Project/Team Name - TigerMart (working name)
Project Leader - Marc Fishman (mfishman@)
Group Members - Ines Franch (ifranch@), Devansh Gupta (devanshg@), Pablo Gutierrez (pablogp@), Ryan Hammarskjold (ryanh@)

SECTION 1: OVERVIEW

TigerMart is a web-based application that works as an online marketplace for Princeton students and professors. It provides an online platform where students and professors can buy/sell from/to other members of the Princeton community anything from textbooks and clothes, to bikes and office chairs.

Our goal is optimize and bring together a variety of resources currently used for these purposes (Tiger Trade, Free & for Sale, Princeton Textbook Exchange) into one single efficient, simple and well designed website that encompasses the entire Princeton marketplace.

Users will be able to login using their Princeton netid and post their items up for sale, as well as look at and search for items other people have posted. When the user finds an item they are interested in, the website will make it possible to look at the seller's profile, contact them, arrange a time and place to meet, and even pay them online!

The web application will be built using MongoDB, Node.js, Express, and AngularJS (MEAN stack). Heroku will be used to host the website and GitHub for the repository. Payments will be implemented using Stripe or Venmo.

SECTION 2: REQUIREMENTS AND TARGET AUDIENCES

TigerMart will solve the inadequacies of the current methods used to buy/sell items within the Princeton community. Tiger Trade and Princeton Textbook Exchange, both USG apps used for similar purposes, fail as a result of their antiquated and obsolete design. The user experience is severely damaged by the aesthetically unpleasing design and the limited functionality of these websites, and that translates to low usage within the Princeton community. Free & for Sale, by far the most popular method used for this purpose, thrives from its simplicity and convenience. However, because Facebook is not designed with the intention of hosting online marketplaces, Free & for Sale lacks many desirable features, like efficient searching and direct payments, that an optimal website would have.

The intended users of our web application are Princeton students and professors in general. The many benefits of having a Princeton-specific online marketplace, chief among them the convenience of buyers and sellers living within a very small radius, are validated by the popularity of existing solutions like Free & for Sale. TigerMart, however, will provide additional benefits, not only because it will solve the inefficiencies of the existing solutions, but also because by doing so it will eliminate the fragmentation that results from having a variety of solutions for the same problem. While this might be desirable in some cases, it is clear that when talking about an online marketplace, users benefit from a single platform that brings together all of the buyers and sellers in the community.

SECTION 3: FUNCTIONALITY

The principal functionality of our project is to allow users to buy and sell items posted on the site.

1. Selling an item

When users want to sell an item that they own, they will create a new post on the site by clicking “Sell New Item.” This will prompt a simple form to be filled out with information about the item. This information will be used to categorize the item and make it easily searchable for a user looking to buy it. To that end, we will have multiple categories of items, of which users can select one to classify their item:

- Textbook
- Apparel
- Electronics
- Furniture
- Tickets
- Dorm Items
- Food & Drink
- Transportation

After the seller selects one of these categories, he or she will also be able to label the item with self-created “tags”; sellers can simply type in key terms that describe their item. These tags will help users to find items more easily. The seller will also provide a selling price for the item. We are not including a bidding system for our site; instead, sellers will be able to indicate if they are willing to negotiate on their selling price. Finally, the seller will add a picture of the item for an enhanced searching experience. For textbook sales, the seller will indicate which course the textbook is for and the condition of the book. For apparel sales, the seller will indicate the newness of the item (e.g., brand new, lightly worn, etc.)

2. Buying an item

All of the information provided by the sellers allows buyers to search for items with relative ease. We will provide options for a user to filter their search by category, by price range, and by the date of the post (e.g., newest to oldest). Beyond this, we will have a search bar in which users can further specify the type of item that they are looking for. Items whose name or whose tags match the search query will appear on the screen during a search. For textbook searches, we will have a specific search feature that builds upon Assignment 4 from this course. That is, when searching for a textbook, users will be able to search for their courses by department, course number, distribution requirement, professor, etc.

The browsing page will provide the item’s name, picture, and price. When buyers click on an item that interests them, they will be presented with additional information (e.g. a description of the item, the seller’s name and netID, etc.). If a user has decided to purchase the item, he or she can email the seller by clicking on the provided netID. Then, the buyer and seller can communicate about payment and about transferring the item from seller to buyer.

We may include other methods of contact, such as a comment thread on the item, or perhaps even an instant-messaging system. We may also provide the ability to pay for items through an app like Venmo or Stripe.

3. User profiles

A user will be able to view the history of items that he or she has bought and sold on the site. Items will be marked “Sold” if they have been sold on the site, “Bought” if they have been purchased by someone on the site, or “Active” if the user is still looking to sell the item. In addition, there will be an “Email & Alert Settings” section for each profile where users can customize how they receive notifications. For example, if we decide to implement the comment thread feature, then users can select whether they want to be notified every time someone comments on their items.

4. Editing a post

Sellers can edit an existing post for an item that has not yet been sold. This allows sellers to change an item’s price, add a new picture, or add more information about the item as they see fit. To do this, they will visit their profile, click on an item labeled “Active” and click “Edit Post.” An edit to an existing post will update the date of the post to the date of the most recent edit.

5. Marking an item as sold

When a transaction is completed, the seller will have to mark the item as sold so that no one else will try to buy it. After a purchase of an item, the item will be removed from searches. On the seller’s profile, the item will be changed from “Active” to “Sold.” On the buyer’s profile, the item will be added to the item history and labeled “Bought.”

SECTION 4: DESIGN

We are going to build a web application with the MEAN stack (MongoDB, Express, AngularJS and Node.js). We will have a three-tier system with a front-end tier (AngularJS), a logic tier (Node.js and Express) and a data tier(MongoDB). The code will all be written in Javascript. We will be using Git and a GitHub repository for version control. The website will most likely be hosted on Heroku. We might implement payments through the app using Stripe or Venmo.

Web Application Front-End

We will be using AngularJS and Bootstrap to build the front-end. We will first authenticate users using CAS, since we are limiting our product to the Princeton University community. The interface will be implementing two different functions: one for sellers to list new products, and another one for buyers to search for existing products. For the first one, we will provide a form for a seller to fill out, and we will be saving those fields as well as the user information from CAS to the database. Buyers will see the available products along with their pictures. They will be able to organize them by category and also to sort them by date posted or price. We will also be implementing a search function, where we will query against the description field of the documents in the MongoDB collection. Items can be removed from the database by the seller, since we might not know if the transaction has happened if it is paid in cash. We will also have an option for a seller to see their profile and all the products he or she has listed, where they will be able to edit or delete the products.

We have different options for communication between the seller and the buyer. The first one is just listing contact information for each user, and allowing them to communicate outside our platform. Another option would be having a comments section for each product (which would be another field for the document in MongoDB) where the buyer and seller would communicate. Finally, the last option would be having a built-in direct messaging function.

Web Application Back-End and Logic

We will be using Node.js and the Express web application framework. The back-end will have the following functions: storing the data for a new product given by the user to the database, interpreting a query and returning and formatting the matching results from the database, sorting the products given a certain option, listing all the products a seller has posted, editing the information for a given product, deleting a product and marking a product as sold. We will use Express to respond to HTTP requests.

We may be integrating a payment option, probably through Stripe or Venmo. Users will still have the option to complete the transaction in cash, in which case the seller would be the one to mark the product as sold.

We will also implement the logic for the user profile section, where a user will be able to see and change their email and alert settings (to get notified when their product is sold, for example) and also see and edit their posts.

The information passing through the different tiers will be in JSON format, specifically BSON which is what MongoDB uses. We will store the documents in this format, write JSON queries on the server and pass the JSON documents received to the front-end.

We will use gulp or grunt as a task runner to automate the development process.

Data Storage

We will be using MongoDB, a NoSQL database. We will use Mongoose.js for adding structure to MongoDB. MongoDB doesn’t have tables but collections of documents, and we will have two collections. We will have a collection of all the products for sale. The seller will provide most of the data. The schema will be the following:

- _id: primary key, the user won’t provide this
- name: name of the product
- description: short description of the product
- extra: section for the seller to provide extra information, optional
- category: category of the item (textbooks, apparel, electronics, etc. - see Section 3)

- tags: key terms that describe the item
- date: date when the product was posted
- price: amount in dollars (it is fixed, there is no bidding system)
- sellerid: netID of the seller, given by CAS
- picture: possibly more than one picture (we might have to use GridFS if we think the size might exceed 16 MB)

We will have another collection to save the data of all the buyers and sellers. We will have the following schema:

- _id: primary key
- fname: first name
- lname: last name
- netid: Princeton NetID, given by CAS
- email: Princeton email, of the format *@princeton.edu
- preferences: alert and email preferences, format to be decided

We will probably also have another collection to save chats, messages or comments, depending on the kind of communication system that we end up deciding to implement.

SECTION 5: TIMELINE

S. No.	Milestones	Due Date
1	Learn how to use Git/Github for Version Control Learn how to use MEAN Stack in general Individually learn (in detail) how to use assigned part of stack (MongoDB/Express/Angular/Node ++) - learn language, syntax, do online tutorials etc. Decide on Project Name (Course Req.) Launch Project Website (Course Req.)	**March 21, Monday**
2	Submit Prototype (Course Req.)	**March 28, Monday**
3	Buy Side – Search/Filter functionality Sell Side – Add New Posts to website, Edit Posts (**Front end** – Design Basic Website Theme, Search/Filter UI, Product Listings UI, Sell Item UI, Edit Post UI **Backend** – Logic for search/filter query replies, adding sell item listings **Database** – Store Item Listings w/ details, allow for Add/Delete/Edit, respond to search queries)	**April 4, Monday**

4	Comment/Chat/Messaging system (**Front end** – Add Basic Comment/Chat/Message UI **Backend** – Logic for message send/receive between buyer and seller **Database** – Store Message/Comment threads, allow for adding messages/comments to thread)	**April 11, Monday**
5	User Profile side – Personal Info User Profile side – Post History and Active Posts User Profile side – Email & Alert Notifications Submit Alpha version of application (Course Req.) (**Front end** – Add User page UI w/ personal info, post history, active posts and email/alert settings **Backend** – Logic for serving user post history and active posts, email integration with comments/messaging, alert users depending on alert settings when new item added **Database** – Store personal user info, alert/email settings, respond to queries for user posts)	**April 18, Monday**
6	Payment options – Card, Venmo/Stripe Submit Beta version of application (Course Req.) (**Front end** – Add payment option buttons **Backend** – Logic for Stripe/Venmo Integration **Database** –)	**April 25, Monday**
7	Final Debugging Demo and Submission (Course Req.)	**May 2 – May 4, Mon-Wed**

SECTION 6: RISKS AND OUTCOMES

Using software tools from outside our project, like MongoDB, will be an important part of our design. This will carry many benefits for us as it would for any project - it will allow us to focus on our own functionality without getting bogged down in the fine mechanics of a more general structure like a database. However, doing so retains some major risks for our project as we will not have control of the software. The key risks here are:

1. **Changes to outsourced software's functionality:** Any changes in the implementation of outside software will likely have negligible effects on our project; however, more general changes to how that software is used could create major bugs in our software, or worse yet render our code completely useless. Such changes may be rare but not unheard of. A great example of this problem in past projects was when Apple changed

the swift language in ways that rendered all iOS projects completely useless. Combatting this risk will require us to closely monitor upcoming updates to the software we use.

2. **Incompatibility between tools:** Not all software is compatible. Using incompatible tools will at the very least make our code extremely clunky and will require extra time. We will need to make sure ahead of time that the different tools we use will fit together.

There will of course be risks in the building of our own code. These include:

1. **Testing Risks:** Certain tests of particular parts of our code may be difficult to anticipate or carry out. Stress tests of the final product for example may require a certain amount of creativity in order to carry out.

2. **Broken Parts:** Some aspects of our desired functionality may be more difficult to implement than we anticipate. Good modularity within our code will therefore be extremely important, for it will allow us to limit the amount of changes to our code when bugs need to be fixed. Good use of modularity will also allow us to hopefully limit the pain if the goal has to be changed. For example, if certain parts of the searching algorithm are harder to implement than expected or we find that we want to change the priority of certain hits over others, it would be helpful if the changes in code that these would require were limited.

3. **Alternate Outcomes of Certain Parts:** These risks will necessitate a certain amount of flexibility when it comes to the desired outcome of our project. Ideally each part of our code can accommodate different outcomes of other parts of the code. For example, if the tags function either gets scrapped or is used differently it should not affect the search function, and the priority of the listing should be able to work without it.