Arman Khondker
EID: aak2464

Assignment 2

Example Business Blueprint

PART 1

Table 1: Stakeholder needs for the e-commerce domain

Quality	Description
Scalable	The service must be able to handle a large influx of users.
Secure	System must securely store and transmit transaction data for sales of new and used items
Flexible	System must allow a wide variety of listing types, to capture wide user audience.
Accessible	Service should work with a wide variety of devices and cater to potential users that need user accommodation.
Fast	Service should be able to load new items in real-time while browsing and provide users with a consumer experience that minimizes irritants.
Consistent	Service should follow a consistent layout, theme, and functionality across all devices and use cases.
Reliable	The service must be reliable and have fail safes in place to prevent financial loss to users.
Clear Navigation	Users should be able to easily navigate to different parts of the product.
Redundant	Data should be stored in more than one location for faster access, and reliability.
Optimized	Users should only see most relevant information in an efficient manner and the app should operate under optimal conditions for performance and functionality.

Table 2: Prioritized Stakeholder Needs

Priority	Need/Quality	Classification	Priority Justification
1	Reliable - The service must be reliable and have fail safes in place to prevent financial loss to users. Additionally, the performance of the app must be reliable and never prevent users from performing their desired actions.	Reliability	The application needs to be reliable to prevent users from suffering from economic loss and potential scam. The perception of the product as being reliable and useable in all circumstances is of the upmost importance. Additionally, we want our users to have a safe and secure place to exchanges goods and for us to meet their inherent needs. This application will be the most successful in the long term only if the

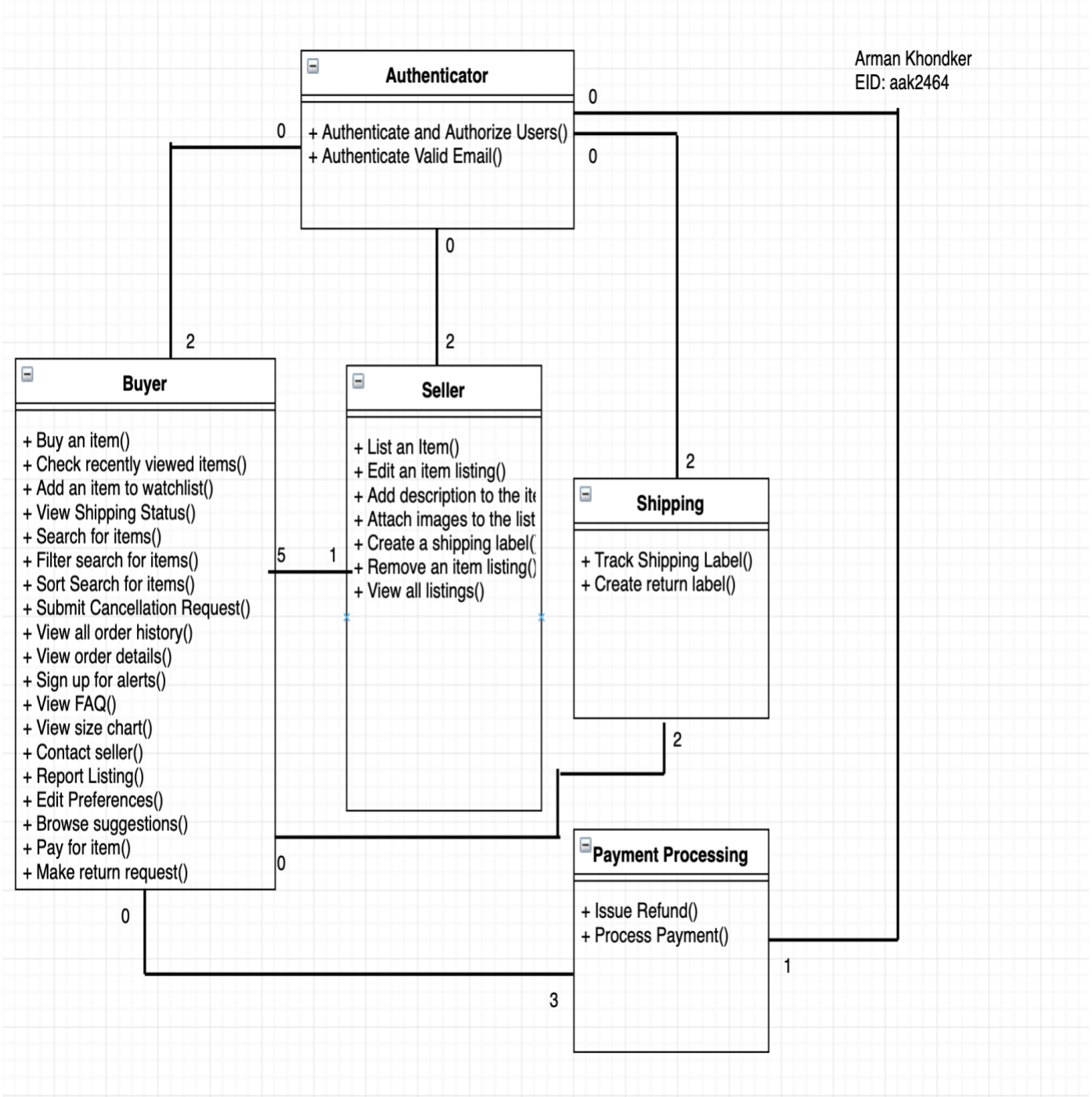
			application is guaranteed to be reliable to a high degree of certainty.
2	Optimized - Users should only see most relevant information in an efficient manner and the app should operate under optimal conditions for performance and functionality.	Performance	This e-commerce implementation needs to be proven to a new user base; therefore, optimization is key because any inefficiencies can cause the product to quickly fail in the long term when adopted by a large user base. The software needs to be optimized to meet the needs of all users
3	Scalable - The service must be able to handle a large influx of users.	Performance	The application needs to be scalable to meet the needs of potentially thousands and thousands of e-commerce transactions that could take place at any given time.
4	Flexible - System must allow a wide variety of listing types, to capture wide user audience.	Flexibility	The application needs to be able to transform to any potential user needs that can develop. Additionally, the product and software need to be flexible so that users within the e-commerce domain will not feel limited in functionality and use cases.
5	Secure - System must securely store and transmit transaction data for sales of new and used items	Usability	Our user's security is of upmost importance and we want to provide users with the ability to use our application without any sort of fear. We want our data to be secure and safe from the hands of malicious groups and individuals.
6	Accessible - Service should work with a wide variety of devices and cater to potential users that need user accommodation.	Usability	The ability to have our website to be flexible and work on different environments is pivotal. Additionally, we need our application to cater to all potential users to prevent the limitation of our application's target audience.
7	Fast - Service should be able to load new items in real-time while browsing and provide users with a consumer experience that minimizes irritants.	Performance	Speed is very impactful on a user's overall experience. Moreover, the application will suffer greatly if it can't operate within a reasonably timely manner. We need our app to be fast to avoid potential issues with user retention.

8	Redundant - Data should be stored in more than one location for faster access, and reliability.	Performance	Redundancy directly applies to the speed of the application which too is important. We want our data to be held across multiple locations so that we can have our website working in the most optimal conditions possible. Additionally, redundancy reduces the risk that we have to take when developing our application in the future.
9	Clear Navigation - Users should be able to easily navigate to different parts of the product.	Performance	User experience is relatively important because it can have a huge impact on how a customer views our product. Additionally, we want to limit the time that users spend figuring out basic layout, while maximizing the time that users can spend browsing our e-commerce platform listings.
10	Consistent - Service should follow a consistent layout, theme, and functionality across all devices and use cases.	Usability	The website needs to function in the same manner no matter what the conditions are to guarantee optimal user experience and product performance. The consistency of our product will also produce a product that deems customer approval and high satisfaction rates.

Arman Khondker
EID: aak2464

PART 2

UML



Arman Khondker
EID: aak2464

Table 3: Derivation Plan

1	**Goal: Reliability**	
1.1	BB Heuristic Reduce Data/Event Dependency (reduce component-to-component coupling between inputs.)	• *Why*: By reallocating functions and data to reduce coupling between components, we can reduce potential bottlenecks which could impact our application's reliability. Additionally, we can make our communication between components faster due to our decrease in coupling between components. • *Priority Justification:* The reliability of our application is the most pivotal goal we are trying to accomplish and reducing potentially bottlenecks through the reduction in dependency is essential.
1.2	BB Heuristic Group based on Data Usage Frequency	• *Why*: By grouping based off the most used function inputs/outputs, we can make our application more reliable and consistent. Additionally, we will be able to derive insights from the different frequency levels observed based on our grouping. • *Priority Justification:* Although the data usage frequency can be important, 1.1 should take precedence because it has more overall impact on the system as a whole.

2	**Goal: Performance**	
2.1	BB Heuristic Group based on Implementation Reality	• *Why*: By grouping our components based off what the existing technology can provide, we will optimize our ability to implement our system. This will in turn improve the performance of the overall system. • *Priority Justification:* If we can ensure that existing technologies can match our component boundaries, then we can begin to heavily optimize our system. This is the first step to optimizing our overall system.
2.2	BB Heuristic Specify Overlapping Capabilities (performer hierarchy)	• *Why*: By increasing inheritance and eliminating duplicate definitions we will reduce coupling and allow us to optimize all of our code. We will have limited developer working on this application initially and having a high degree of code reuse with heavily optimize our solution. • *Priority Justification:*
2.3	BB Heuristic Leverage Task Parallelism	• *Why*: The ability to allow functions to run in parallel will allow us • *Priority Justification:* While having functions run parallel can be useful, the developers will benefit more from the above heuristics and get more ROI from the above heuristics.
2.4	BB Heuristic Isolate risk – Functions	• *Why*: By isolating the risk for functions through isolating services such as the buy/sell. We can then update each of these microservices so that they can be updated with minimal impact upon other functions. This will make the application easy to scale and thus improve the performance of the application.

		• *Priority Justification:* This heuristic is fundamental in scaling the application to the size needed to serve our potential pool of users; however, the other above heuristics can have a higher impact on performance.
3	**Goal: Flexibility**	
3.1	BB Heuristic Reduce Blueprint Complexity – Abstraction	• *Why*: By reducing the complexity of our business blueprint initially, we can increase the depth of the inheritance hierarchy and make our solution more flexible for the future. This would allow our solution to be easier to adapt to multiple situations because of the inherent lack of complexity that our components hold. • *Priority Justification:* This is the only heuristic here (N/A)
4	**Goal: Usability**	
4.1	BB Heuristic Group based on Architectural Style – Client/Server based on Functions	• *Why*: By grouping based off the Client/Server architectural style, we can make our application more usable by putting functions which need to be highly available into a single component. This will also improve the cohesion of all of our components. • *Priority Justification:* This is the only heuristic here (N/A)

All stakeholder needs are addressed in the above derivation plan.
The bootstrap I specified is a reasonable starting point based on my derivation plan because it primarily takes into account my top identified goals and the subsequent heuristics required to effectively achieve that goal. Furthermore, the bootstrap I specified clearly strives to achieve the goals laid out in the derivation plan because it allocates components and functions in a manner that aims to achieve the goals laid out in the derivation plan.

Table 4: Potential Conflicts and impact on derivation plan

Potential Conflict	**Potential Resolution**
Group based on Data Frequency Usage (1.2) may conflict with *Group based on Implementation Reality (2.1) because* of the inherent boundaries that each of the components in the business blueprint. The grouping based on implementation reality imposes boundaries based on existing solutions while Group based on data frequency usage imposes boundaries based on how frequently function input/outputs are used.	Given that in Goal #2, Group based on implementation Reality has the highest priority and performance is an extremely important goal we should give precedence. Thus, we should set boundaries based on the implementation reality first when bootstrapping the blueprint, and then potentially move to apply boundaries for data frequency usage on later iterations.
Isolate Risk – Functions (2.4) may conflict with a number of other heuristics such as *Specify Overlapping Hierarchy (2.2)* because when we increase the inheritance between components and reducing coupling it may decrease our ability to collect and use functions that are based on implementations that tend to change quickly.	Given that *Isolate Risk – Functions (2.4)* is of low priority we can ignore it for the most part and give priority to *Specify Overlapping Hierarchy (2.2)* as it has greater impact on improving the overall quality of the system through bigger performance benefits.
Leverage task parallelism (2.3) may conflict with *Group based on Architectural Style – Client/Server based on Functions (4.1)* because when we place functions into different components so that they can be performed in parallel, we reduce the opportunity to put functions that use highly available data into a single component.	Since *Leverage task parallelism (2.3)* is not as high precedence as *Group based on Architectural Style – Client/Server based on Functions (4.1)* and Grouping based on architectural style will have a higher overall impact on its respective goal of usability, we should treat 4.1 as the heuristic that plays a greater role in our bootstrap and design decisions.

Arman Khondker
EID: aak2464

Allocation of functions and data components

Component	Functions and Data
Authenticator	Functions: • Authenticate and Authorize Users • Authenticate valid email Data: • Username • Password • Email Address • Street Address
Seller	Functions: • List an Item • Edit an item listing • Add Description to the item • Attach images to the listing • Create a shipping label • Remove item listing • View all listings Data: • Pictures of items • Description of items • Cost of items • Condition of items • Seller history • Listing history

Arman Khondker
EID: aak2464

Buyer	Functions: • Buy an item • Check Recently Viewed Items • Add an item to watchlist • View shipping status • Search for items • Filter search for items • Sort Search for items • Submit cancellation request • View all order history • View order details • Sign up for alerts • View FAQ • View size chart • Contact seller • Report listing • Edit preferences • Browse suggestions • Pay for an item • Make a return request Data: • Viewed items history • Watchlist of items • Purchase history • Search History • Email preferences • Item preferences
Payment Processing	Functions: • Issue refund • Process Payment Data: • Transaction Cost • Payment Method • Buyer ID • Seller ID

Arman Khondker
EID: aak2464

Shipping	Functions: • Track shipping label • Create return label Data: • Transaction ID • Buyer address • Seller address

Component-to-Component I/O Dependencies

Components	Functions and Data
FROM: Authenticator TO: Seller	• List an item is allocated to Seller and requires that authenticate and authorize which is allocated to Authenticator create the event that the user is logged in • Create a shipping label which is allocated to Seller requires data from Authenticator which is created during the authenticate email
FROM: Authenticator TO: Buyer	• Buy an item is allocated to Buyer and requires that authenticate and authorize which is allocated to Authenticator create the event that the user is logged in • Sign up for alerts is allocated to Buyer and requires that authenticate. Email which is allocated to Authenticator create the event that email has been verified
FROM: Seller TO: Buyer	• Buy an item is allocated to Buyer and requires that list an item which is allocated to seller create the event that the user has listed the item for sale • View shipping status which is allocated to Buyer requires that the event take place from the Seller that the item has been shipped • Report a listing which is allocated to Buyer requires that the function list an item which is allocated to Seller has been completed. • Contact seller which is allocated to buyer requires the seller account which is in the Seller component

	• Submit cancellation request which is allocated to buyer needs the seller address data which is allocated to the seller
FROM: Buyer TO: Seller	• Create a shipping label which is allocated to seller requires the data from the Buyer which holds their shipping address
FROM: Authenticator TO: Payment Processing	• Issue refund is allocated to Payment processing and requires that authenticate and authorize which is allocated to Authenticator create the event that the user is logged in • Process payment is allocated to Payment processing and requires that authenticate email which is allocated to Authenticator create the event that the user has authenticated their email
FROM: Authenticator TO: Shipping	• Track shipping label is allocated to Shipping and requires that authenticate and authorize which is allocated to Authenticator create the event that the user is logged in • Create return label is allocated to Shipping and requires that authenticate and authorize which is allocated to Authenticator create the event that the user is logged in
FROM: Buyer TO: Shipping	• Track shipping label which is allocated to shipping requires pay for an item which is allocated to buyer to create the event that the item has been paid for before a user can track an item • Create return label which is allocated to shipping requires the address of the buyer which is allocated to buyer
FROM: Buyer TO: Payment Processing	• Process Payment which is allocated to Payment Processing requires the address of the buyer which is allocated to buyer • Process Payment which is allocated to Payment Processing requires that pay for item be completed by the Buyer • Issue refund which is allocated to Payment Processing requires that the function valid return or valid cancellation which is allocated to the Buyer component

I/O Dependencies within the same component

Components	Functions and Data
Authenticator	• Authenticate and authorize users requires username and password from Authenticator component

	• Authenticate valid email requires the data from the Authenticator component for a user's email
Seller	• List an item requires • View all my listings which is allocated to Seller requires the given input for all listings that a seller has made • Remove item listing requires data from the Seller component such as listing ID
Buyer	• View all orders which is allocated to Buyer requires the given input for all purchases a user has made • Make return request which is allocated to Buyer requires the order history of a buyer which is data within the Buyer component itself
Payment Processing	• Issue refund which is allocated to Payment Processing requires data on the transaction amount which is allocated to the payment processing component itself

Component-and-External Producers/Consumers

Components	Functions and Data
Authenticator	• Authenticate and Authorize is allocated to Authenticator and requires username and password as input from an external consumer • Authenticate valid email is allocated to Authenticator and requires the email of a user as input from an external consumer
Seller	• List an item is allocated to Seller and requires pictures of items, description of item, cost if item, and item condition as input from an external consumer • Edit an item Listing is allocated to Seller and requires pictures of items, description of item, cost if item, and item condition as input from an external consumer
Buyer	• Buy an item is allocated to Buyer and requires user requested an item as input from an external consumer • Check recently viewed items is allocated to Buyer and requires User requested to view list of items recently viewed from external consumer • Add an item to Watchlist is allocated to Buyer and requires user requested to add item to watch list from an external consumer.

	• Search for items is allocated to buyer and requires a user input that comes from an external consumer • Sign up for alerts is allocated to. Buyer and requires the user to indicate alert criteria which is from an external consumer
Shipping	• Create return label which is allocated to the shipping component requires that a user generate an external request to return an item

PART 3

Coupling and Cohesion Metrics

Component	**# of inputs received from other components**	**# of outputs sent to other components**	**# of Dependencies (number of components that this component interacts with)**	**Degree of Cohesion (Percent of functions that receive inputs and send outputs within this component)**
Authenticator	0	4	5 (all components interact with the authenticator component)	0% (both functions send an event to other components which call these functions) (0/2)
Seller	9	13	4	0/7 = 0% (all 7 of the functions require that the user be logged in and thus require an input from the authenticator component)
Buyer	23	15	5	7/19 = 37% (only 7 of the functions either don't require the user to be logged in or don't send data to other components)
Payment Processing	3+3 = **6**	4	2	**0%** (both functions require the user to be logged in which is received as an input from the authenticator component) (0/2)
Shipping	3+2 = **5**	3	2	**0% (**both functions require the user to be logged in which is received as an input from the authenticator component) (0/2)

Arman Khondker
EID: aak2464

Size and Complexity Metrics

Component	**# of functions**	**# of data elements**	**Component Complexity (Data Elements + Functions + Inputs/Outputs)**
Authenticator	2	4	4+2+9 = **15**
Seller	7	6	*7+6+24 =* ***37***
Buyer	18	6	*18+6+36 =* ***50***
Process Payment	2	4	2+4+6 = **12**
Shipping	2	3	2+3+6= **11**

There are a total of **5 components**, 31 functions and 23 data elements **in the blueprint as a whole.**

Support for Applied Heuristic

The heuristic of *Group based on architectural style – Client/server based on functions* has been applied and is displayed from the tables because in our blueprint we have collected functions that need to be highly available and put them into a single Buyer component. The metrics of the # of functions for the Buyer component and the component complexity for the buyer component clearly show that the heuristic of *Group based on architectural style – Client/server based on functions* has been applied.
Additionally, the heuristic of *Reduce Blueprint Complexity – Abstraction* has been applied and is displayed from the tables because in our blueprint we have reduced the number of components and made them more abstract. The metrics of total # of components, component complexity for each of our components, and # of dependences between our components which are high clearly show that the heuristic *Reduce Blueprint Complexity – Abstraction* has been applied.