RENTIT

TEAM MEMBERS:

SAHIL SEHGAL

ADITI MALLESH

TEJAS MEHTA

AKSHAY MAHAJAN

NIKHIL GOHIL

Syracuse University

AGENDA

- What is RentIt?
- User requirements
- System requirements
- UML diagrams
- System architecture
- Design Principles
- Design Patterns

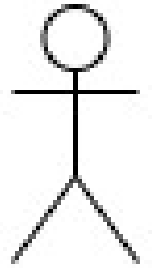
WHAT IS RENTIT?

"Why simply BUY or SELL when you can RENT?"

Syracuse University

USER REQUIREMENTS

- Functional Requirements
  - User should be able to view the available products based on location.
  - User should be able to search the products.
  - User should be able to rent the products.
  - User should be able to add products with their images on the portal.
  - User should be able manage the uploaded products.
  - User should be able to view the order history.
- Non-Functional Requirements
  - The application should be up and running for everyone with internet access.
  - Privacy of user should be maintained, and data should be stored securely.
  - Application should maintain different roles and permissions for product owner and renter.



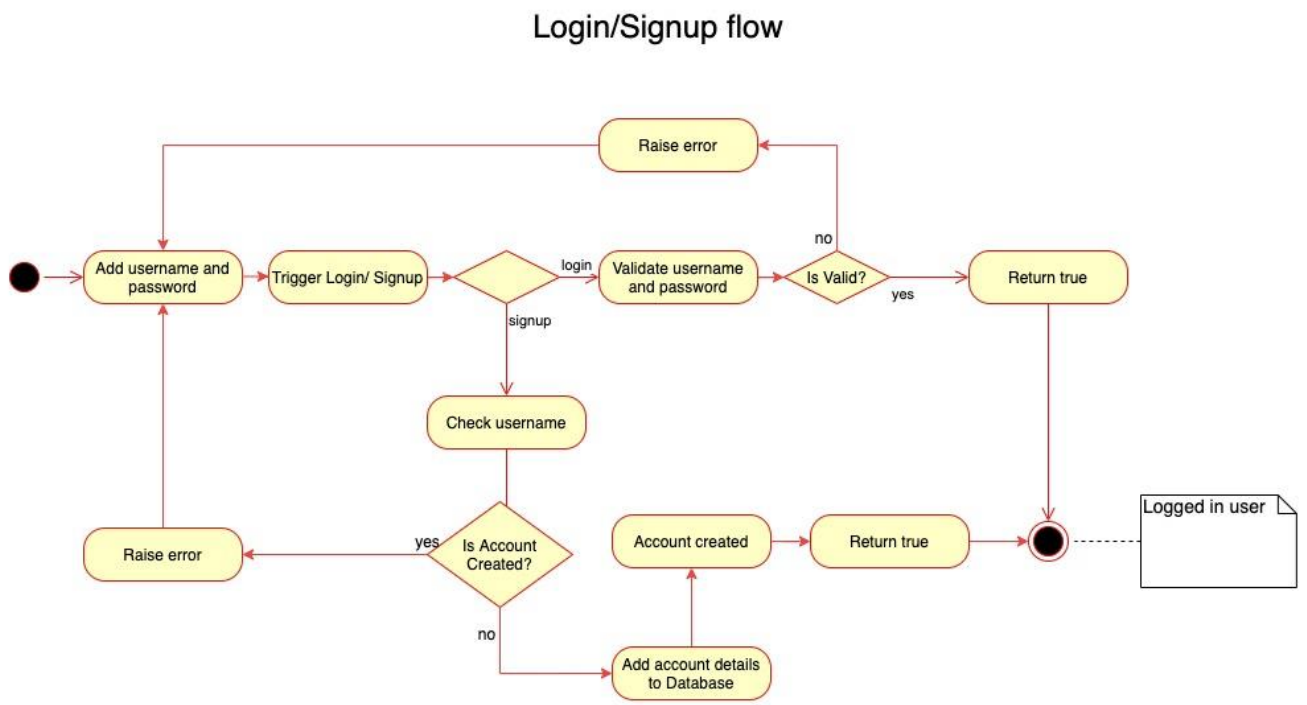
User

SYSTEM REQUIREMENTS

- Functional Requirements
  - System should be able to provide list of products with search filters.
  - System should be able to recommend products to user based on the zip code.
  - System should be able to recommend products based on product type.
  - System should be able to store user information in the database.
  - System should be able to store the products inside the database.
  - System should be able to send email notifications to both the users when an order is placed.
- Non-Functional Requirements
  - System should be scalable, resilient and fault tolerant.
  - System should be secure and protect user data from hackers.
  - System should offer a user-friendly UI and it should be self explanatory.

UML DIAGRAM

- Activity Diagram for Login/Signup flow



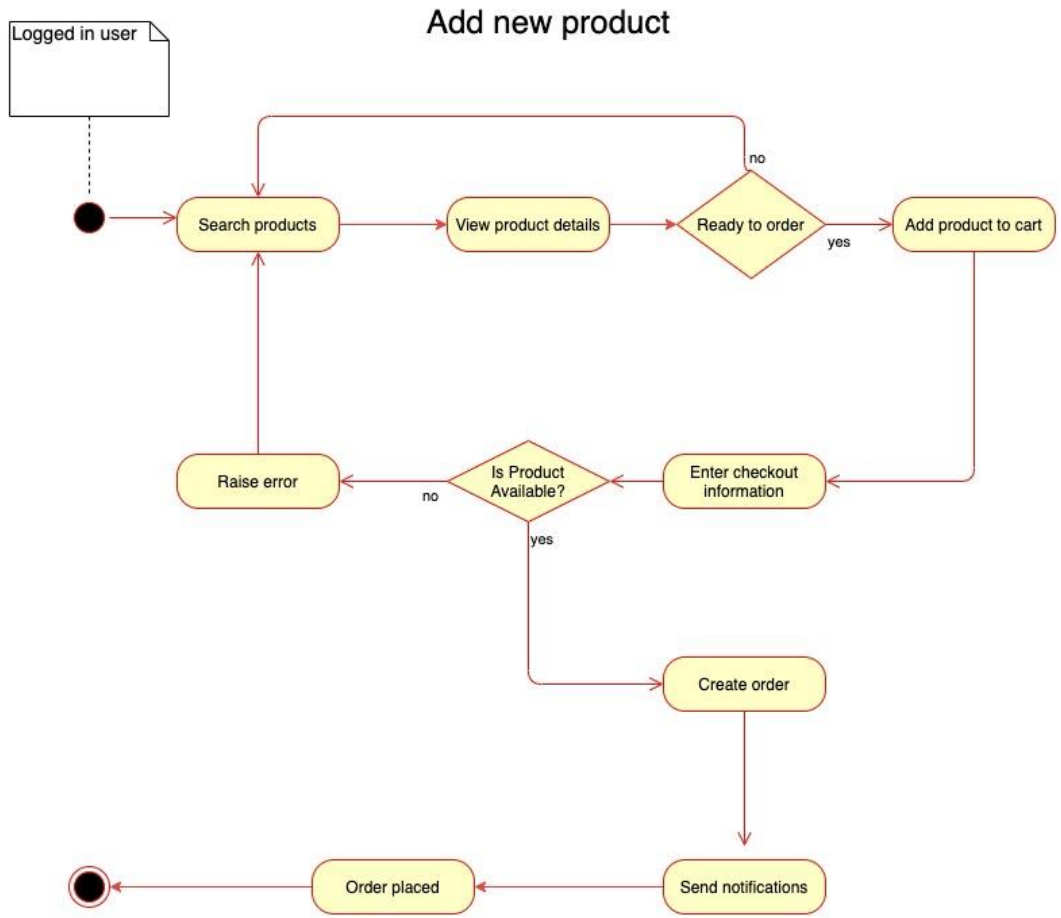
Syracuse University

ACTIVITY DIAGRAM FOR LOGIN/SIGNUP FLOW

- The activity diagram shows the scenario for login/signup of users.
- The user can either sign up for a new account or login to an existing one.
- The user also has the ability to reset the password by selecting Forgot password.
- The users can either take the role of: Publisher and consumer, once logged in.

UML DIAGRAM

- Activity Diagram for adding new product



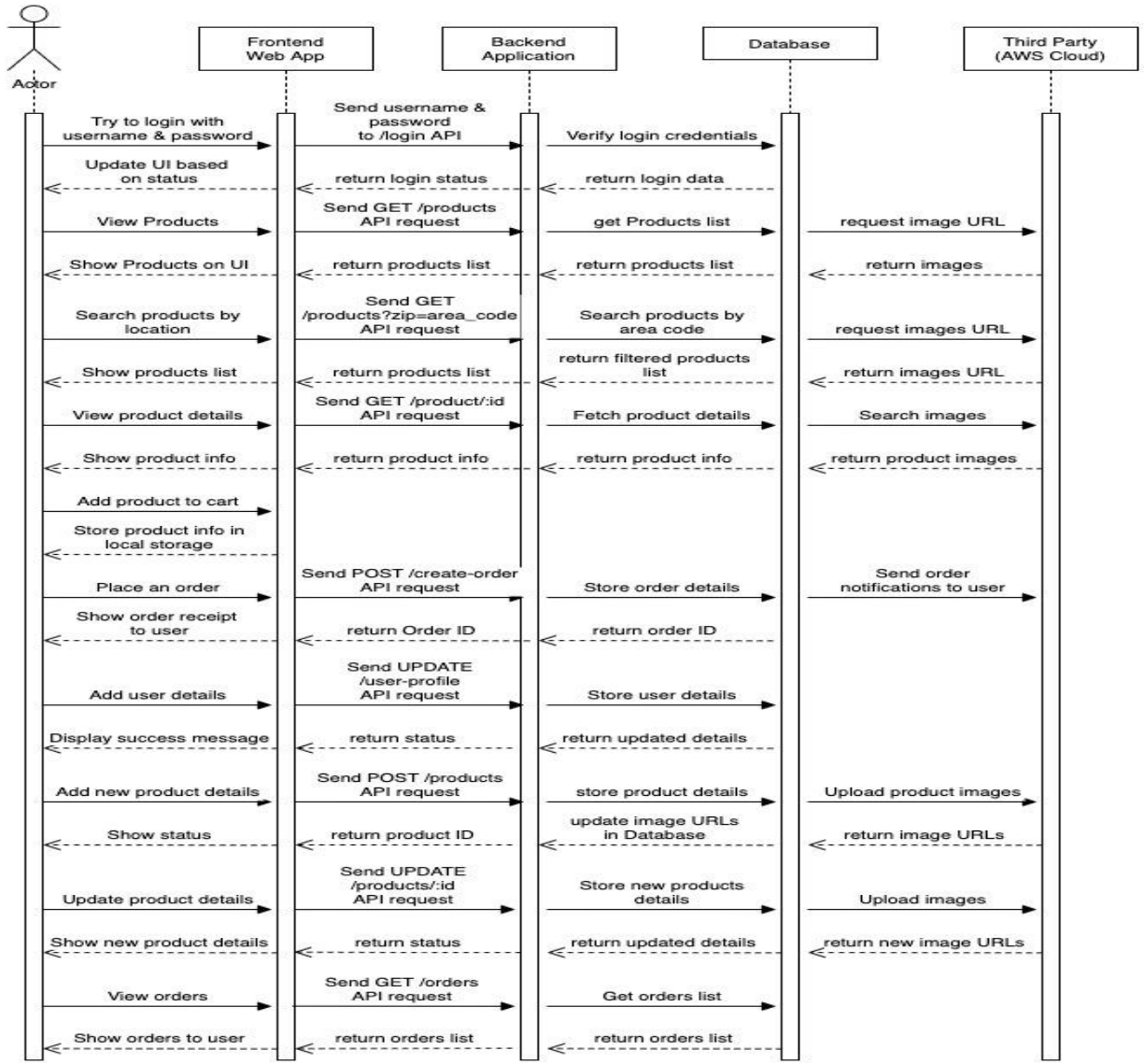
Syracuse University

ACTIVITY DIAGRAM FOR ADDING NEW PRODUCT

- The diagram shows the scenario for adding a new product.
- The Publisher has the ability to add new products which a consumer can rent based on availability.
- Publishers can add the product details in the Add new product page, once the user is verified, the details are updated on the database.
- Once updated the user will be notified on the same.

UML DIAGRAM

- Sequence diagram



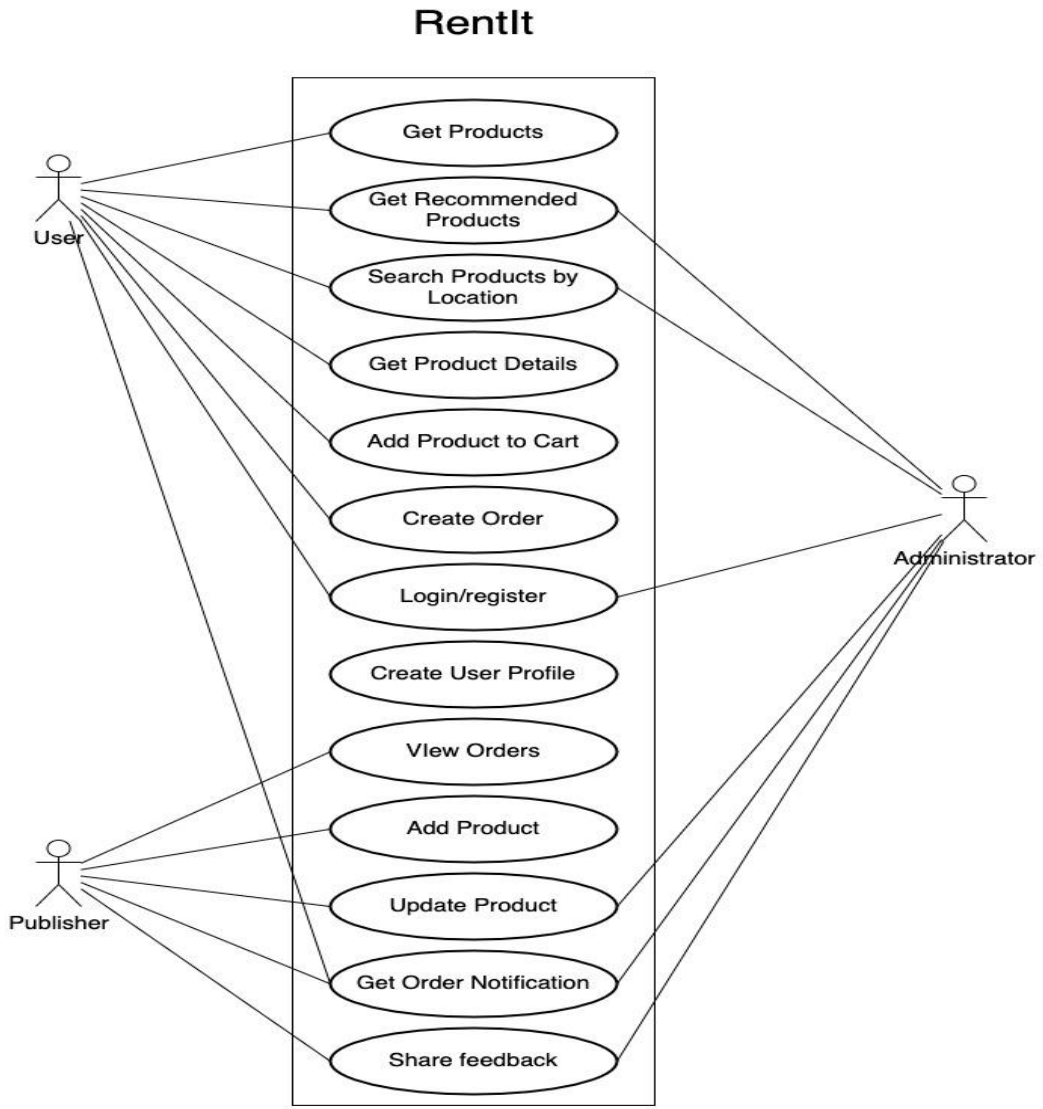
Syracuse University

SEQUENCE DIAGRAM

- The sequence diagram is a representation of the message transfers between the Publisher and the Consumer across RentIt platform.
- It describes the actions that are performed by the Publisher and Consumer with respect to the permissions they have.
- It shows how the data flow occurs across the system, starting from publisher adding new product to consumer searching for the product of interest, retrieving the details of the product, finally being able to rent the product based on its availability.

UML DIAGRAM

- Use case diagram



Syracuse University

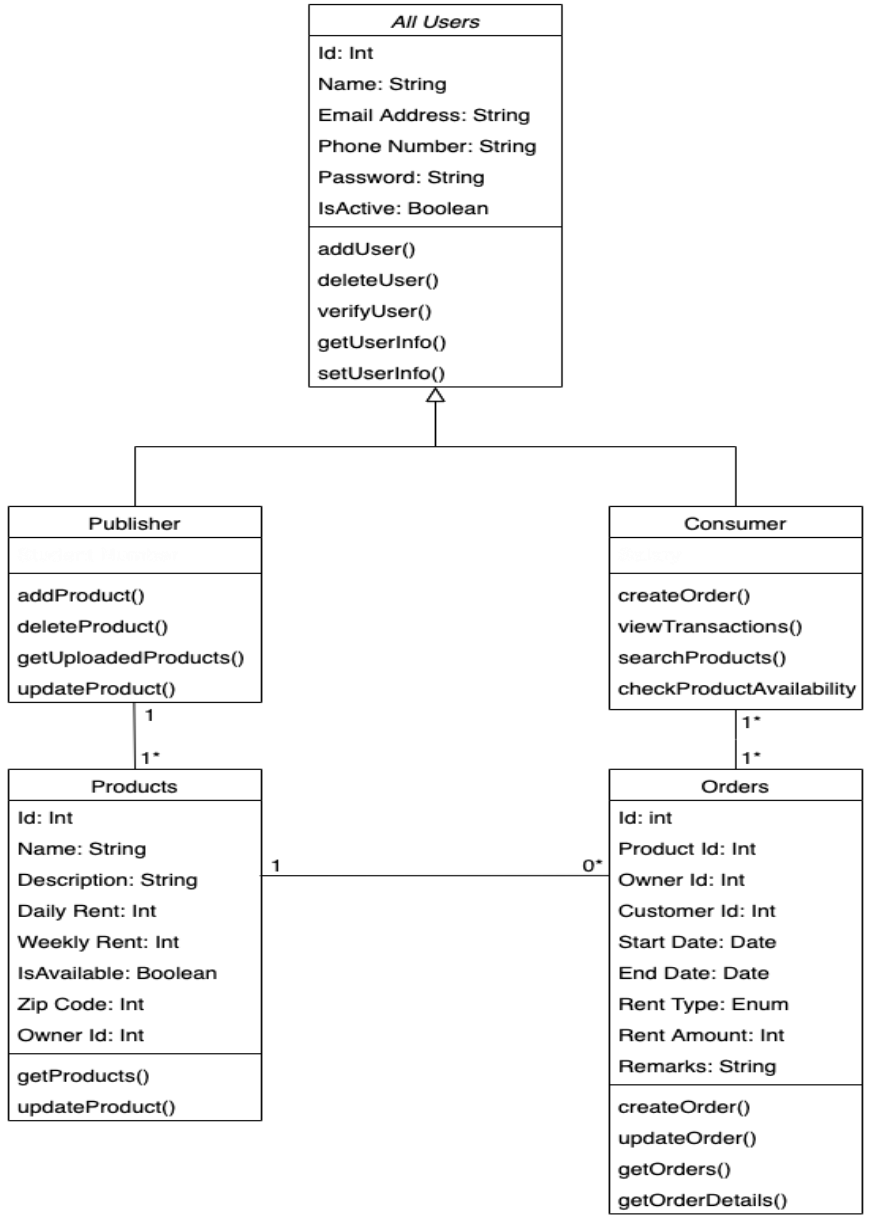
USE CASE DIAGRAM

The above use-case diagram is a representation of the associations between application users (publisher and consumer) and the use cases:

- **Create Account:** The user can create a profile on web application using registration form.
- **Select User Role:** The user can have more than one role, publisher/consumer.
- **Add New Product:** Publisher has ability to add new products that he/she wish to put on rent.
- **Manage Products:** Publisher has the ability to add/update/delete the post.
- **View Transaction:** Publisher can view the order details for respective products and update the availability dates on each item accordingly .
- **Search Products:** Consumer can search for the products based on the interest.
- **View Products:** The products will be displayed on the homepage based on the
- availability within the location, according to zip code entered by the user.
- **Rent Products:** Ability for the consumer to rent the product of the interest that is
- Available.

UML DIAGRAM

- Class Diagram



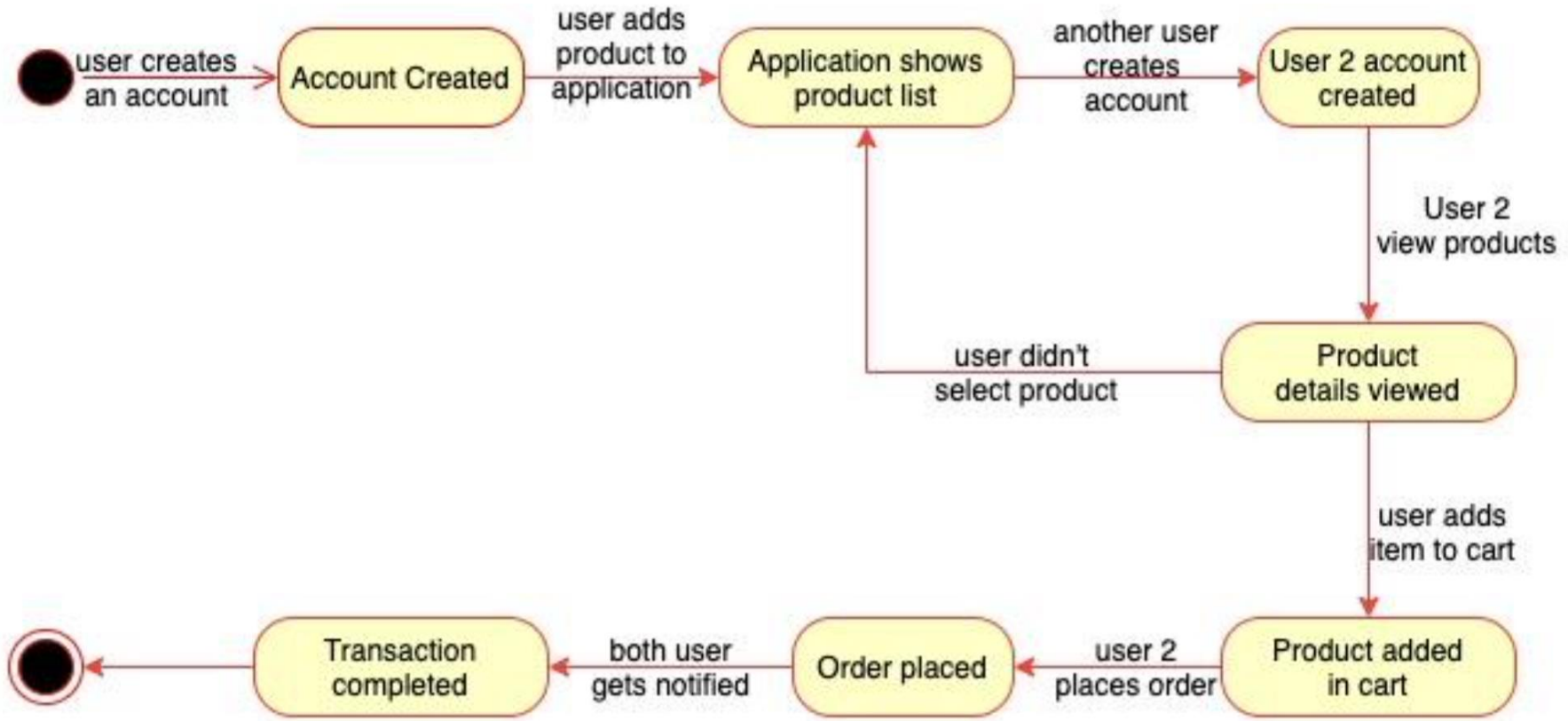
Syracuse University

CLASS DIAGRAM

- **All user:** Includes all methods and details related to registration of the user and the roles
  - **Publisher** - Includes all functions of users who can add/update/delete products they wish to give for rent
  - **Consumer** - Includes all functions of users who can search and place order for the products they wish to rent based on its availability
- **Products:** Includes all functions related to the products like name, description, availability, owner details and zip code to help for better recommendation
- **Orders:** Includes all details related to orders placed, like owner and renter details, the price paid and also, the information related to the date (start and end of the rental)

UML DIAGRAM

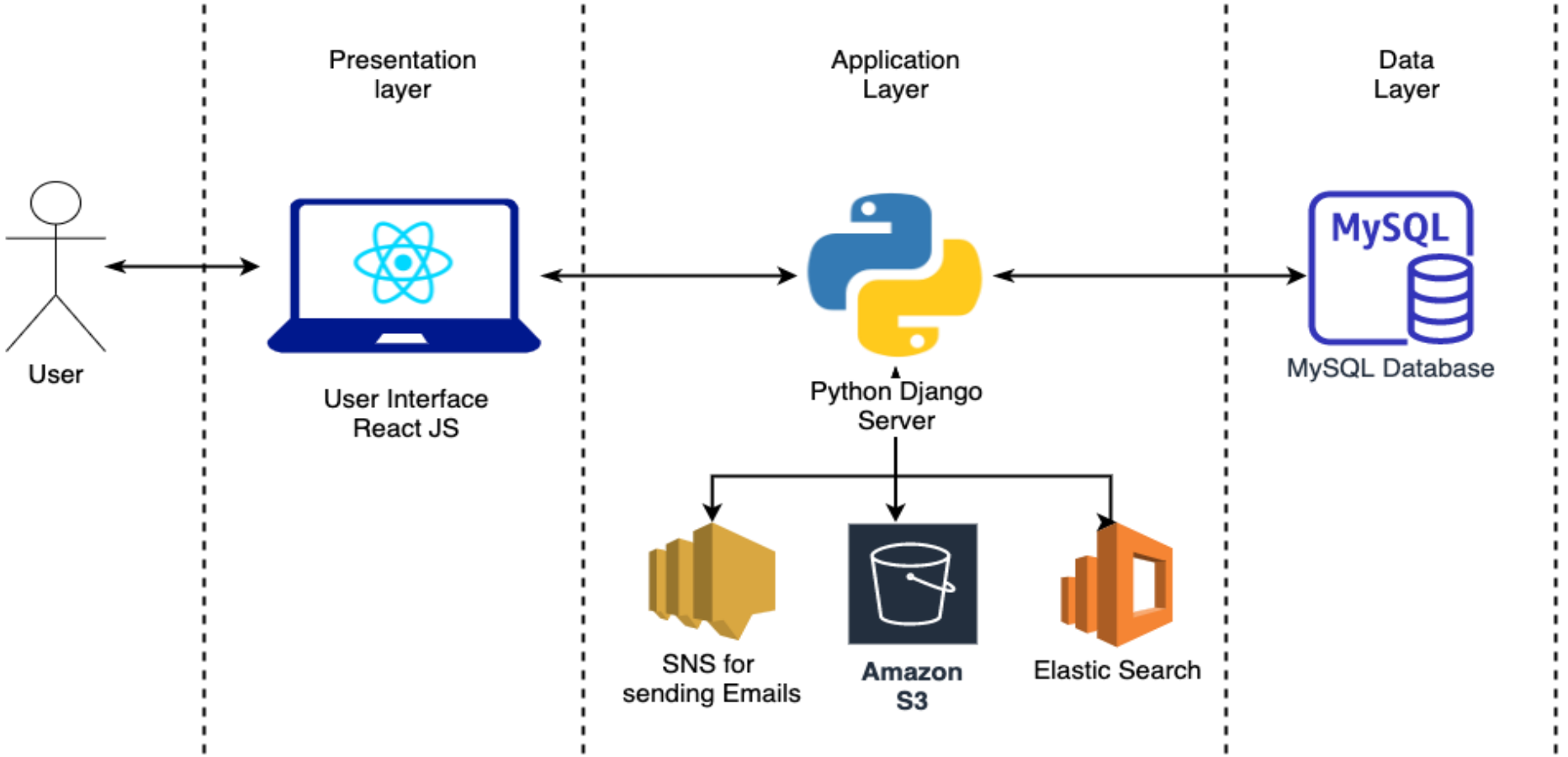
- State Diagram



STATE DIAGRAM

- Once account is created, the application displays the list of the items and their availability based on the information provided by the Publisher.
- Products and the details of the product are displayed to the Consumer based on the location through zip code
- Display the availability of the product, verify the same before placing order
- If product is available, allow user to rent it and update the database
- Send email notification to both Publisher and Consumer of the product

SYSTEM ARCHITECTURE



DESIGN PRINCIPLES

- **Don’t Repeat Yourself:** The Don’t Repeat Yourself (DRY) principle states that duplication in logic should be eliminated via abstraction; duplication in process should be eliminated via automation. Django models create separate constructs based on DRY principle.
- **Encapsulate what changes:** There is only one thing which is constant in the software field and that is "Change", So, encapsulate the code you expect or suspect to be changed in future. The benefit of this OOP Design principle is that It's easy to test and maintain proper encapsulated code.
- **Open closed Design principle :** "Classes, methods or functions should be Open for extension (new functionality) and Closed for modification". The key benefit of this design principle is that already tried and tested code is not touched which means they won't break.
- **Loosely coupled architecture:** The Django framework being an MVC framework operates across multiple tiers (e.g. HTML templates, business logic methods and databases). However, Django takes great care of maintaining a loosely couple architecture across all the components that operate across these tiers.

DESIGN PATTERNS

- **Model:** Model classes include attributes of an application entity and an instance of a model class represents state of a data record belonging to that application entity.
- **View:** Controller services user requests to modify application data. For example, a user can submit a form containing data relevant to an application entity, controller validates the data submitted and then either commits the data to the database or generates an appropriate error message to the user.
- **Template:** View renders data to application users and handles user generated requests to modify data by invoking appropriate methods in controller.

DESIGN PATTERNS

- We are following MVT design pattern for our project and have created one model class for every application entity.
- **Listing Class** -Listing entity
- **Contact class-**Contact entity
- **User class** -User Details entity
- **Enquire class** -Enquire entity
- **Template**-> In order to render data to application users, We have used Django Template binding from the server end, as to help faster binding and SEO friendly. Django Template
- **View**-> Any modification requests to one of the above application entities goes through servlets, which accepts input data from application users and then commits it to the database.
- **Databases ->**All the structured data like user-profile, auctions, items and likes are stored in MySQL because we need a Database which follows ACID properties. It blends in SQL operations and to query, process data through connector or Play MVT. It will be deployed in AWS-RDS

Thank You.

Syracuse University