrderModel`. Each of the `DatabaseObject` child classes is a wrapper for the corresponding `-Model` interface, and each corresponds to the collection in the database which stores documents of that type, e.g., a `User` wraps a `UserModel` which would be stored in the users collection. These wrapper classes enable additional functionality which is useful when modifying or accessing documents. The wrappers contain additional properties, akin to SQL computed columns, where the value of that property is not stored in the database, but determined from an expression upon request. An example is the `User` class's `fullName` property, shown in Figure 21.

These wrappers also enable more complex operations like setting a `User`'s password. This is shown in Figure 22. After the route verifies basic input requirements, i.e., did the user enter their previous password correctly and did they enter their new password the same way twice, it will call the `setPassword` function on the `User` which corresponds to the current HTTP request, passing in the user's requested new password. This function also ensures that the old and new passwords are valid and ensures more specifically that the new password satisfies security requirements. Currently, that means that all passwords must be ten characters long and contain one uppercase letter, one lowercase letter, & one number. If it does, the password will be hashed by the `DBConnector` class, and then the hash will be stored on the `passwordHash` property. If not, an `Error` will be thrown back to the route, which can be shown to the user so they understand why the request wasn't successful.

4.5. Security

Security is provided throughout the app by following best security practices and adopting the latest in security-minded web features.

4.5.1. Environment Configuration

This website connects to a number of external resources. The connections to these resources are protected with passwords or API keys and secrets. One often overlooked concern is that these keys are committed to public GitHub repositories, where they are free for anyone to steal or abuse. In a six-month scan of public GitHub repositories, Meli, McNiece, and Reaves [17] found that over one hundred thousand had publicly leaked secrets and that thousands more secrets are leaked every day.

To ensure that the secrets for this project are not published to GitHub, the project stores all keys, secrets, etc. in a file called config.json. This is a simple JSON file placed at the root of the project directory and explicitly excluded from Git via the .gitignore file. Because this file is excluded from Git, updating secrets on the production server requires connecting to the server via SSH and manually inserting them.

The name of the current environment is also included in config.json. This value disambiguates production from development, and is used to provide different debugging output between the two.

This setup supports different environments that can be invoked with little effort. During development, for example, an environment was used to test payment processing against a sandboxed version of PayPal's payments API. By entering the sandbox secrets for the development environment and the "real" secrets for production, it is possible to test payments with identical code in both environments while spending no actual money in development.

4.5.2. HTTPS

Connections to the server are protected using industry-standard HTTPS encryption. Let's Encrypt is an HTTPS certificate authority which issues certificates programmatically and for free. Using Let's Encrypt, all clients' connections to the `monicajean.photography` domain are secured using a 4096-bit asymmetric key. The certificate that Let's Encrypt issues to `monicajean.photography`, as with all other certificates issued by Let's Encrypt, is valid for ninety days. Every twelve hours, the EC2 instance checks whether it should renew the certificate for the next ninety days, and if it can, it does so. When the old certificate is replaced, a file watcher will see the change and update the certificate being used by the website without needing to restart the server or Node process.

Clients attempting to connect with unsecured HTTP will be redirected to the HTTPS version. Additionally, the site utilizes HTTP Strict Transport Security (HSTS). On browsers which support HSTS, the `Strict-Transport-Security` header is used to notify browsers to only ever connect with HTTPS in the future. The `monicajean.photography` domain is also registered in the HSTS preload list, a list shipped as part of all major browsers that prevents *any* unsecured connection from being made to this domain from those browsers [18].

Using HTTPS ensures that user data will not be transmitted in plain text between the server and client, preventing classes of attacks like man-in-the-middle, where an attacker intercepts communications between two computers and is able to monitor or alter them. HSTS optimizes the use of HTTPS by removing the need for browsers to first attempt an HTTP connection only to be redirected to HTTPS.

4.5.3. XSS Mitigation

HTTP Session Cookie The single cookie this site sets is an HTTP session ID. As a cookie, the HTTP session ID is sent on every request to the `monicajean.photography` server, which is looked up in a database table to retrieve the data stored for that HTTP session. This means that session data is never sent to the client.

The cookie includes a number of security features:

- It includes the `secure` flag so it is only ever sent over HTTPS connections.
- It includes the `maxAge` flag so browsers will delete the cookie after two weeks, essentially logging the user out after two weeks of inactivity.
- It includes the `httpOnly` flag so no scripts can access the cookie's value, which prevents malicious scripts from directly accessing the HTTP session ID.
- Its name begins with `__Secure-`. On supporting browsers, this prevents the cookie from being set unless it has the `secure` flag and is set via an HTTPS connection.
- It includes the `SameSite=lax` flag. If a site other than `monicajean.photography` attempts to load any resource from `monicajean.photography`, e.g., private photos, this flag will omit the HTTP session cookie from that request. Since the HTTP session ID cookie is not sent, the server would view the request as coming from an unauthenticated user and prevent any private data from being included in the response.

CORS Another web feature used to secure private data are the Cross-Origin Resource Sharing (CORS) headers. The middleware which makes this possible is shown in Figure 23. Blocking as many requests as possible from sites other than `monicajean.photography` further reduces the avenues by which private data could leak from this site by restricting access to these resources. The middleware is mounted in RootApp and prior to all routes, which ensures that it will be fired on all requests and block applicable requests before they can engage application logic, respectively. The regular expression describes all allowed request origins; essentially, it matches any origin which ends with "monicajean.photography".

CSRF Cross-Site Request Forgery (CSRF) attacks are mitigated through the use of a CSRF prevention token. The token is a random string which is included with every non-safe request, i.e., requests which create, edit, or delete a resource. This token is compared with the original on the server. If it matches, the request is valid; if not, a CSRF attack is in progress, and the session is destroyed (the user is logged out). With this token in place, no third party site will be able to create a valid request because the prevention token is not included.

Pixieset's implementation of CSRF protection has a flaw in that the prevention token is not required on the login form, leaving open the risk of a login CSRF attack. With this flaw, an attacker could create an account with the host site, then trick the victim into unknowingly using the attacker's account on the host site instead of their own. While the victim is unwittingly logged into the attacker's account, they could perform any number of actions, like saving payment card information, uploading pictures, or exposing their clients' email addresses, all directly to the attacker's account [19]. This project enforces CSRF even on the login page to prevent this form of attack.

4.5.4. Content Security Policy (CSP)

A CSP is an HTTP header which specifies which resources a site can use and how it may use them. Supporting browsers can use this data to block requests and halt execution of scripts which are in violation. For example, a CSP could restrict the origin of all resources within the site, so all scripts, images, fonts, etc. could only originate from the same domain as the site itself. It can also block scripts and styles which are defined inline rather than in separate files, which greatly reduces the ability of a malicious actor to inject scripts with an XSS attack.

For this site, the CSP, shown in Figure 24, has been defined using the NPM package Helmet to restrict resources almost entirely to those served from `monicajean.photography` or `gallery.monicajean.photography`. There are a few notable exceptions to this rule, however.

- The site is allowed to connect to and display frames from `paypal.com`, which is needed to handle payments and show PayPal's payment buttons.
- Any `script` tags which are included in the site must include the `nonce` attribute. The nonce is a server-generated, request-specific random string which must match the value specified in the CSP to ensure that the tag came from the server and was not injected by a third party.
- CSS styling can be included inline, without any verification of its source, hence `unsafe-inline`. PayPal's payment buttons do not render correctly without this. PayPal specifies a method to pass a nonce to their script so that `unsafe-inline` is not needed here, but it did not work for an undetermined reason. Ideally, the `style-src` directive would exactly match the `script-src` directive—allow styles from anywhere in this domain, and external ones if they were added with the correct nonce.

Pixieset fails to include any such CSP. Failing to include this means that Pixieset's users are more vulnerable to XSS attacks. If an attacker can get a script tag inserted into a victim's page, that script would have free reign over the contents of the page and could transmit them to a third-party server. If the same attacker inserted a script tag into this project, the script would never be run, since all scripts loaded inline or from outside `monicajean.photography` must specify the nonce which was generated by the server.

4.6. Image Optimizations

The largest page on this site by total file size is the page which shows all the images from a session. Depending on the photographic session being viewed, this page can include hundreds of images, each a few megabytes in size. Under such a large load, page loading time would exceed acceptable limits, cause lower-end devices to slow down, and, in cases where the user is on a cell network, swallow up a costly chunk of mobile data.

4.6.1. Uploads

To reduce the size of the images on the site, two thumbnails are generated from each image uploaded by the administrator during session creation, one WebP format, one JPEG. WebP was chosen as a first choice for its better compression and JPEG was chosen as a fallback for its universal support. When an image is uploaded, these two thumbnails are immediately generated from it. Then, all three versions are uploaded to an S3 bucket. The original is retained so that when a user downloads the images in a session they can have the full-resolution files rather than the scaled-down thumbnails.

To ensure that this optimization is comparable to Pixieset, the file size of the project's thumbnails was compared to that of Pixieset's thumbnails and to the original images. Using 1,315 test images totaling 2.13 gigabytes provided by the sponsor, the originals were converted to JPEG and WebP thumbnails and uploaded to Pixieset. All three had a maximum dimension of 640 pixels. The JPEG thumbnails totaled 160 megabytes, a 92.6 percent reduction. The WebP thumbnails fared even better at just 115 megabytes, a 94.7 percent reduction. Pixieset's JPEG thumbnails totaled 169 megabytes, slightly larger than this project's JPEG thumbnails and almost 50 percent larger than the WebP thumbnails, which Pixieset does not have. This demonstrates that this project succeeds in providing a more efficient image version than Pixieset when provided with the same input.

4.6.2. Viewing

When a client views the images in a session, Pug generates HTML like that in Figure 25 for each image. This HTML instructs the browser to load the WebP image only if it understands how to display such an image. If it doesn't, it will instead load the JPEG specified by the `img` tag. Additionally, `loading="lazy"` instructs supporting browsers not to load this image at all unless it is on screen or nearly on screen, as determined by the browser. This reduces the time needed to load the first few images so the client can begin viewing the list.

The `img` element boasts effectively universal support, having been introduced in 1995 [20]. However, the `picture` and `source` elements are much newer, having only been first conceived of in 2013. Fortunately, all browsers (and HTML itself) are designed to ignore any elements which they do not understand, so as to future-proof them. The result of this behavior is that those browsers will ignore the `picture` and `source` tags and immediately try to load the JPEG image instead. This is important for compatibility; the point of having two thumbnails for each image is to reduce the amount of data being downloaded if possible but while keeping as many people as possible able to use the site.

4.6.3. Caching

All images in all sessions are sent with a `Cache-Control` HTTP header. The `Cache-Control` header tells a client browser and any intermediate servers whether and how to cache a response to more efficiently and quickly respond to later requests for the same content. In this project, all session images are sent with `Cache-Control: private, max-age=31622400, immutable`. This says that only the end browsers may store this image in cache—not intermediate servers—for up to 31,622,400 seconds (366 days). Additionally, this resource is guaranteed never to change because no image will ever have a duplicate ID, so the browser can serve the image from its cache without checking for changes first. Ideally, this means that each browser that views an image will only need to download it one time every year, reducing the load on both the server and client device.

4.6.4. Downloads

When enabled by the administrator, clients can download the images in a session at full resolution. The images files are zipped to speed up downloading.

To ensure that creating ZIP files does not overwhelm the server's storage or memory, the ZIP file is created via a Node.js stream. As each image is downloaded from its S3 bucket, it is piped into an NPM package called Archiver which zips it, and the zipped data is immediately piped to the client. The ZIP file can begin downloading as soon as the first image is zipped; once each image is zipped, it can be garbage collected from the server's memory; and at no point do the session images or ZIP file touch the server's storage. It is possible (and likely) for images to still be getting downloaded to the server even as the client begins downloading to their computer.

4.7. Checkout Process

The easiest way to avoid leaking personal information is to never store it in the first place. Therefore, the only personally identifiable information stored by this project is users' phone numbers, email addresses, and, when they make orders, their physical addresses. Leveraging PayPal's transaction handling means that no payment data could ever be seen by this server.

When a client chooses to begin the checkout process by clicking on the Cart button at the top of any page while logged in, they first must ensure that their cart contains the correct items. The client is also allowed to update anything which might be incorrect or about which they have changed their mind. Shipping, tax, and a grand total are displayed as well. This is shown in Figure 26.

Next, on the payment page, a unique ID created for this business by PayPal is used to retrieve a script from PayPal. This script is executed and exposes a method called `paypal.Buttons` which inserts payment option buttons, seen in Figure 27. Clicking on these buttons opens a pop-up window from PayPal where card information and shipping details will be entered. The Pug template for the script that renders these payment buttons is Figure 28. Figure 29 is an image of the payment window displayed in front of the website.

The `paypal.Buttons` method has one argument, an object with `createOrder` and `onApprove` properties.

`createOrder` The `createOrder` function is called when the user clicks one of the payment buttons. Pug inserts the correct total cost into the function. When executed by the browser, that value is used to create a new PayPal order and return its ID. At this point, a separate PayPal payment window is displayed and the main `monicajean.photography` tab is dimmed by PayPal to redirect user focus onto the new payment window.

`onApprove` Once payment has been approved by the user in the PayPal payment window, the window closes and the `onApprove` function is called. It receives details about the approved order, including details like how much was paid, the shipping address entered, etc. To complete the approval process, the order is `capture`d, essentially approving of the payment from the merchant side. At this point, the transaction is complete. Once captured, all available details are sent to the EC2 server so they may be saved for future reference.

Storing PayPal Order Details Now, the endpoint /checkout/payment-approved receives a POST containing a completed PayPal order. It first re-retrieves the order from the merchant side using the order's ID, to ensure that all available data is stored.

After selecting a payment option, the user will be presented with a popup window from PayPal to enter card information and billing & shipping details. Once payment is completed in that window, the window will close and notify the server that payment has been made via a PayPal order ID. If the server can verify the amount paid on that PayPal order, the site's internal `Order` will be marked completed, the user will be taken to a completion page, and confirmation will be sent via email, thereby completing the transaction and clearing the user's cart.

Outsourcing transaction processing to a company which exists to handle money and has been used for millions of successful transactions already is far easier, far more secure, and far more reliable than creating a custom solution.

4.8. Emails

AWS Simple Email Service (SES) is an email service. SES can be used via the universal Simple Mail Transfer Protocol (SMTP) or the SES-specific API provided by Amazon. The NPM package Nodemailer is used to send outbound emails through SES's SMTP interface. Emails received by the `monicajean.photography` domain, however, are handled by DNS. They are forwarded to the photographer's preexisting Gmail account for convenience. Each `monicajean.photography` address is forwarded to a different plus address. Plus addresses are a Gmail-specific feature which allows users to append a plus sign and a string to the username of their email address while still receiving it at the same account. In this case, `hello@monicajean.photography` is forwarded to `example+mjphello@gmail.com`, which will be received at `example@gmail.com`. The plus address is used to create a filter which sorts emails into folders. Gmail's aliasing feature can also be used to configure `hello@monicajean.photography` as an alias for `example@gmail.com` so that emails sent from `example@gmail.com` will appear as having originated from the alias instead. From the photographer's point of view, emails received to `hello@monicajean.photography` are placed in a folder in her Gmail account and are replied to just like any other email. From a recipient's point of view, emails are being sent to and from `hello@monicajean.photography`, completely masking the `example@gmail.com` address.

4.8.1. Sender Policy Framework (SPF)

SPF has been enabled for all IP addresses which are authorized to send email for `monicajean.photography`. This allows recipients to verify that emails claiming to originate from this domain come from an authorized source. The specifics of the SPF DNS record are discussed below.

4.8.2. DomainKeys Identified Mail (DKIM)

DKIM is a way to verify that an email has come from an authorized source and has not been otherwise altered in transit. DKIM is enforced on an email by first digitally signing it when it is sent. The signature uses a private key to hash the email's body and some/all of its headers. The specific headers which are hashed are up to the receiver, but the `From:` header *must* be included, since it denotes the claimed sender of the email. The signature is added as a new header to the email, and then the email is sent. Then, when the email is received, the receiver uses the DKIM DNS record, which contains the public key corresponding to the private key the email was signed with, to verify that the signature still matches the data which was signed.

For `monicajean.photography`, DKIM can only be configured for emails sent by AWS SES, since Google handles DKIM for all emails sent from Gmail. The DKIM signing process is performed by SES's servers as the email is being sent. The DKIM DNS record which enables this is discussed below.

4.9. Deployment

4.9.1. DNS

Because the `monicajean.photography` domain was purchased through Google Domains, Domain Name System (DNS) configuration is performed through Google Domains' web console, one screen of which is shown in Figure 31. Each DNS record has a name, a type that describes what information it holds, a time to live (TTL) which lists how long other computers should remember the record before re-checking it, and the data.

`A` The `A` record denotes the IP address which is responsible for serving `monicajean.photography`. AWS provides the server with the static IP address 18.232.202.61, so this is the value of this record.

`MX` An `MX`, or mail exchange, record lists the mail server or servers tasked with receiving email to the domain. The project sponsor's preferred means of receiving email was to be able to read and respond to emails from her existing personal Gmail account. Google Domains offers special capability for email forwarding, called address aliasing. To achieve this, aliases were added which map each desired <name>@monicajean.photography address to <personal-email>@gmail.com.

Certificate Issuance The `CAA` record lists the certificate authorities permitted to issue an HTTPS certificate for the domain. Two authorities are listed: letsencrypt.org and pki.goog. Let's Encrypt is the authority in charge of issuing the HTTPS certificate for this domain, while Google must be included to allow emails to be securely sent and received by the project sponsor at @monicajean.photography addresses.

Let's Encrypt offers three types of test which can be used in order for the certificate authority to prove that the entity requesting a certificate actually controls the domain they're requesting it for, called challenges [21]. Additionally, `monicajean.photography` needs a special type of certificate called a wildcard certificate, which secures all subdomains as well as the root domain (specifically `gallery.monicajean.photography`). The only type of challenge which will issue such a certificate involves inserting a randomized string into the DNS records for this domain. The two records with names that start with "_acme" are configured so that the EC2 instance can perform this action automatically.

SPF A type of `TXT` record, called an SPF record, lists all IP addresses which are authorized to send email for `monicajean.photography`. This record includes any IP addresses listed by _spf.google.com and amazonses.com, the SPF records for Gmail and AWS SES respectively. Since no other sources of email exists for this domain, all email originating from other hosts is rejected using `-all`.

DKIM The DKIM signing process is performed by SES's servers as the email is being sent. Since DKIM can only be configured for emails sent by SES and not Gmail, requests for the domain's DKIM DNS record are mapped to SES's servers. To achieve this, SES specifies three `CNAME` records which must be created to achieve this mapping, each containing the "_domainkey" string which signifies that they hold DKIM information.

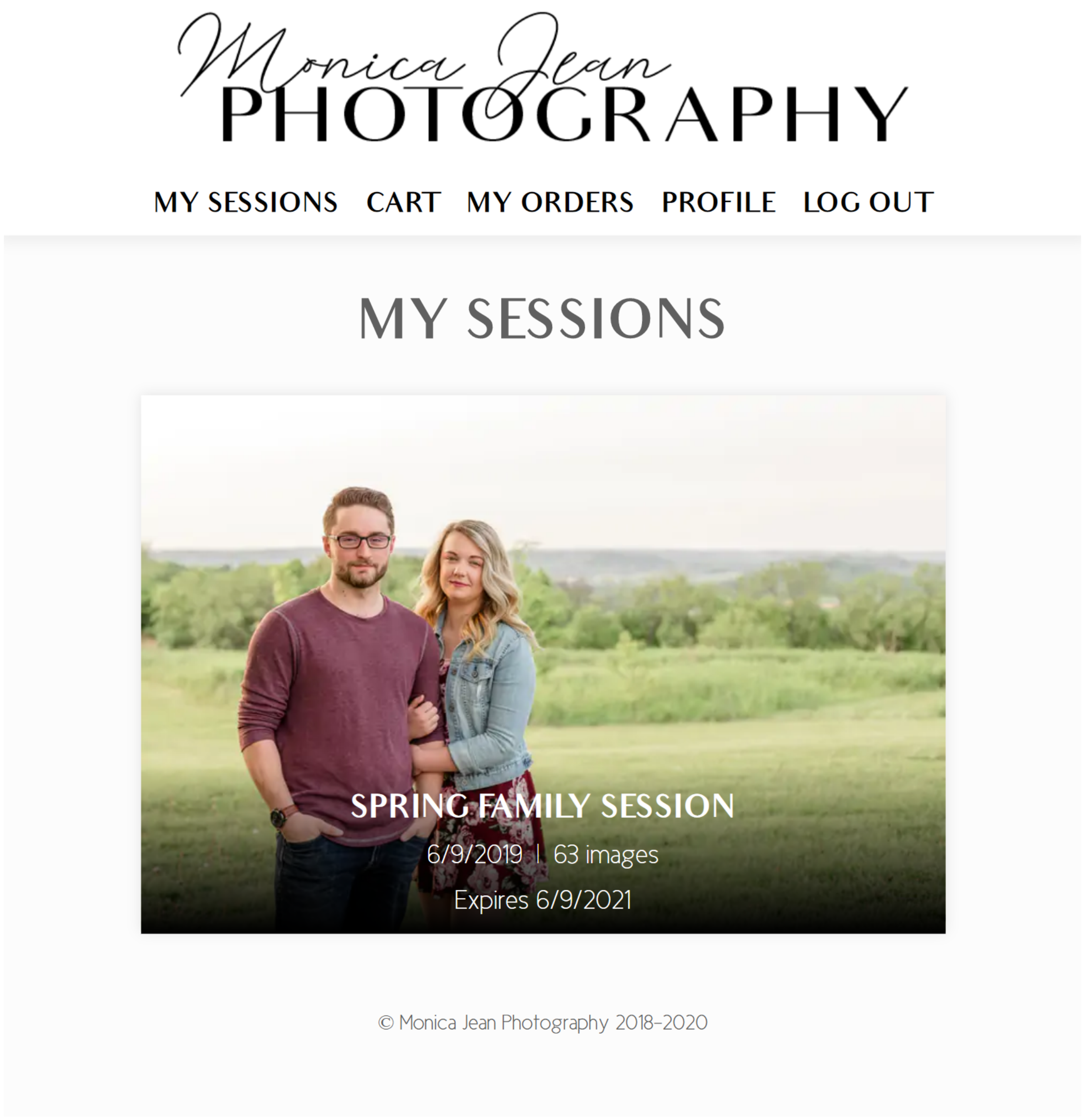


Figure 20. The resulting webpage generated by the code in Figures 18 and 19

```
public get fullName(): string {
   if( this.isABusiness ) {
      return this.givenName;
   }
   else {
      return ( this.givenName + " " + this.familyName ).trim();
   }
}
```

Figure 21. `User.fullName` property accessor

```
public static readonly passwordRegEx: RegExp = new RegExp(/^(?=.*[A-Z])
    (?=.*[a-z])(?=.*\d).{10,}$/);

public async setPassword(newPassword: string): Promise<void> {
   if( User.passwordRegEx.test(newPassword) ) {
      this.model.passwordHash = await DBConnector.hash(newPassword);
      this._updated = true;
   }
   else {
      throw new Error("Passwords must be at least ten characters long and
         have at least one digit, one uppercase letter, & one lowercase
         letter.");
   }
}
```

Figure 22. `User.setPassword` function

```
this.app.use(cors({
   origin : new RegExp(`(?:^|\\.)${ this.app.locals.rootDomain }\\.?$`, "i")
}));
```

Figure 23. This CORS middleware blocks all requests not originating from `monicajean.photography`.

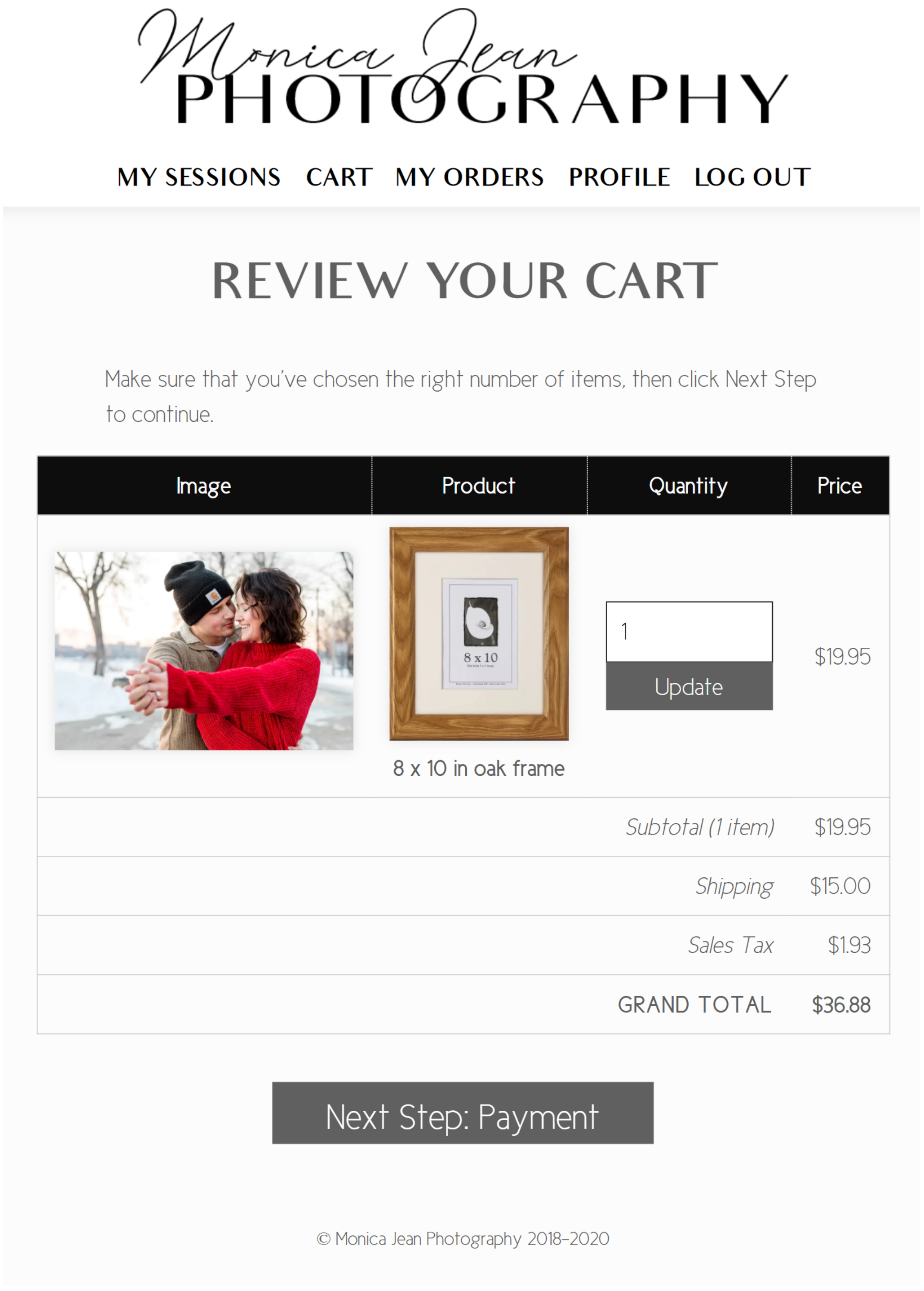
```
const anySubdomainOrRoot: string[] = [ "https://*." + this.app.locals.
   rootDomain, "https://" + this.app.locals.rootDomain ];
const disallow = [ "'none'" ];

this.app.use(helmet({
    contentSecurityPolicy: {
        "default-src": disallow,
      "connect-src": [ ...anySubdomainOrRoot, "https://*.paypal.com" ],
      "font-src": anySubdomainOrRoot,
      "img-src": anySubdomainOrRoot,
      "manifest-src": anySubdomainOrRoot,
      "media-src": anySubdomainOrRoot,
        "object-src": disallow,
      "frame-src": [ "https://paypal.com", "https://*.paypal.com" ],
      "style-src": [ ...anySubdomainOrRoot, "'unsafe-inline'" ],
      "script-src": [ ...anySubdomainOrRoot, (req: Request, res: Response)
         => `'nonce-${ res.locals.inlineNonce }'` ],
      "base-uri": anySubdomainOrRoot,
      "form-action": anySubdomainOrRoot,
      "frame-ancestors": disallow
    }
});
```

Figure 24. The content security policy specification for this site, as it is passed to the Helmet package.

```
<picture>
    <source srcset="/private/sessions/5fb9964fc641f83d3802d667.webp" type="
       image/webp">
    <img src="/private/sessions/5fb9964fc641f83d3802d667.jpeg" loading="lazy
       " width="450">
</picture>
```

Figure 25. HTML generated for each image on the site



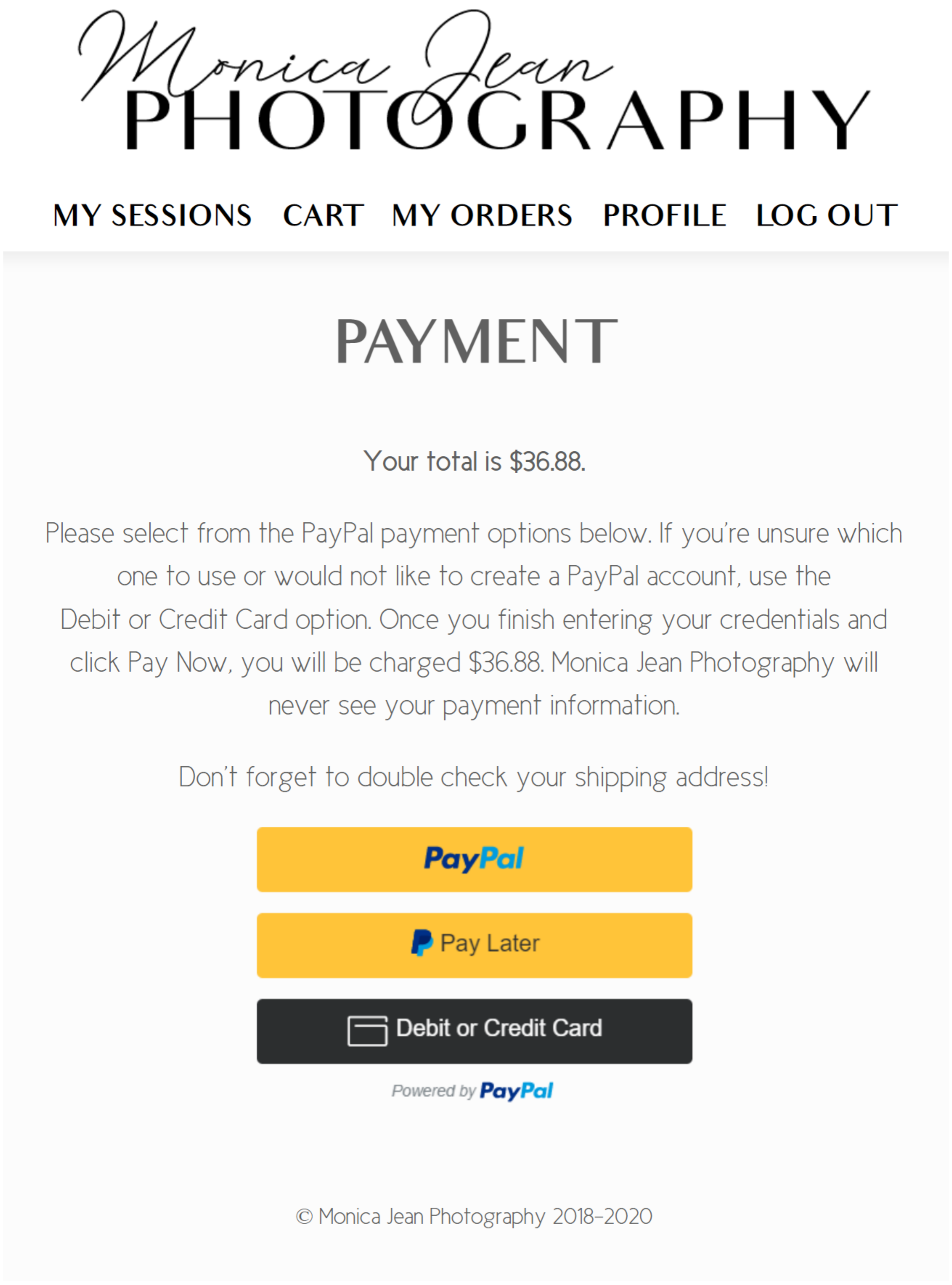
REVIEW YOUR CART

Make sure that you've chosen the right number of items, then click Next Step to continue.

Image	Product	Quantity	Price
	8 x 10 in oak frame	1	$19.95
		Subtotal (1 item)	$19.95
		Shipping	$15.00
		Sales Tax	$1.93
		GRAND TOTAL	$36.88

© Monica Jean Photography 2018-2020

Figure 26. The first step of the checkout process is to review the cart.



PAYMENT

Your total is $36.88.

Please select from the PayPal payment options below. If you're unsure which one to use or would not like to create a PayPal account, use the Debit or Credit Card option. Once you finish entering your credentials and click Pay Now, you will be charged $36.88. Monica Jean Photography will never see your payment information.

Don't forget to double check your shipping address!

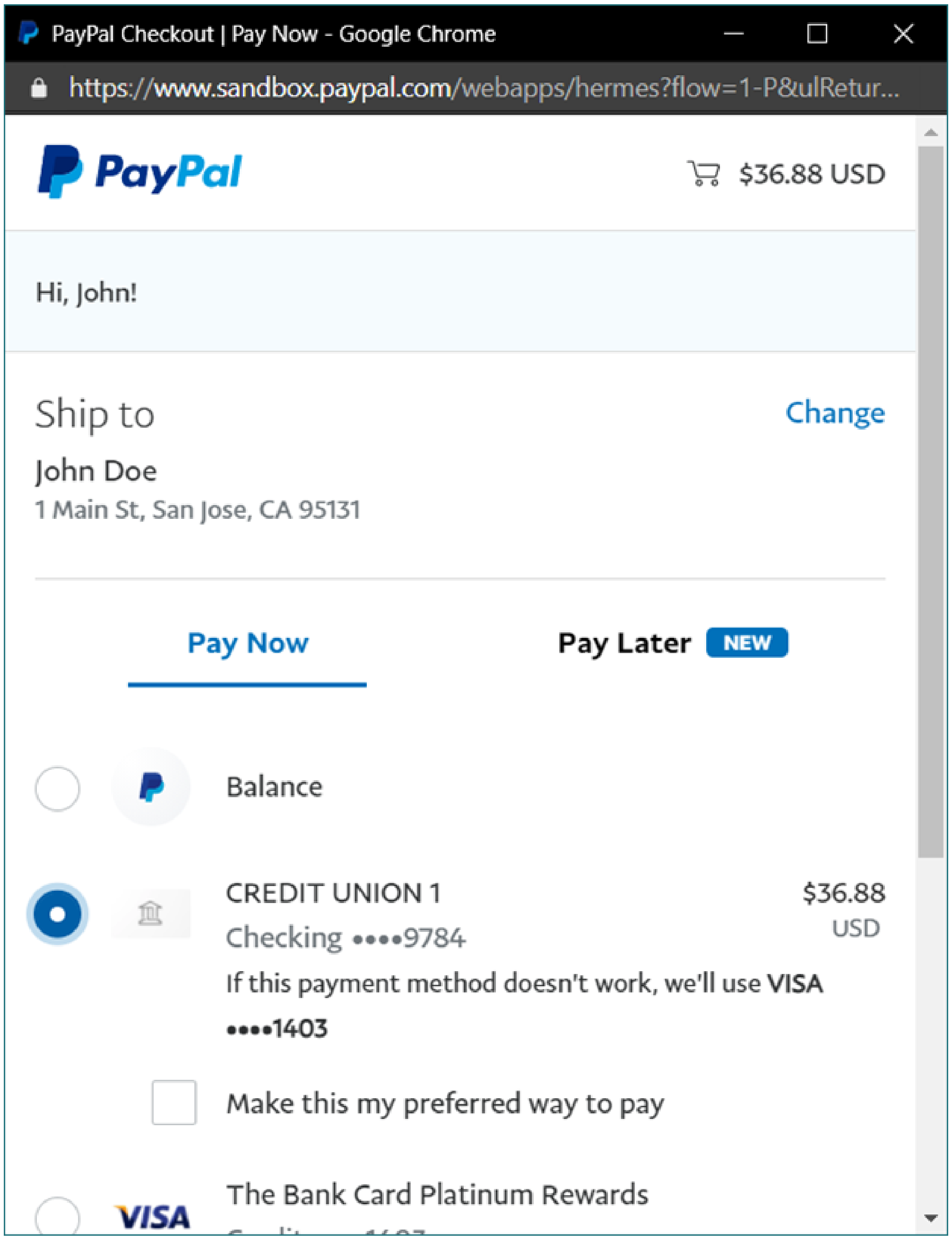
© Monica Jean Photography 2018-2020

Figure 27. The second step of the checkout process is to pay for the items in the cart.

```
paypal.Buttons({
   // createOrder sets up the details of the transaction, including the
      amount and line item details.
   createOrder: function(data, actions) {
      return actions.order.create({
         purchase_units: [
            { amount: { value: "#{order.grandTotal / 100}" } }
         ]
      });
   },
    // onApprove captures the funds from the transaction.
   onApprove: function(data, actions) {
      // show loader
      // code to show spinner omitted

      return actions.order.capture().then(function(details) {
         // save the transaction
         return fetch('/checkout/payment-approved?processor=paypal', {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({
               data,
               details,
               _csrf : "#{csrfToken}"
            })
         }).then(function() {
            window.location.href = "/checkout/complete";
         });
      });
   }
}).render("#paypal-button-container");
```

Figure 28. This script calls a method exposed by PayPal to display payment buttons.


PayPal Checkout | Pay Now - Google Chrome

https://www.sandbox.paypal.com/webapps/hermes?flow=1-P&ulRetur...

Hi, John!

Ship to

John Doe
1 Main St, San Jose, CA 95131

( ) Balance

(●) CREDIT UNION 1 — $36.88 USD
Checking ••••9784
If this payment method doesn't work, we'll use VISA ••••1403

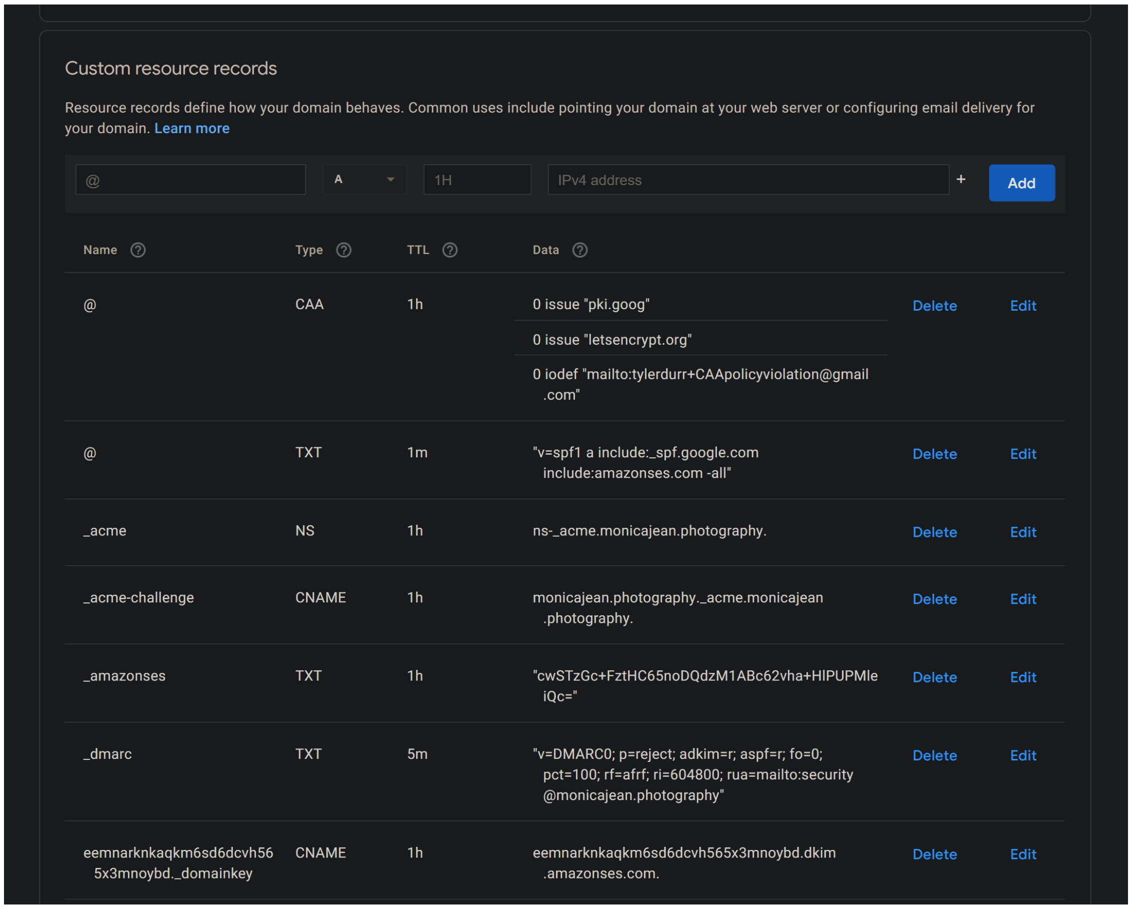
☐ Make this my preferred way to pay

( ) The Bank Card Platinum Rewards

Figure 29. PayPal displays this window after selecting a payment option.

```
this.router.route("/payment-approved").post(async (req: Request, res:
   Response) => {
    const { user, order : ourOrder } = res.locals;
    const { data, details } = req.body;
    // call PayPal to get the transaction details
    const request = new paypalCheckoutSDK.orders.OrdersGetRequest(data.
       orderID);
    try {
       const paypalOrder: any = await PayPalClient.client.execute(request);
       if( Number(paypalOrder.result.purchase_units[ 0 ].amount.value) ===
           ourOrder.grandTotal /
            100 ) {
          await ourOrder.markAsCompleted(TransactionProcessor.PayPal, { data
             , ...details });
          res.locals.ourOrder = ourOrder;
          user.addOrder(ourOrder.id);
          await user.store();
          res.status(204).send();
       }
       else {
          res.status(400).send();
       }
    }
    catch( err ) {
       throw err;
    }
});
```

Figure 30. A just-completed order is updated to reflect the payment.



Custom resource records

Resource records define how your domain behaves. Common uses include pointing your domain at your web server or configuring email delivery for your domain. Learn more

Name	Type	TTL	Data		
@	CAA	1h	0 issue "pki.goog" 0 issue "letsencrypt.org" 0 iodef "mailto:tylerdurr+CAApolicyviolation@gmail .com"	Delete	Edit
@	TXT	1m	"v=spf1 a include:_spf.google.com include:amazonses.com -all"	Delete	Edit
_acme	NS	1h	ns-_acme.monicajean.photography.	Delete	Edit
_acme-challenge	CNAME	1h	monicajean.photography._acme.monicajean .photography.	Delete	Edit
_amazonses	TXT	1h	"cwSTzGc+FztHC65noDQdzM1ABc62vha+HlPUPMle iQc="	Delete	Edit
_dmarc	TXT	5m	"v=DMARC0; p=reject; adkim=r; aspf=r; fo=0; pct=100; rf=afrf; ri=604800; rua=mailto:security @monicajean.photography"	Delete	Edit
eemnarknkaqkm6sd6dcvh56 5x3mnoybd._domainkey	CNAME	1h	eemnarknkaqkm6sd6dcvh565x3mnoybd.dkim .amazonses.com.	Delete	Edit

Figure 31. Google Domains DNS console, showing some of the records which have been configured for `monicajean.photography`

5. Testing

5.1. Overview

For Scrum, testing is part of the iterative process. This project was GUI, unit, integration, and functionally tested.

5.2. Test Data

Previously, the project sponsor discussed testing this website with her real clients. This is called end-user or user acceptance testing. End-user testing ensures that users understand how to use the website and can complete the tasks which the site was meant to facilitate. Unfortunately, out of an abundance of caution, the sponsor has not had any new clients since the beginning of the COVID-19 pandemic, and therefore, there are no clients with whom to test the website.

Even though there were no clients to test the application with, there was client *data* which could be used. The sponsor provided previous clients' images and session information to utilize for testing, a total of 1,315 images. This ensured that the site was tested with a large set of genuine data and would function as expected with future clients' data as well.

5.3. ResponsivelyApp

GUI testing was performed visually, by viewing the various pages of the site. To ensure that the site would render correctly on a range of screen sizes, a special browser called ResponsivelyApp was used. ResponsivelyApp works like any other browser, except that it will display the current webpage as it would appear on multiple devices. The browser comes with a few popular device presets but it also allows you to create your own, including configuring the viewport size, device type (mobile/tablet/desktop), and user agent string, an HTTP header which identifies the browser to the server. Scrolling and page navigation can also be synced between devices. The website was navigated with the various viewports in ResponsivelyApp to ensure that it would work correctly on many device sizes.

5.4. Jest

Jest is a JavaScript testing framework. A package called ts-jest turns Jest into a *Type*Script testing framework. Jest was used for unit and integration testing.

An example of Jest's test files is Figure 33. First, the `User` class is imported and a global variable is declared. The call to `beforeEach` specifies a function which is executed before each individual test in the file. Here, it resets the `emptyUser` variable so that previous tests don't influence subsequent ones. Line 6 specifies that the TypeScript compiler should not type-check the next line, in this case because the empty object would not typically be a

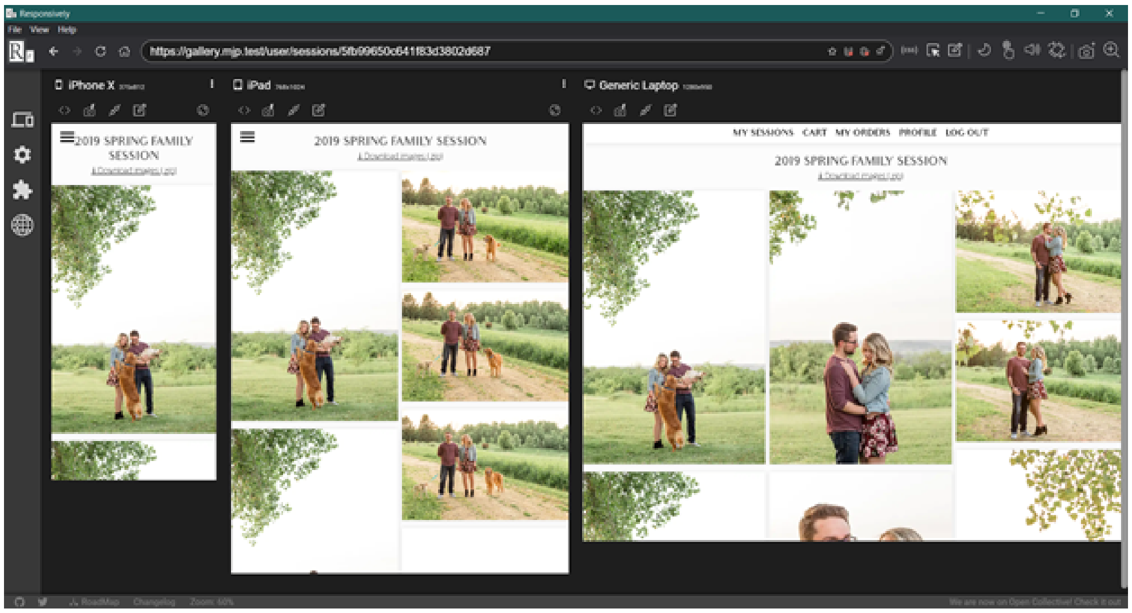

Responsively
File View Help
https://gallery.mjp.test/user/sessions/5fb99650c641f83d3802d687
iPhone X
iPad
Generic Laptop
2019 SPRING FAMILY SESSION
MY SESSIONS CART MY ORDERS PROFILE LOG OUT

Figure 32. Visually testing a session view with ResponsivelyApp

valid argument for the `User` constructor. Lines 12 and 13 `describe` what `it` (the code being tested) should do with strings that can be parsed and displayed by the command line or IDE. Line 14 uses Jest's global `expect` object to declare that exactly one assertion will be made by this test. Lines 16 through 18 are the assertion: Trying to set `user.phoneNumber` to a value which is not a phone number is expected to throw an error.

If all assertions in a test are satisfied, the test passes. Otherwise, it fails, printing the expectation and the value which failed to meet that expectation. Test output can be viewed on the command line, through an IDE, or even as an exported HTML file, shown in Figure 34.

5.4.1. Mocking

Sometimes, a function or class needs to be isolated from external dependencies for testing. The `Email` class is an example of this. Testing this class by allowing it to send real emails would not only be irritating, but fragile—if the internet connection were out, email tests would fail even though the code was working as intended.

For these situations, Jest can "mock", or imitate, dependencies. `Email` depends on an NPM package called Nodemailer to transmit emails. While the `Email` class is being tested, Nodemailer is effectively substituted with a no-op. This substitution is opaque to `Email`, so the test will complete as normal, but without having actually sent an email.

```
import { User } from "../../src/DatabaseObjects/User";

let emptyUser: User;

beforeEach(() => {
   // @ts-ignore: invalid UserModel
   emptyUser = new User({});
});

// other tests...

describe("phoneNumber setter", () => {
   it("fails on an invalid number", () => {
      expect.assertions(1);

      expect(() => {
         emptyUser.phoneNumber = "not-a-number";
      }).toThrowError();
   });
});
```

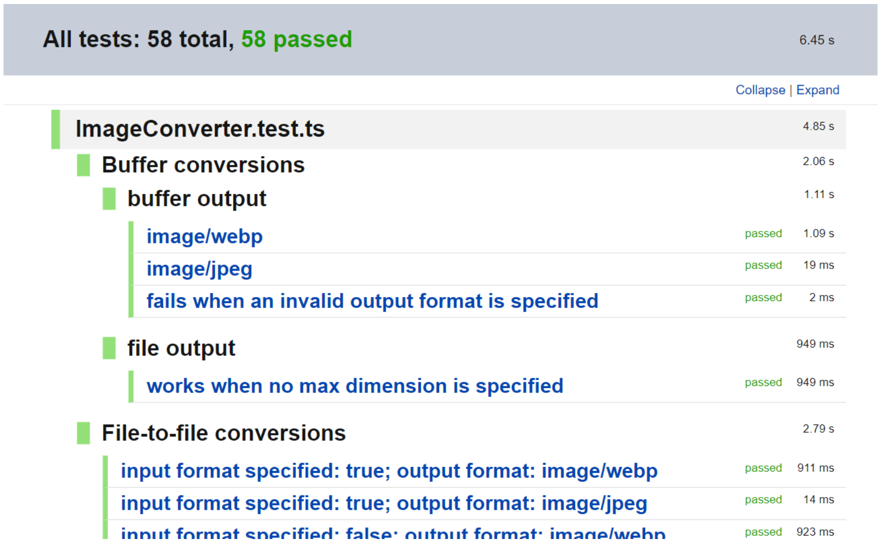
Figure 33. Jest `User.phoneNumber` setter test

5.5. Functional and Acceptance Testing

Functional testing was performed by using the website as a user would, creating accounts, adding sessions, making purchases, etc. Erroneous inputs for forms were tested to ensure that the form would display an error message, but not crash or save invalid data to the database.

A separate database and S3 bucket were made for local testing. The developer's laptop was configured to internally redirect the domain `mjp.test` to a locally-run server, and a new HTTPS root certificate was created and added to the operating system's trusted certificates so that the `mjp.test` domain could be correctly served over HTTPS. Performing this configuration ensured that a maximum of code being tested would be shared between the development and production environments, while keeping interactions between the environments at zero.

Acceptance testing occurred frequently during the project. By demonstrating new features at weekly scrums and sprint reviews, critical immediate feedback was gained. Such feedback was important in ensuring that the project was moving down the path the sponsor wanted, rather than having significant amounts of time be wasted on features or functionality which doesn't meet expectations.



```
All tests: 58 total, 58 passed                                                   6.45 s
                                                                      Collapse | Expand
ImageConverter.test.ts                                                           4.85 s
  Buffer conversions                                                             2.06 s
    buffer output                                                                1.11 s
      image/webp                                                     passed      1.09 s
      image/jpeg                                                     passed      19 ms
      fails when an invalid output format is specified               passed      2 ms
    file output                                                                  949 ms
      works when no max dimension is specified                       passed      949 ms
  File-to-file conversions                                                       2.79 s
    input format specified: true; output format: image/webp          passed      911 ms
    input format specified: true; output format: image/jpeg          passed      14 ms
    input format specified: false; output format: image/webp         passed      923 ms
```

Figure 34. Jest HTML test output

6. Conclusion

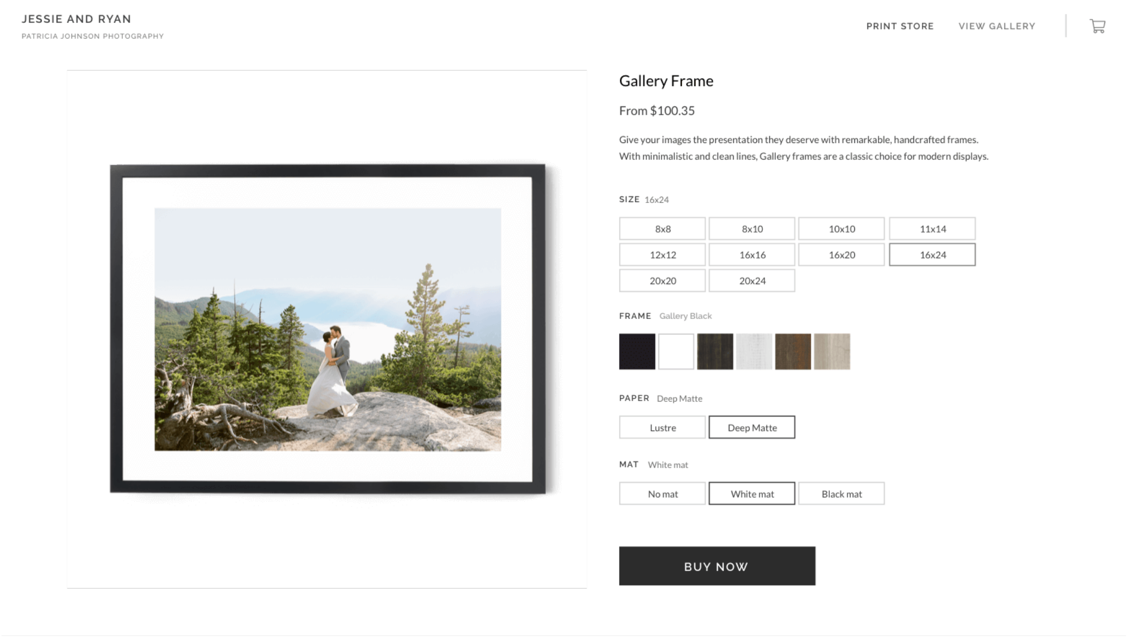
6.1. Overview

The project has succeeded in achieving its overall goal of creating an alternative to Pixieset which has reduced functionality at a reduced price, while maintaining security, efficiency, and reliability.

6.2. Comparison

In comparison to the free features Pixieset lists in Figure 1, this project offers all the features which Pixieset notes, except for coupons and gift cards. Storage limits and commission fees are eliminated, though PayPal still charges a transaction fee. It also supports the three features not included in the free version, “Connect your domain”, “Remove Pixieset branding”, and “Add custom logo & branding”, as intrinsic parts of having engineered a custom-made solution.

Reduced functionality is most clear on the page where items are added to the user’s cart. Pixieset’s item page in Figure 35 is configured such that a wide variety of sizes, frames, and papers can be chosen from a single page. In contrast, there is no way to configure these product features in this project, shown in Figure 36. To do this, a separate product would need to be entered for each configuration of those features.



Gallery Frame

From $100.35

Give your images the presentation they deserve with remarkable, handcrafted frames. With minimalistic and clean lines, Gallery frames are a classic choice for modern displays.

SIZE 16x24

8x8 | 8x10 | 10x10 | 11x14 | 12x12 | 16x16 | 16x20 | [✓ 16x24] | 20x20 | 20x24

FRAME Gallery Black

PAPER Deep Matte

Lustre | [✓ Deep Matte]

MAT White mat

No mat | [✓ White mat] | Black mat

Figure 35. Pixieset offers many options for the kinds of products that can be purchased.

Measures of a website’s overall security necessarily involve a level of subjectivity. Though it might be possible to focus on a single security feature and objectively determine whether it is or is not implemented, combining multiple features introduces questions about how to weight those features. To minimize the developer's potential biases in comparing the security of Pixieset and this project, the Mozilla Observatory was used to test security. Created by the Mozilla Foundation and used on over 170,000 sites, the site "tests for preventative [measures] against cross-site scripting attacks, man-in-the-middle attacks, cross-domain information leakage, cookie compromise, content delivery network compromise, and improperly issued certificates" [22].

On a scale from 0/100 to 135/100, this project scored a 120/100. Pixieset, on the other hand, scored 0/100 for missing basic features like cookies with the `Secure` attribute, implementing HSTS, and failing to implement a content security policy, all features this project has. Though no site can paint a complete picture of a site's security, Mozilla Observatory makes it clear that this site provides at *least* equivalent security to Pixieset.

The site is reliable in part because of the reliability of AWS. The 99.99 percent uptime guarantee of EC2 has been met and exceeded [23]. Since migrating to the platform over a year ago, only four minutes of downtime have been attributable to EC2 itself.

AWS's free tier offers twelve free months to new customers for select services with relevant limitations on storage, server type, etc. Because this website is able to fit entirely within these limitations, the only costs incurred by the first year of this project's operation were for the domain monicajean.photography, which is $20 per year, and per-transaction fees charged by PayPal. Due to the COVID-19 pandemic, there have been no new customers with which to test this website, and so no transactions—or transaction fees—were created. Therefore, and with the clear understanding that this is atypical, the first year of operation cost a grand total of just $20.

After the first year, costs rose to the standard AWS pricing rates. Storage costs became $0.0125 per month, server costs became $0.0094 per hour (average of $6.87 per month) [9], [24]. Database hosting with MongoDB Atlas is effectively permanently free, with an estimated 1.5 million image records capable of fitting within the 512 MB limit of the free tier [25]. Even in a worst-case estimate, this solution will remain more cost-effective than Pixieset until the number of images stored reaches approximately 750,000. Future optimizations to the way images are stored and cached could make this even more cost-effective.

6.3. Problems and Complications

Generally, issues encountered in the software engineering process stemmed from poor personal time management. At the outset of the independent study which started this project over a year ago, the intended completion date for the project was May 2020. Communication between the developer and project advisor, deadlines & goals, and the creation of this paper were each hampered by problems which could have been avoided with better time management; respectively, weekly scrums were not always held as intended, the expected completion date for this project was moved by over half a year, and paper creation took far longer than necessary. Even within individual sprints, work lagged until it became absolutely necessary to complete it (see Figure 4). Improving technique and skill in this area will be critical to the outcome of future projects, career outlook, and personal satisfaction.

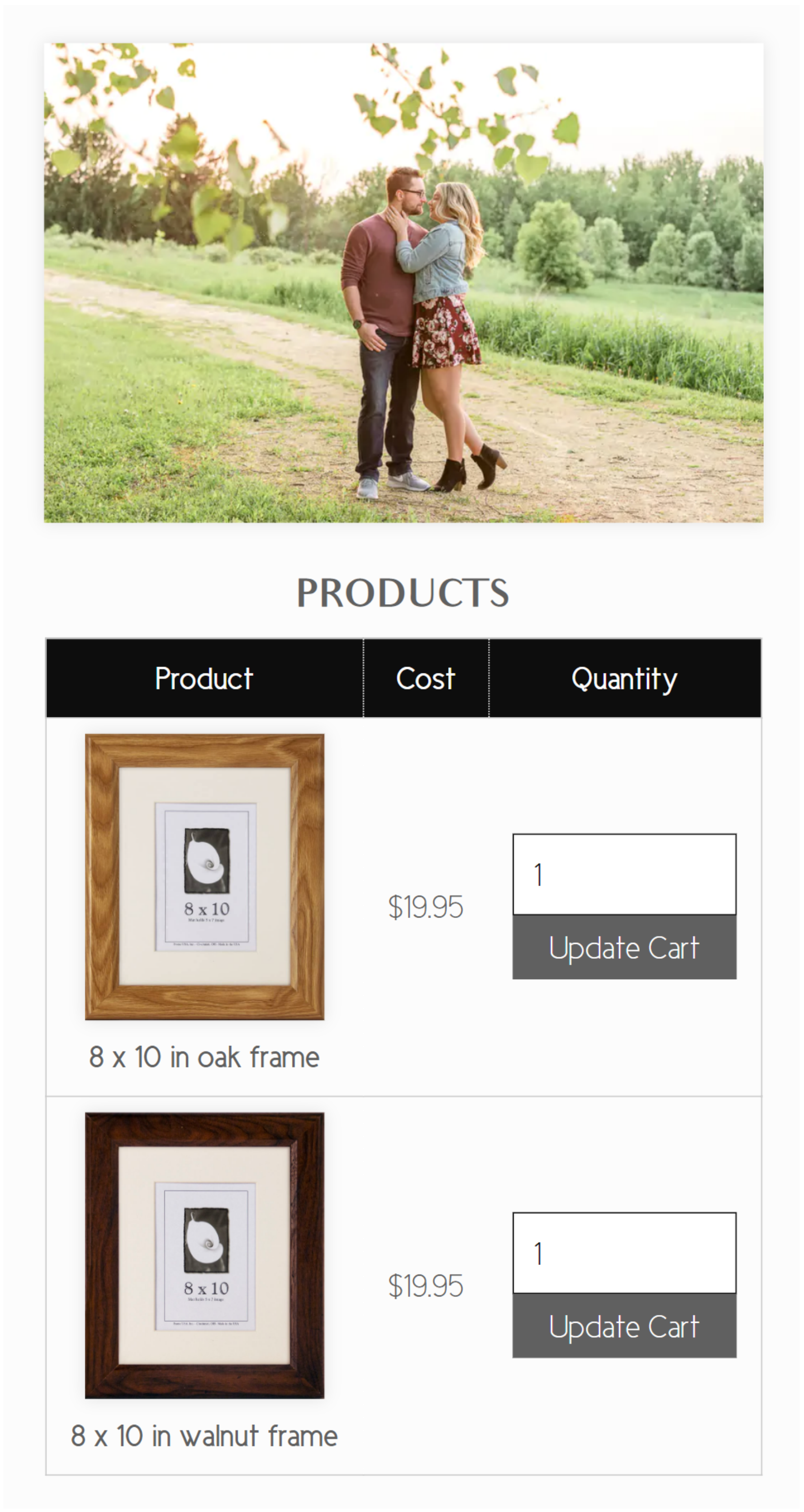
A temptation encountered throughout the project was to make changes without adhering to the process of making a backlog item and then waiting for the next sprint to implement it. Regrettably, this did happen during the project. Password resets were implemented as part of a backlog item for automatic password generation, for example. Unplanned work such as this potentially slowed the project as a whole by invading on time which could otherwise have been dedicated to features which were actively included in the backlog. This is, perhaps, a pitfall of adopting Scrum on a solo project; in an industry application of Scrum, a person called a Scrum master would be tasked with supporting the Scrum methodology and avoiding issues such as this.

6.4. Future Improvements

Currently, shipping is a flat rate of $15 for any shipment. There are no options for shipping type. The US Postal Service provides APIs which calculate shipping cost and track packages throughout their shipment. These sorts of features are common among sites which make shipments to their customers. Therefore, adding calculated shipping and package tracking make good candidates for the next enhancements to be made.

One way to reduce the load on the server would be to send only the data needed to render a page's HTML and then have a client-side script do so, rather than the current system of having the server render the HTML itself. This would require the creating of an applications programming interface, or API, as well as the client-side script to make these requests and display the responses correctly. Such an API would be almost trivial to implement. The HTML renderer, Pug, already receives an object containing all the data which is needed to render a page. The API would simply bypass Pug and send this data instead. While This moves the burden of generating the HTML from the server to the client. A number of frameworks exist for client-side rendering as well, such as Angular, Vue.js, and Svelte.

S3 costs could be reduced by moving old or expired sessions into cheaper storage tiers. S3 has multiple tiers for storage, each of which offers a different balance between cost and access speed. By choosing slower access speeds for images in sessions which have already expired, they would cost less to store.



PRODUCTS

Product	Cost	Quantity
8 x 10 in oak frame	$19.95	1 Update Cart
8 x 10 in walnut frame	$19.95	1 Update Cart

Figure 36. This project does not have configurable products like Pixieset's.

7. Bibliography

[1] R. Rabbat. (Jun. 29, 2020). WebP, a new image format for the Web, [Online]. Available: https://blog.chromium.org/2010/09/webp-new-image-format-for-web.html.

[2] Pixieset. (2020). Pixieset - Client photo gallery for modern photographers., [Online]. Available: https://pixieset.com.

[3] ——, (2020). Pixieset - Pricing for Client Gallery, [Online]. Available: https://pixieset.com/pricing/.

[4] Digital.ai Software, Inc., "The 14th Annual State of Agile Report," Digital.ai Software, Inc., technical report, May 26, 2020. [Online]. Available: https://stateofagile.com/#ufh-i-615706098-14th-annual-state-of-agile-report/7027494.

[5] K. Schwaber and J. Sutherland, *The Scrum Guide, The Definitive Guide to Scrum: The Rules of the Game*, Nov. 2017. [Online]. Available: https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf.

[6] Agile Alliance. (Jul. 9, 2020). Daily Meeting, [Online]. Available: https://www.agilealliance.org/glossary/daily-meeting/.

[7] M. Rehkopf. (2019). What is a kanban board? [Online]. Available: https://www.atlassian.com/agile/kanban/boards.

[8] U. Eriksson. (Apr. 5, 2012). Why is the difference between functional and Non-functional requirements important? [Online]. Available: https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/.

[9] Amazon Web Services, Inc. (2020). Amazon S3 Pricing, [Online]. Available: https://aws.amazon.com/s3/pricing/.

[10] solid IT. (Oct. 2020). DB-Engines Ranking. Wayback Machine: https://web.archive.org/web/20201027/https://db-engines.com/en/ranking, [Online]. Available: https://db-engines.com/en/ranking (visited on 10/27/2020).

[11] B. Shneiderman. (2016). The Eight Golden Rules of Interface Design, [Online]. Available: https://www.cs.umd.edu/users/ben/goldenrules.html.

[12] J. Nielsen. (Apr. 24, 1994). 10 Usability Heuristics for User Interface Design, [Online]. Available: https://www.nngroup.com/articles/ten-usability-heuristics/.

[13] B. Tognazzini. (Mar. 5, 2014). First Principles of Interaction Design (Revised & Expanded), [Online]. Available: https://asktog.com/atc/principles-of-interaction-design/.

[14] O. Therox, E. Barzilay, and K. Nagae. (2020). The TypeScript Handbook, About this Handbook, [Online]. Available: https://www.typescriptlang.org/docs/handbook/intro.html (visited on 10/29/2020).

[15] B. Bunge, "Adopting TypeScript at Scale," presented at the JSConf Hawai'i (Waikiki Beach Marriott Resort, Feb. 7–8, 2020), Feb. 7, 2020. [Online]. Available: https://www.youtube.com/watch?v=P-J9Eg7hJwE.

[16] Z. Gao, C. Bird, and E. T. Barr, “To Type or Not to Type, Quantifying Detectable Bugs in JavaScript,” in *Proceedings of the 39th International Conference on Software Engineering*, May 20, 2017, pp. 758–769. DOI: `10.1109/ICSE.2017.75`.

[17] M. Meli, M. R. McNiece, and B. Reaves, “How Bad Can It Git? Characterizing Secret Leakage in Public GitHub Repositories,” presented at the Network and Distributed Systems Security Symposium 2019 (Feb. 24–27, 2019), Internet Society, Feb. 26, 2019. DOI: `10.14722/ndss.2019.23418`. [Online]. Available: `https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_04B-3_Meli_paper.pdf`.

[18] The Chromium Projects. (2017). HTTP Strict Transport Security, [Online]. Available: `https://www.chromium.org/hsts`.

[19] CheatSheet Series Team. (Nov. 9, 2020). Cross-Site Request Forgery Prevention Cheat Sheet, [Online]. Available: `https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html`.

[20] T. Berners-Lee, “Hypertext Markup Language - 2.0,” RFC Editor, RFC 1866, Nov. 1995. [Online]. Available: `https://www.rfc-editor.org/rfc/rfc1866.txt`.

[21] Let’s Encrypt. (Feb. 24, 2020). Challenge Types, [Online]. Available: `https://letsencrypt.org/docs/challenge-types/`.

[22] Mozilla Foundation. (). Mozilla Observatory, [Online]. Available: `https://observatory.mozilla.org/`.

[23] Amazon Web Services, Inc. (Jul. 22, 2020). Amazon Compute Service Level Agreement, [Online]. Available: `https://aws.amazon.com/compute/sla/`.

[24] ——, (2020). Amazon EC2 On-Demand Pricing, [Online]. Available: `https://aws.amazon.com/ec2/pricing/on-demand/`.

[25] MongoDB, Inc. (2020). MongoDB Pricing, [Online]. Available: `https://www.mongodb.com/pricing`.