SRS FOR PUBLIC PHOTOGRAPHY CONTEST

1. Introduction

1.1Purpose

This Public Photography Contest with Live Voting System Software Requirement Specification (SRS) main object is to provide a base for the foundation of the project. It gives comprehensive view of how the system is supposed to work and requirements for implementing the system.

This SRS can also be used for future as basis for detailed understanding on how project was started. It provides a blueprint to upcoming new developers and maintenance teams to assist in maintaining and modifying this project as per required changeability.

1.2 Scope

The system will be used to properly portray the skills of photographers. It maintains three levels of users: Admin, Photographers and Voters (or users).

Admins will be able to conduct contest based on specific themes, notify the registered users, check if uploaded photograph is not taken from any other source and declare the winners of a contest. Photographers can register to a contest and upload photographs to show his skills. Voters can vote for three best photographs in a contest.

1.3Definitions, Acronyms and abbreviations

SRS	Software Requirement Specification
HTML	Hyper-Text Markup Language
CSS	Cascading Style Sheets
UI	User Interface
OS	Operating System
IT	Information Technology
AMD	Advanced Micro Devices

1.4References

[1] Fundamentals of Software Engineering, Rajiv Mall

1.5 Document overview: The remaining part of this document contains other information about the system such as general description of the system, functional requirements, external interface requirements, performance requirements, design constraints and other requirements.

2 General Description

2.1 Product Perspective:

This is a web based system implementing client-server model. The system provides a platform for photographers where they can show their skills and win some awards.

2.2 Product Functions Overview:

- Users registration
- Users login
- Create contest
- Contest registration
- Originality check
- Limitation check
- Users voting
- Winner declaration

2.3 User Characteristics

There are 3 user levels in this system:

A. Admin
B. Photographers
C. Voters

Admin

Admin can conduct contest based on a theme. The admin must check whether images belong to corresponding photographer only, and not taken from other sources. If entries exceed the required participation limit, admin will select top images and users will vote for the best one. Admin can choose the winner either by the number of likes on a photograph or directly select the winner. Admin must have good knowledge of application server.

Photographer

Photographer can register to contests and upload a photo. Photographer must have basic IT knowledge to register and upload photo.

Voters

They can view all the ongoing contests and vote for photograph they like (maximum 3 votes for each contest). Voters must be capable of using the web UI interface.

2.4 General Constraints

I. **Memory:** System will have only 10GB space of data server.
II. **Language Requirement:** Software must be only in English.
III. **Budget Constraint:** Due to limited budget, this system is intended to very simple and just for basic functionalities. UI is going to be very simple.
IV. **Implementation Constraint:** Application should be based on Java.

3 Functional Requirements

Function 1	**Users registration**
Input	Name, Email_id, Password
Processing	A new record with the above details will be created in the users table of the database.
Output	System should send confirmation message to the user

Function 2	**Users Login**
Input	Email-id, Password
Processing	Check if there exists a user in the database with the entered credentials.
Output	The user should be directed to the main page if the credentials are correct, otherwise the system should ask the user to enter his credentials correctly.

Function 3	**Create contest**
Input	Contest dates, theme
Processing	A new record with the above details will be created in the contests table of the database.
Output	System should send notification to all the users about the contest one week before the contest starts.

Function 4	**Contest Registration**
Input	Contest_id , Email_id, photograph
Processing	A new record with the above details should be inserted into the contest_image table of the database.
Output	System should send confirmation message to the photographer that he has been successfully registered for the contest.

Function 5	**Originality check**
Input	Image doubted by the admin
Processing	Redirect to google images and upload the image and show results of image sources to admin. Admin selects the images if they are taken from other sources.
Output	The selected images are removed from the contest.

Function 6	**Limitation check**
Input	All the images registered for the contest
Processing	If number of images exceed the limit, admin must select top images for the contest.
Output	Unselected images are removed from the database.

Function 7	**Users voting**
Input	User_id, contest_image_id
Processing	A new record with given user_id and entry_id is created in the voting table if the number of images voted by the voter is less than three.
Output	Display a message that user has voted successfully or ask him to deselect the extra selected images.

Function 8	**Winner declaration**
Input	Contest_id
Processing	Check in votes table for the entry which has secured maximum votes or select the photographer chosen by the admin.
Output	Declare the winner.

4 External Interface Requirements:

4.1 User Interfaces:

The user interface for the system must be compatible to any type of web browser such as Mozilla Firefox, Google Chrome and Internet Explorer.

4.2 Software Interfaces:

- OS(Windows)
- Google Images
- Java, HTML, CSS
- MySQL
- Apache server

4.3 Hardware Interfaces:

4.3.1 Server Side

Monitor	Processor	RAM	Disk Space
Resolution 1024*768	Intel or AMD 2GHZ	4 GB	10GB

4.3.2 Client Side

Monitor	Processor	RAM	Disk Space
Resolution 1024*768	Intel or AMD 1GHZ	512MB	2GB

5 Performance Requirements

- Data in database should be updated within 2 seconds
- Query results must return results within 5 seconds
- Load time of UI should not take more than 2 seconds
- Login validation should be done within 3 seconds

SRS FOR NEWSPAPER AGENCY AUTOMATION SOFTWARE

1. Introduction

1.1 Purpose

This Newspaper Agency Automation Software Requirement Specification (SRS) main objective is to provide a base foundation of the project. It gives comprehensive view of how the system is supposed to work and requirements for implementing the system. This SRS can also be used for future as basis for detailed understanding on how project was started. It provides a blueprint to upcoming new developers and maintenance teams to assist in maintaining and modifying this project as per required changeability.

1.2 Scope

The software will be used to manage the delivery of newspapers and magazines, print delivery details and bills. It maintains two levels of users: Manager and Delivery persons. Manager can add details of newspapers / magazines, add new subscription or modify or stop subscriptions of customers and also stop delivery for certain period for a customer. The delivery persons can check the addresses to which he must deliver the publications.

1.3 Definitions, Acronyms and abbreviations

SRS	Software Requirement Specification
HTML	Hyper-Text Markup Language
UI	User Interface
OS	Operating System
IT	Information Technology
AMD	Advanced Micro Devices

1.4 References

[1] Fundamentals of Software Engineering, Rajiv Mall

1.5 Document overview:

The remaining part of this document contains other information about the system such as general description of the software, functional requirements, external interface requirements, performance requirements, design requirements and other requirements.

2. General Description

2.1 Product Perspective

This software is developed to automate various clerical activities associated with the local newspaper and magazine delivery agency.

2.2 Product Functions Overview:

- Manager/ Deliverer login
- Add Publication
- Edit publication details
- Add customer
- Edit customer details
- Publications to be delivered
- Delivery summary
- Customer bills
- With-Hold subscription
- Payment receipt
- Deliverer payment

2.3 User Characteristics:

There are 2 user levels in this system:

A. Manager

B. Deliverer

Manager

Manager can add new newspaper/ magazine details, add new customers and can edit those details. Manager can view summary of the deliveries and print bills of customers. He can also add, modify, stop and hold subscriptions for a while if customers go out of station. Manager must have basic IT knowledge to operate the software.

Deliverer

Deliverer can view the addresses to which he must deliver the publications every day. He can also view his monthly payment details. Deliverers must be capable of using UI interface of the software.

2.4 General Constrains:

I. **Memory**: Software will require 2GB space.

II. **Language Requirement**: Software must be only in English.

III. **Budget Constraint**: Due to limited budget, this software is intended to very simple and just for basic functionalities. UI is going to be very simple.

IV. **Implementation Constraint**: Application should be based on Java.

3. Functional Requirements

Function 1	**Manager/ Deliverer login**
Input	Username, Password
Processing	Check if there a exists a user in the database with the entered credentials.
Output	The user is directed to his main page if the entered credentials are correct, else the system will ask the user to enter his correct credentials

Function 2	**Add Publication**
Input	Newspaper name, language, description, price
Processing	A new record with the above details is inserted into the database.
Output	Notify the user whether the data is successfully inserted or not.

Function 3	**Edit Publication details**
Input	Newspaper name, language, description, price
Processing	Modify the existing record with the above details.
Output	Notify user about the update.

Function 4	**Add customer**
Input	Customer name, address, phone, subscription
Processing	A new record with the above details is inserted into the database.
Output	Notify the user whether the data is successfully inserted or not.

Function 5	**Edit customer details**
Input	Customer name, address, phone, subscription
Processing	Modify the existing record with the above details.
Output	Notify user about the update.

Function 6	**Publications to be delivered**
Input	Customer address, Customer subscriptions, Deliverers
Processing	Each deliverer is assigned an address to which the subscribed publication must be delivered. The addresses are assigned such a way that the communication of deliverer is minimal.
Output	Print the publications to be delivered.

Function 7	**Delivery summary**
Input	Customers, subscribed publications
Processing	Check which publications the customers have received for the whole month.
Output	Print summary of the delivered publication details in a month.

Function 8	**Customer bills**
Input	Customers, subscriptions, publications
Processing	Calculate the total amount to be paid by the customer for a month based on the type of publication and number of publications received by a customer.
Output	Print the bill for each customer which includes publication type, the number of copies delivered in a month and cost for these.

Function 9	**With-Hold subscription**
Input	Customer id, Duration
Processing	Add above details to database and stop deliveries to the customer for specified duration.
Output	Notify user about the update.

Function 10	**Payment receipt**
Input	Customer id, paid amount, cheque number
Processing	Update database about the payment.
Output	Print payment receipt of the customer.

Function 11	**Deliverer payment**
Input	Deliverers, deliveries made
Processing	Calculate the amount a deliverer should receive by adding 2.5% of the value of the publications delivered by him.
Output	Print the amount payable to each delivery boy.

4. External Interface Requirements

4.1 User Interfaces

The user interface for the system must be compatible to windows OS.

4.2 Software Interfaces

- OS (Windows)
- Java, HTML
- MySQL

4.3 Hardware interfaces

Monitor	Processor	RAM	Disk Space
Resolution 1024*768	Intel or AMD 1GHZ	512MB	2GB

5. Performance Requirements

- Data in database should be updated within 2 seconds
- Query results must return results within 5 seconds
- Load time of UI should not take more than 2 seconds
- Login validation should be done within 3 seconds

Entity Relationship Diagram (ERD)

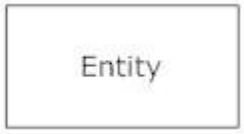
An entity relationship diagram (ERD) shows the relationships of entity sets stored in a database. An entity in this context is a component of data. In other words, ER diagrams illustrate the logical structure of databases.

At first glance an entity relationship diagram looks very much like a flowchart. It is the specialized symbols, and the meanings of those symbols, that make it unique.

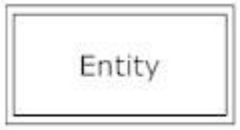
Common Entity Relationship Diagram Symbols

An ER diagram is a means of visualizing how the information a system produces is related. There are five main components of an ERD:

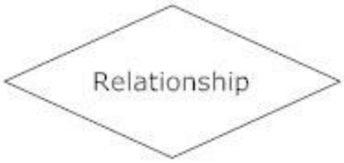
- **Entities**, which are represented by rectangles. An entity is an object or concept about which you want to store information.



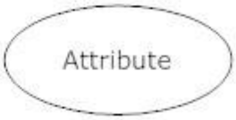
A weak entity is an entity that must defined by a foreign key relationship with another entity as it cannot be uniquely identified by its own attributes alone.



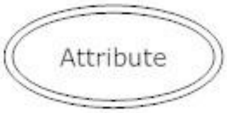
- **Actions**, which are represented by diamond shapes, show how two entities share information in the database.



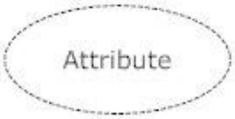
- **Attributes**, which are represented by ovals. A key attribute is the unique, distinguishing characteristic of the entity. For example, an employee's social security number might be the employee's key attribute.



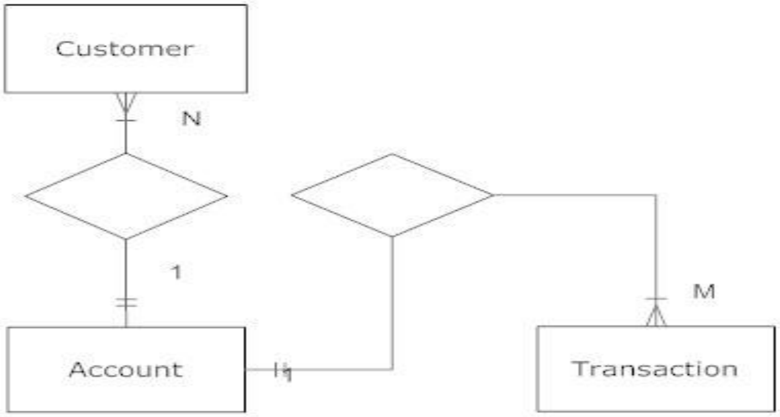
A multivalued attribute can have more than one value. For example, an employee entity can have multiple skill values.



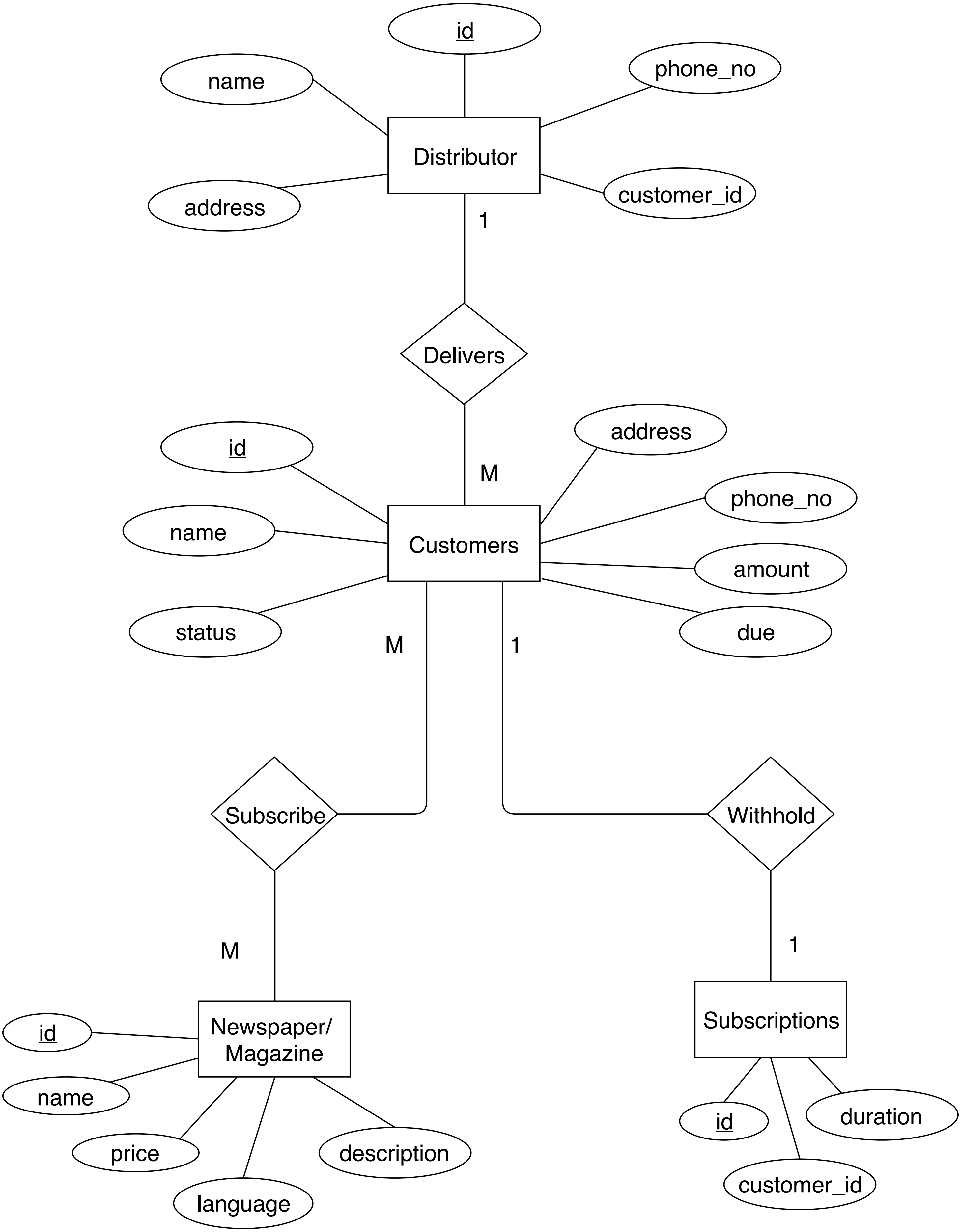
A derived attribute is based on another attribute. For example, an employee's monthly salary is based on the employee's annual salary.



- **Connecting lines**, solid lines that connect attributes to show the relationships of entities in the diagram.
- **Cardinality** specifies how many instances of an entity relate to one instance of another entity



ER DIAGRAM FOR NEWSPAPER AGENCY AUTOMATION SOFTWARE



DATA FLOW DIAGRAMS (DFDs)

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination.

Data flowcharts can range from simple, even hand-drawn process overviews, to in-depth, multi-level DFDs that dig progressively deeper into how the data is handled. They can be used to analyse an existing system or model a new one.

Symbols and notations used for DFDs

Two common systems of symbols are named after their creators:

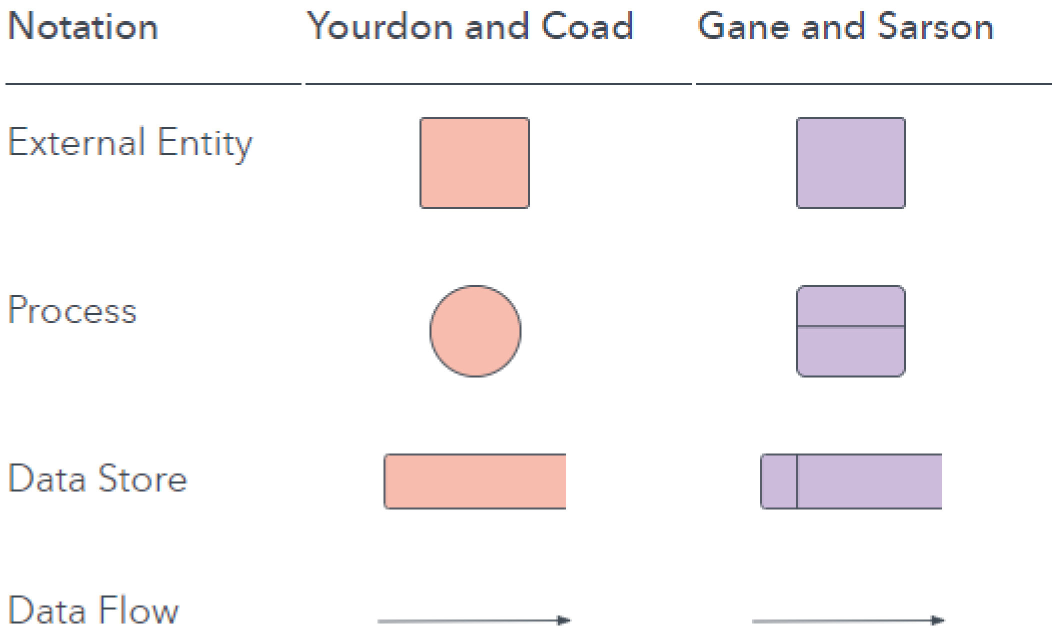
- Yourdon and Coad
- Yourdon and DeMarco

1. **External entity:** an outside system that sends or receives data, communicating with the system being diagrammed. They are the sources and destinations of information entering or leaving the system. They might be an outside organization or person, a computer system or a business system. They are also known as terminators, sources and sinks or actors. They are typically drawn on the edges of the diagram.

2. **Process:** any process that changes the data, producing an output. It might perform computations, or sort data based on logic, or direct the data flow based on business rules. A short label is used to describe the process, such as “Submit payment.”

3. **Data store:** files or repositories that hold information for later use, such as a database table or a membership form. Each data store receives a simple label, such as “Orders.”

4. **Data flow:** the route that data takes between the external entities, processes and data stores. It portrays the interface between the other components and is shown with arrows, typically labelled with a short data name, like “Billing details.”



DFD levels and layers

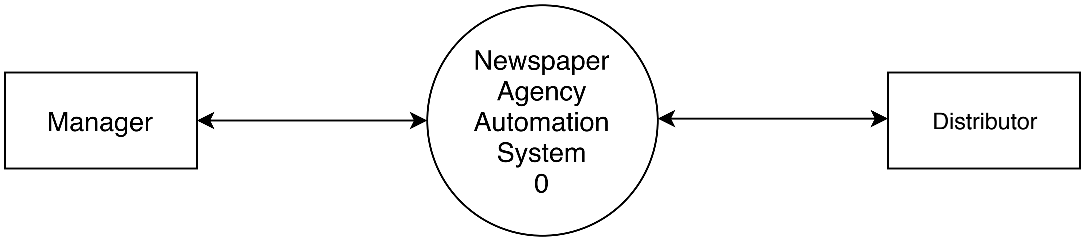
A data flow diagram can dive into progressively more detail by using levels and layers, zeroing in on a particular piece. DFD levels are numbered 0, 1 or 2, and occasionally go to even Level 3 or beyond. The necessary level of detail depends on the scope of what we are trying to accomplish.

- DFD Level 0 is also called a Context Diagram. It's a basic overview of the whole system or process being analysed or modelled. It's designed to be an at-a-glance view, showing the system as a single high-level process, with its relationship to external entities. It should be easily understood by a wide audience, including stakeholders, business analysts, data analysts and developers.

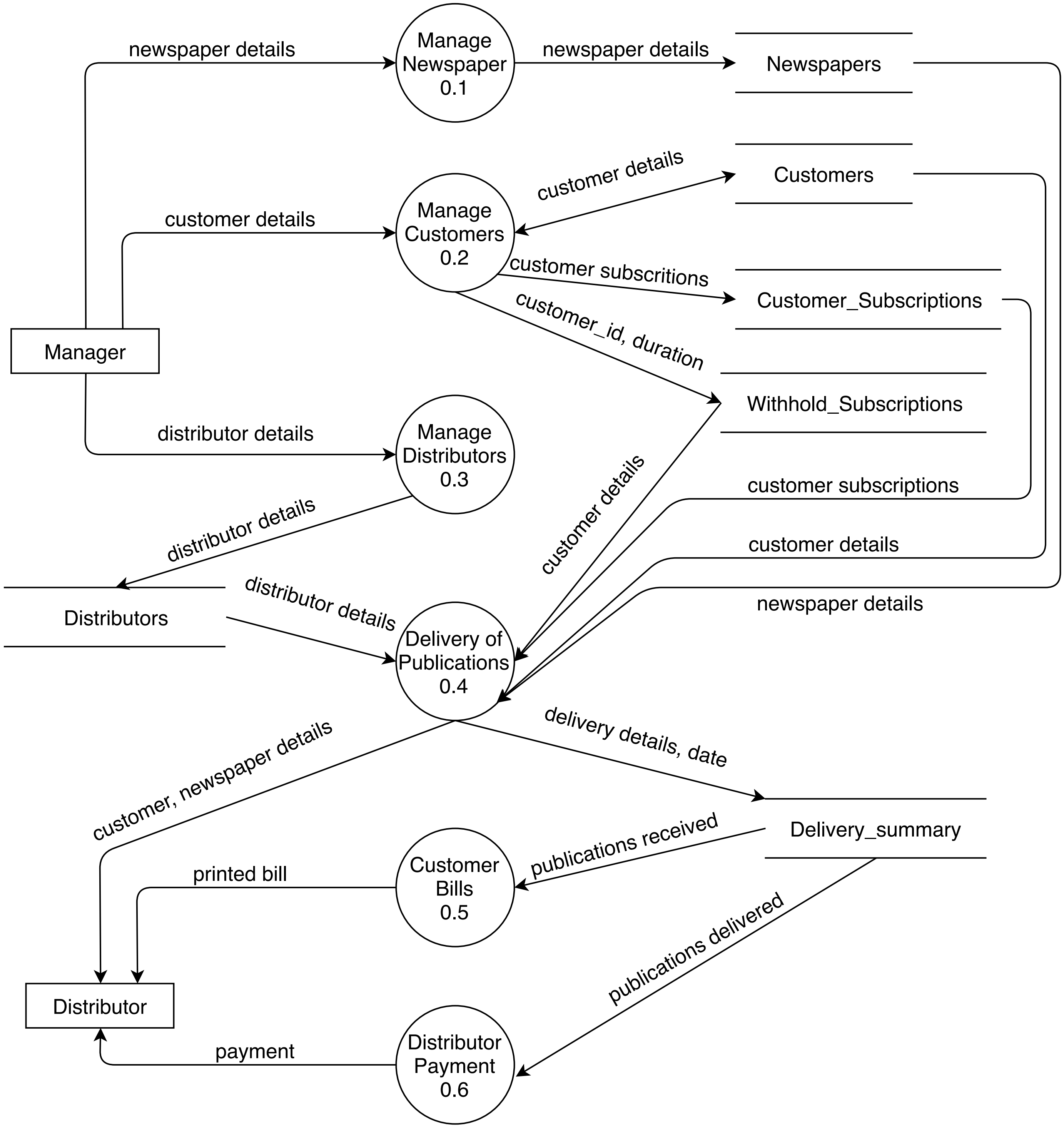
- DFD Level 1 provides a more detailed breakout of pieces of the Context Level Diagram. You will highlight the main functions carried out by the system, as you break down the high-level process of the Context Diagram into its sub processes.

- DFD Level 2 then goes one step deeper into parts of Level 1. It may require more text to reach the necessary level of detail about the system's functioning.

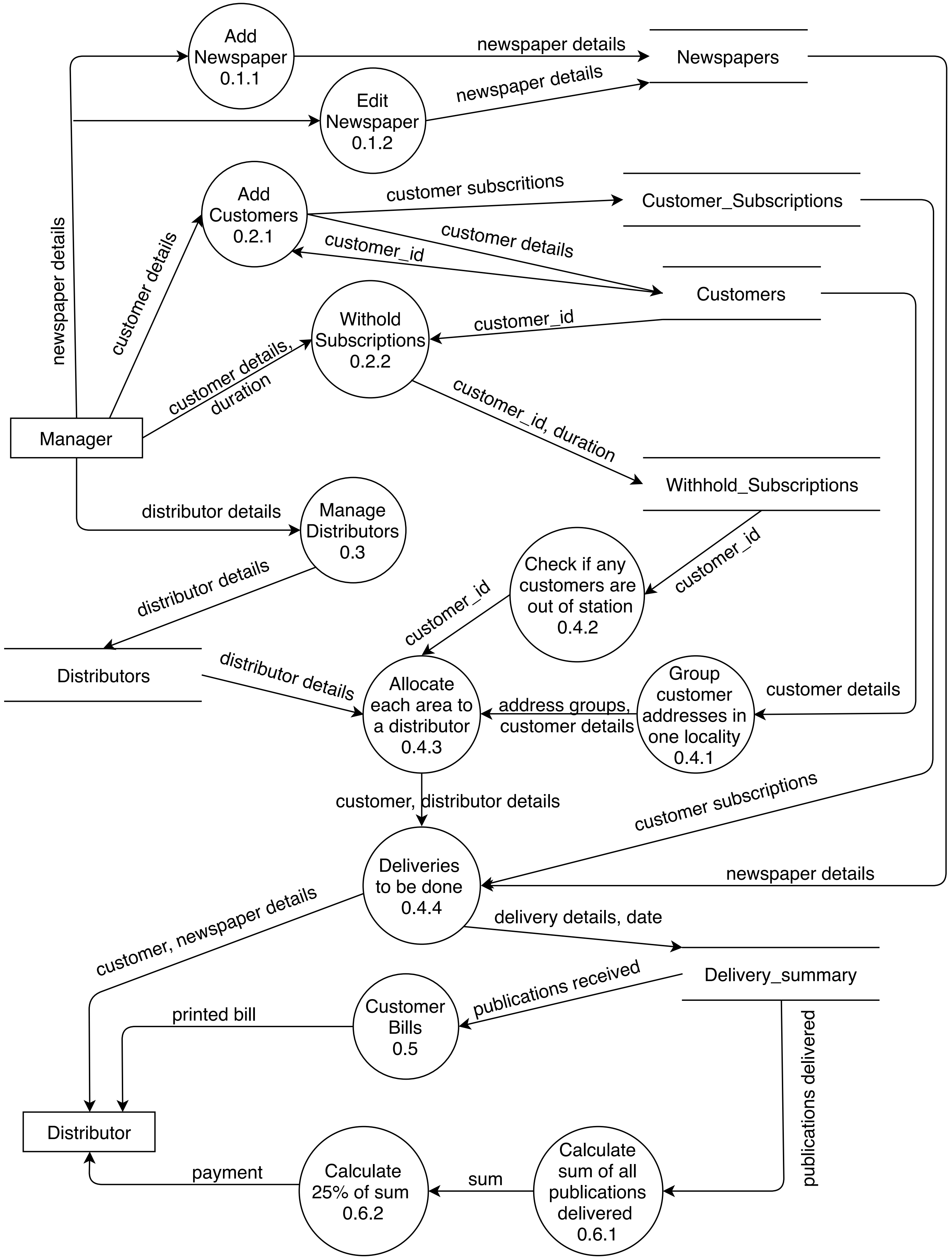
DATAFLOW DIAGRAM (LEVEL 0) FOR NEWSPAPER AGENCY AUTOMATION SYSTEM



DATAFLOW DIAGRAM (LEVEL 1) FOR NEWSPAPER AGENCY AUTOMATION SOFTWARE



DATAFLOW DIAGRAM (LEVEL 2) FOR NEWSPAPER AGENCY AUTOMATION SOFTWARE



USE CASE DIAGRAMS

In the Unified Modeling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialized symbols and connectors. An effective use case diagram can help your team discuss and represent:

- Scenarios in which your system or application interacts with people, organizations, or external systems
- Goals that your system or application helps those entities (known as actors) achieve
- The scope of your system

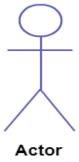
Use Case Diagram objects

Use case diagrams consist of 4 objects.

- Actor
- Use case
- System
- Package

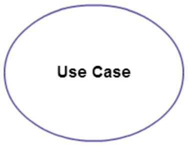
Actor

Actor in a use case diagram is any entity that performs a role in one given system. This could be a person, organization or an external system and usually drawn like skeleton shown below.



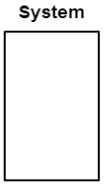
Use Case

A use case represents a function or an action within the system. It’s drawn as an oval and named with the function.



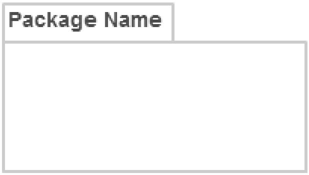
System

The system is used to define the scope of the use case and drawn as a rectangle. This an optional element but useful when you're visualizing large systems. For example, you can create all the use cases and then use the system object to define the scope covered by your project. Or you can even use it to show the different areas covered in different releases.



Package

The package is another optional element that is extremely useful in complex diagrams. Similar to class diagrams, packages are used to group together use cases. They are drawn like the image shown below.

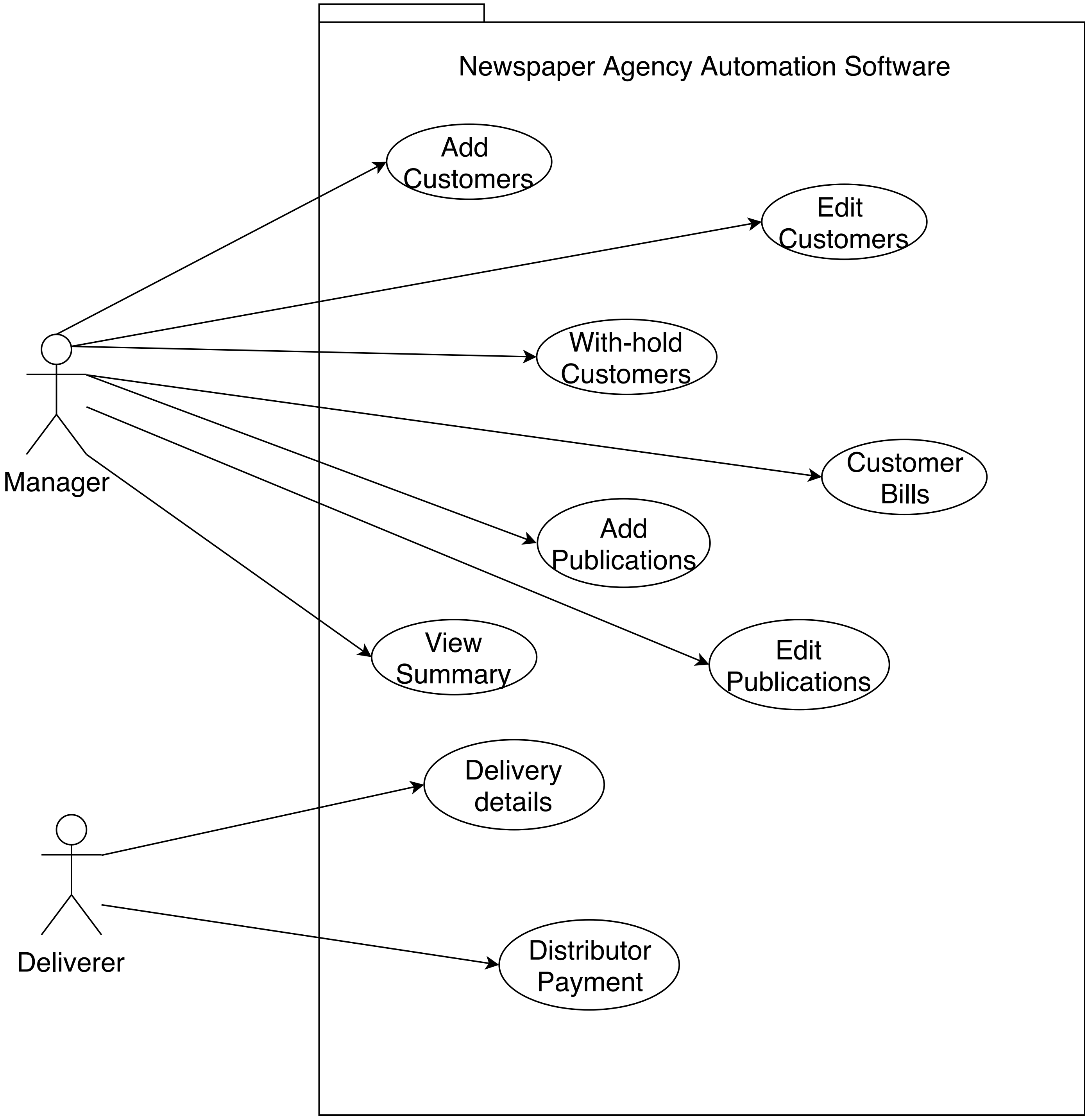


Relationships in Use Case Diagrams

There are five types of relationships in a use case diagram. They are

- Association between an actor and a use case
- Generalization of an actor
- Extend relationship between two use cases
- Include relationship between two use cases
- Generalization of a use case

USE CASE DIAGRAM FOR NEWSPAPER AGENCY AUTOMATION SOFTWARE



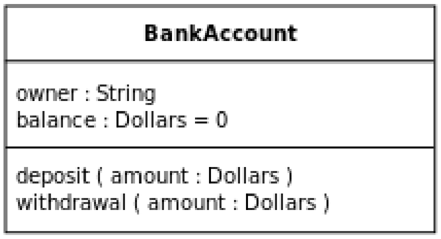
CLASS DIAGRAMS

Class diagrams are one of the most useful types of diagrams in UML as they clearly map out the structure of a particular system by modeling its classes, attributes, operations, and relationships between objects.

Basic components of a class diagram

The standard class diagram is composed of three sections:

- **Upper section**: Contains the name of the class. This section is always required, whether you are talking about the classifier or an object.
- **Middle section**: Contains the attributes of the class. Use this section to describe the qualities of the class. This is only required when describing a specific instance of a class.
- **Bottom section**: Includes class operations (methods). Displayed in list format, each operation takes up its own line. The operations describe how a class interacts with data.



BankAccount

owner : String
balance : Dollars = 0

deposit (amount : Dollars)
withdrawal (amount : Dollars)

Member access modifiers

All classes have different access levels depending on the access modifier (visibility). Here are the access levels with their corresponding symbols:

- Public (+)
- Private (-)
- Protected (#)
- Package (~)
- Derived (/)
- Static (underlined)

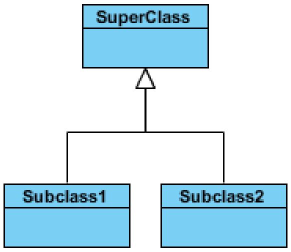
Class Relationships

A class may be involved in one or more relationships with other classes. A relationship can be one of the following types

Relationship Type **Graphical Representation**

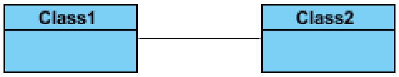
Inheritance (or Generalization):

- Represents an "is-a" relationship.
- SubClass1 and SubClass2 are specializations of Super Class.
- A solid line with a hollow arrowhead that point from the child to the parent class



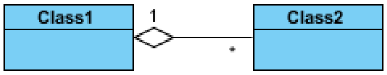
Simple Association:

- A structural link between two peer classes.
- There is an association between Class1 and Class2
- A solid line connecting two classes



Aggregation:

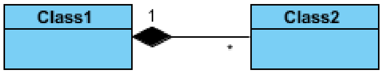
- Many instances (denoted by the *) of Class2 can be associated with Class1.
- A solid line with a unfilled diamond at the association end connected to the class of composite



Composition:

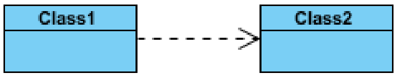
A special type of aggregation where parts are destroyed when the whole is destroyed.

- Objects of Class2 live and die with Class1.
- Class2 cannot stand by itself.
- A solid line with a filled diamond at the association connected to the class of composite

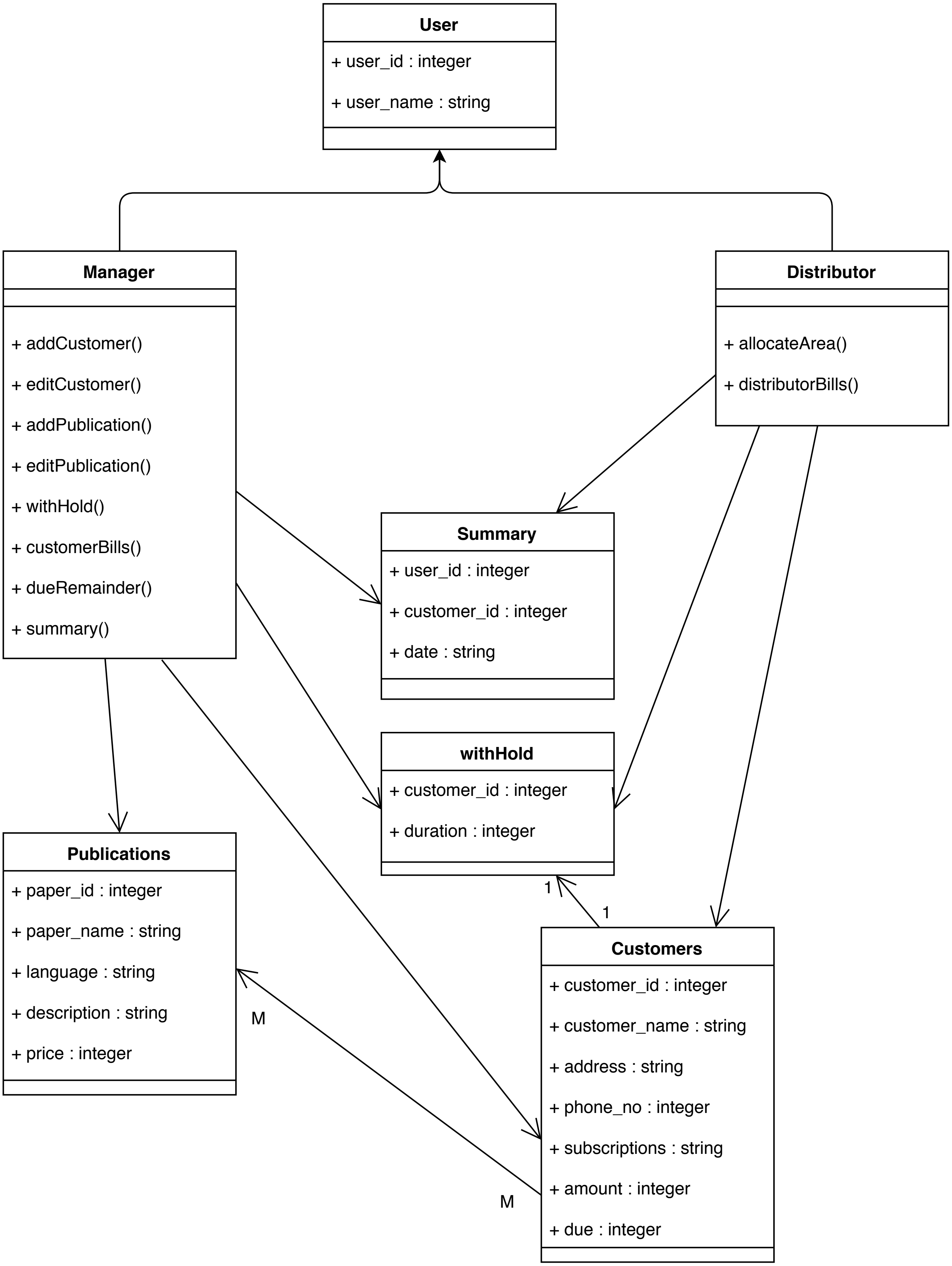


Dependency:

- Exists between two classes if changes to the definition of one may cause changes to the other (but not the other way around).
- A dashed line with an open arrow



CLASS DIAGRAM FOR NEWSPAPER AGENCY AUTOMATION SOFTWARE



OBJECT DIAGRAM

Object is an instance of a particular moment in runtime, including objects and data values. A static UML object diagram is an instance of a class diagram; it shows a snapshot of the detailed state of a system at a point in time, thus an object diagram encompasses objects and their relationships at a point in time. It may be considered a special case of a class diagram or a communication diagram.

Purpose of Object Diagram

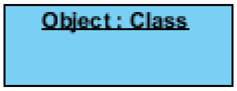
The use of object diagrams is fairly limited, mainly to show examples of data structures.

- During the analysis phase of a project, you might create a class diagram to describe the structure of a system and then create a set of object diagrams as test cases to verify the accuracy and completeness of the class diagram.
- Before you create a class diagram, you might create an object diagram to discover facts about specific model elements and their links, or to illustrate specific examples of the classifiers that are required.

Basic Object Diagram Symbols and Notations

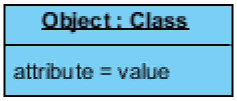
Object Names:

- Every object is actually symbolized like a rectangle, that offers the name from the object and its class underlined as well as divided with a colon.

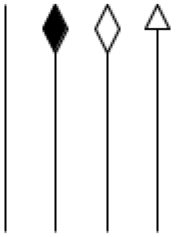


Object Attributes:

- Similar to classes, you are able to list object attributes inside a separate compartment. However, unlike classes, object attributes should have values assigned for them.

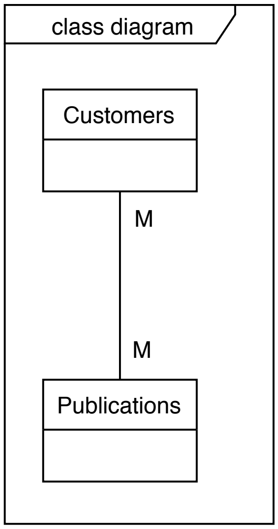


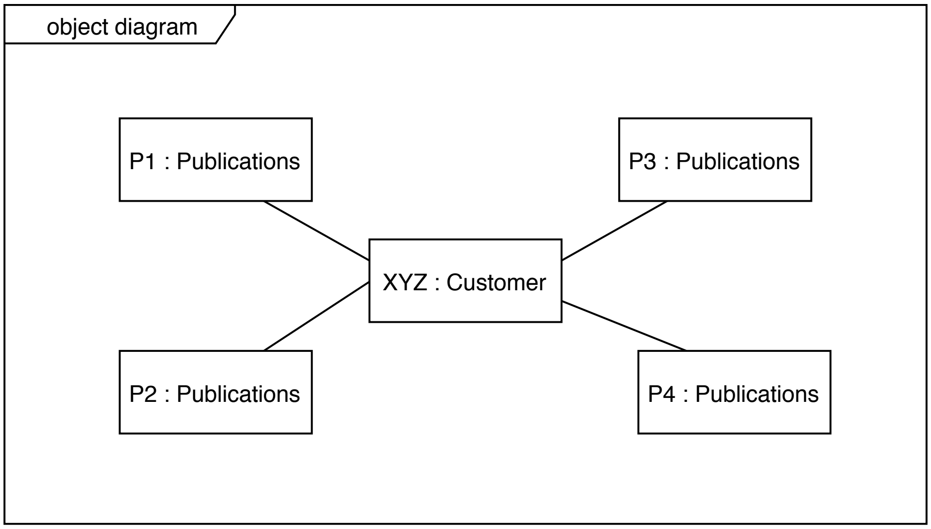
Links:



- Links tend to be instances associated with associations. You can draw a link while using the lines utilized in class diagrams.

OBJECT DIAGRAM FOR NEWSPAPER AGENCY AUTOMATION SOFTWARE



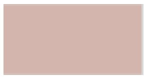
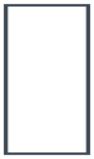
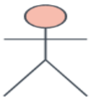
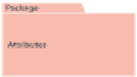
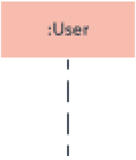
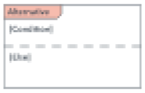


SEQUENCE DIAGRAMS

Sequence diagrams are a popular dynamic modeling solution in UML because they specifically focus on lifelines, or the processes and objects that live simultaneously, and the messages exchanged between them to perform a function before the lifeline ends.

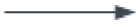
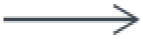
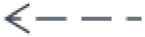
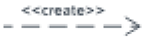
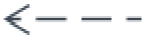
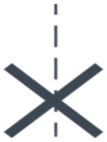
Basic symbols and components

To understand what a sequence diagram is, you should be familiar with its symbols and components. Sequence diagrams are made up of the following icons and elements:

Symbol	Name	Description
	Object symbol	Represents a class or object in UML. The object symbol demonstrates how an object will behave in the context of the system. Class attributes should not be listed in this shape.
	Activation box	Represents the time needed for an object to complete a task. The longer the task will take, the longer the activation box becomes.
	Actor symbol	Shows entities that interact with or are external to the system.
	Package symbol	Used in UML 2.0 notation to contain interactive elements of the diagram. Also known as a frame, this rectangular shape has a small inner rectangle for labeling the diagram.
	Lifeline symbol	Represents the passage of time as it extends downward. This dashed vertical line shows the sequential events that occur to an object during the charted process. Lifelines may begin with a labeled rectangle shape or an actor symbol.
	Alternative symbol	Symbolizes a choice (that is usually mutually exclusive) between two or more message sequences. To represent alternatives, use the labeled rectangle shape with a dashed line inside.

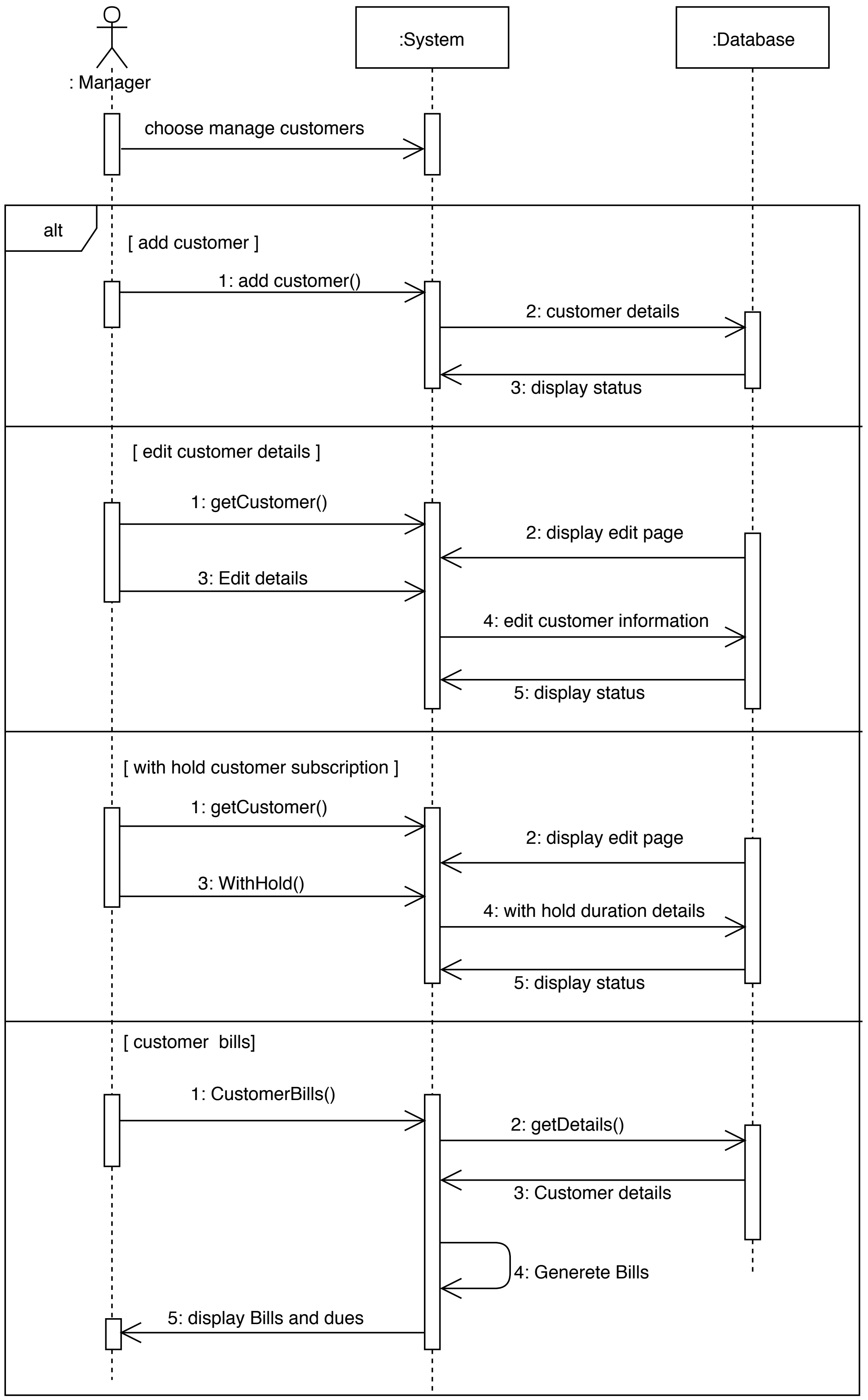
Common message symbols

Use the following arrows and message symbols to show how information is transmitted between objects. These symbols may reflect the start and execution of an operation or the sending and reception of a signal.

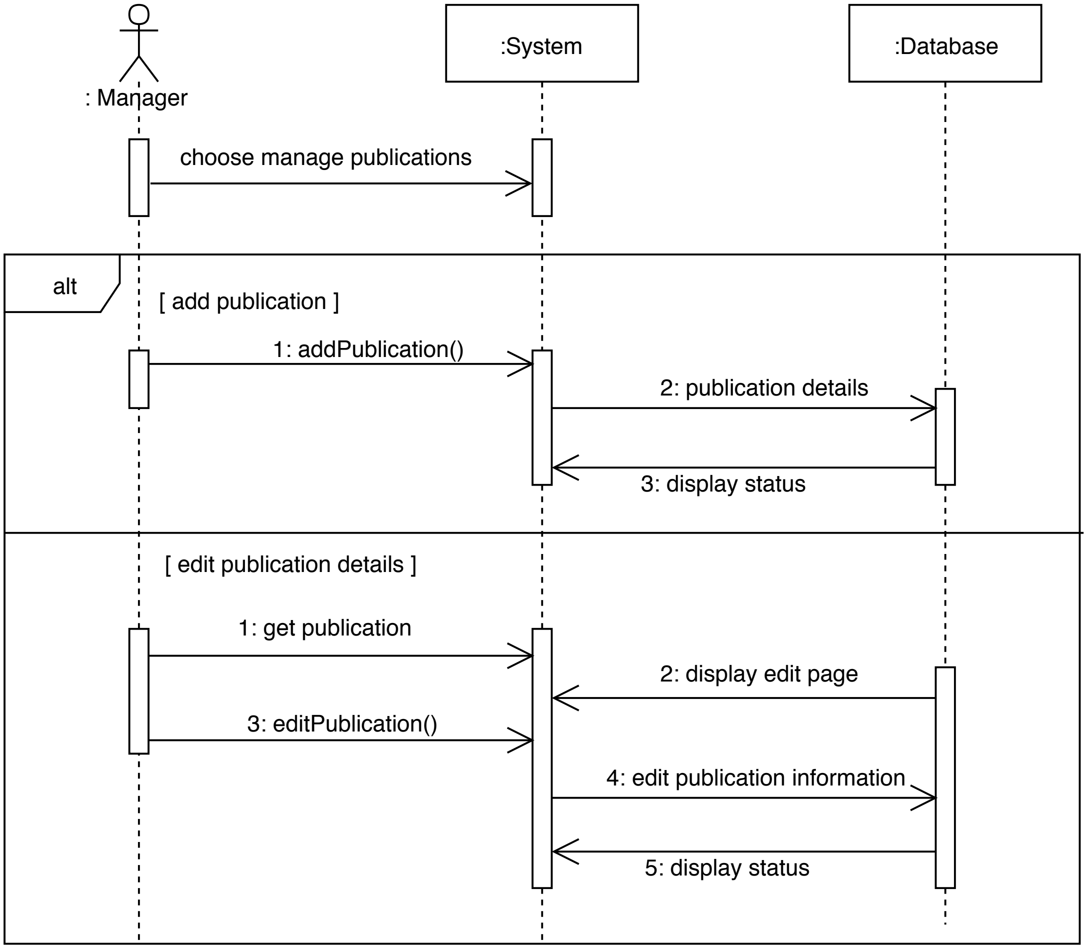
Symbol	Name	Description
	Synchronous message symbol	Represented by a solid line with a solid arrowhead. This symbol is used when a sender must wait for a response to a message before it continues. The diagram should show both the call and the reply.
	Asynchronous message symbol	Represented by a solid line with a lined arrowhead. Asynchronous messages don't require a response before the sender continues. Only the call should be included in the diagram.
	Asynchronous return message symbol	Represented by a dashed line with a lined arrowhead.
	Asynchronous create message symbol	Represented by a dashed line with a lined arrowhead. This message creates a new object.
	Reply message symbol	Represented by a dashed line with a lined arrowhead, these messages are replies to calls.
	Delete message symbol	Represented by a solid line with a solid arrowhead, followed by an X. This message destroys an object.

SEQUENCE DIAGRAM FOR NEWSPAPER AGENCY AUTOMATION SOFTWARE

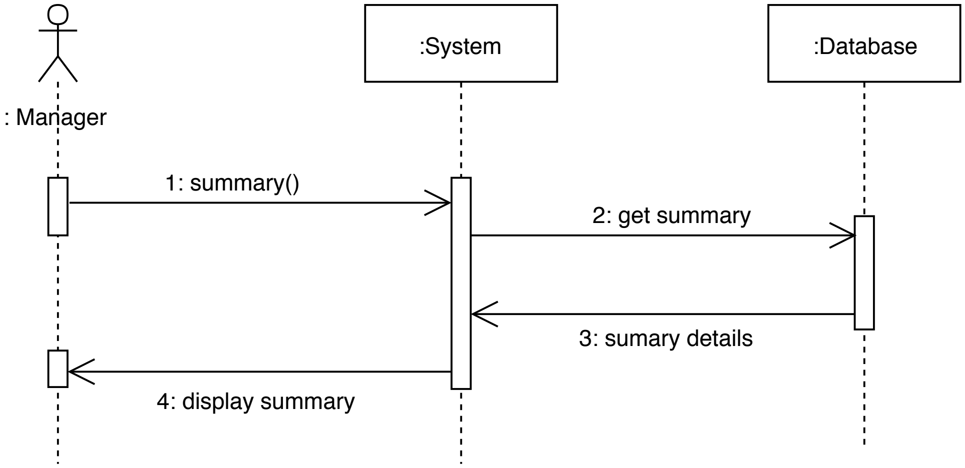
Manage Customers



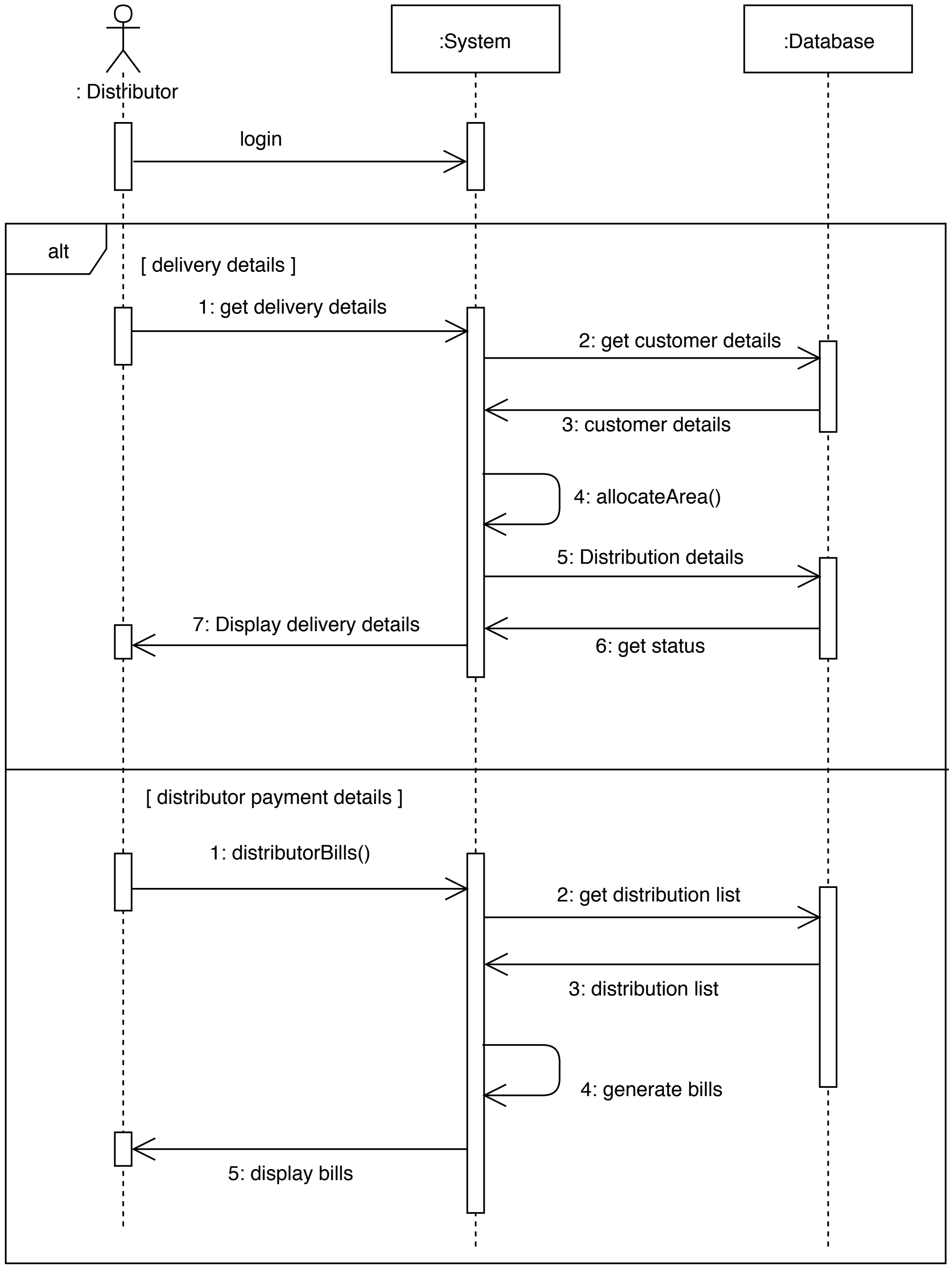
Manage Publications



View Summary of Distributions



Delivery details



COLLABORATION DIAGRAM

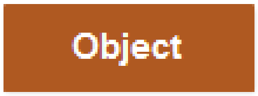
It shows the object organization as seen in the following diagram. In the collaboration diagram, the method call sequence is indicated by some numbering technique. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram.

Method calls are similar to that of a sequence diagram. However, difference being the sequence diagram does not describe the object organization, whereas the collaboration diagram shows the object organization.

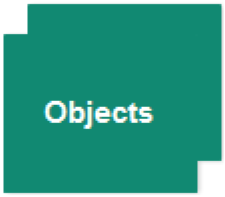
To choose between these two diagrams, emphasis is placed on the type of requirement. If the time sequence is important, then the sequence diagram is used. If organization is required, then collaboration diagram is used.

UML Collaboration Diagram Symbols

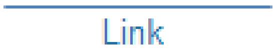
- **Objects** are model elements that represent instances of a class or of classes.



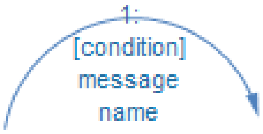
- **Multi-object** represents a set of lifeline instances.



- **Association role** is optional and suppressible.

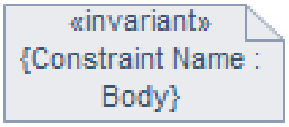


- **Association role** is optional and suppressible.

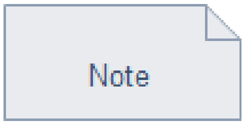


- **Delegation** is like inheritance done manually through object composition.

- **Link to self** is used to link the objects that fulfill more than one role.

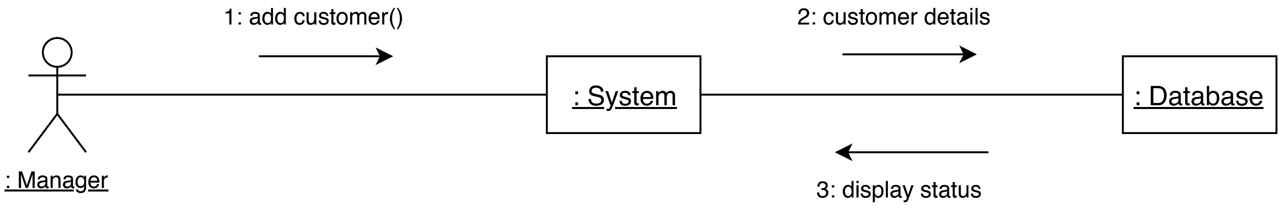


- **Constraint** is an extension mechanism that enables you to refine the semantics of a UML model element.

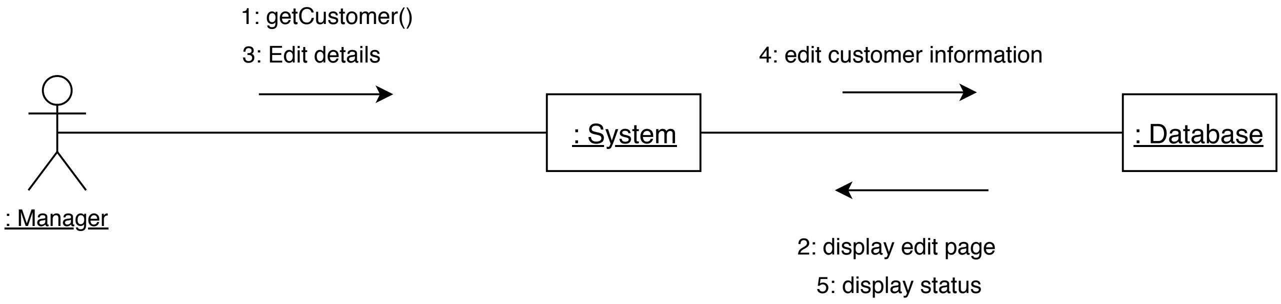


COLLABORATION DIAGRAM FOR NEWSPAPER AGENCY AUTOMATION SOFTWARE

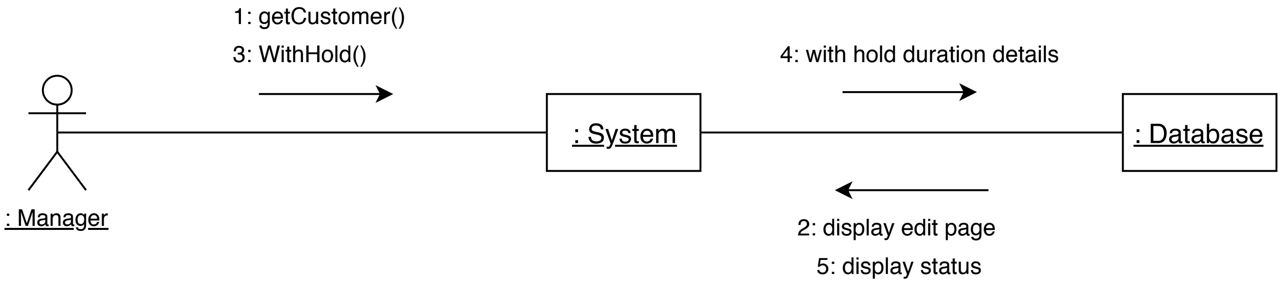
Add Customers



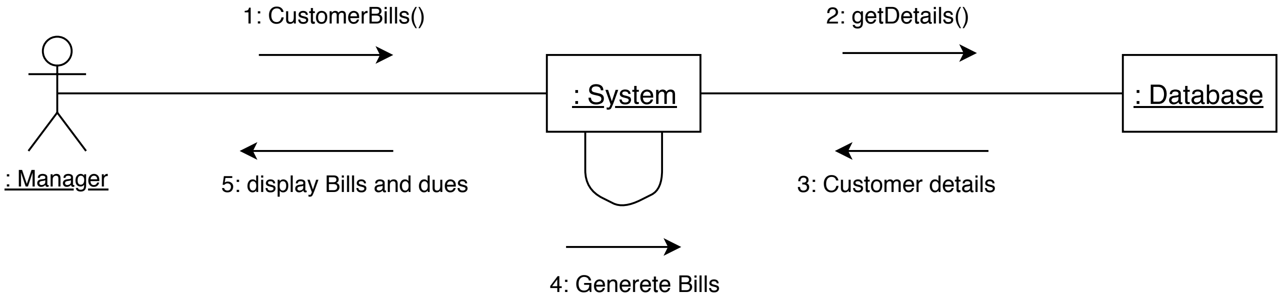
Edit Customer details



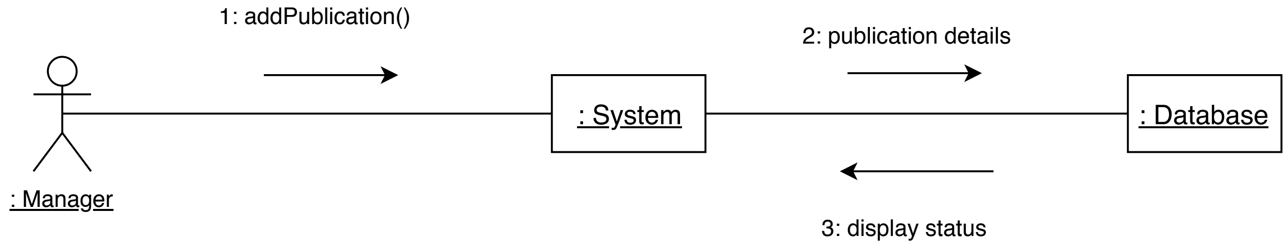
Withhold Customers



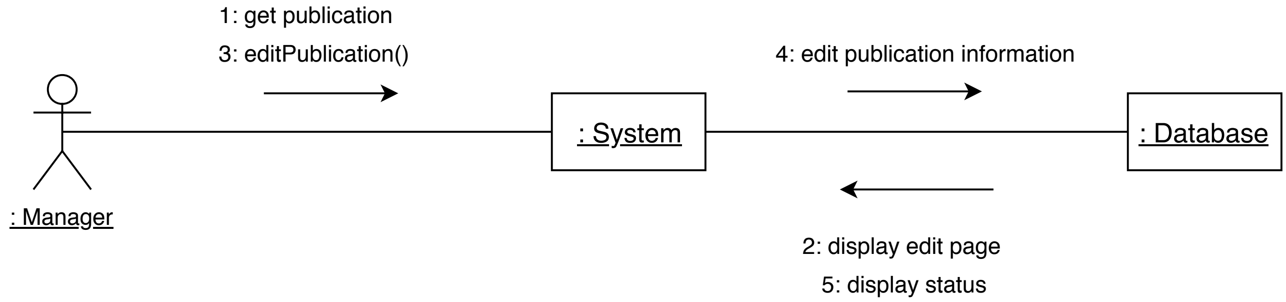
Customer Bills



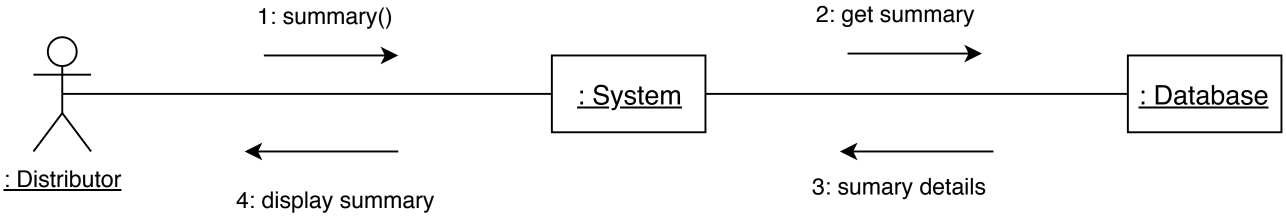
Add Publications



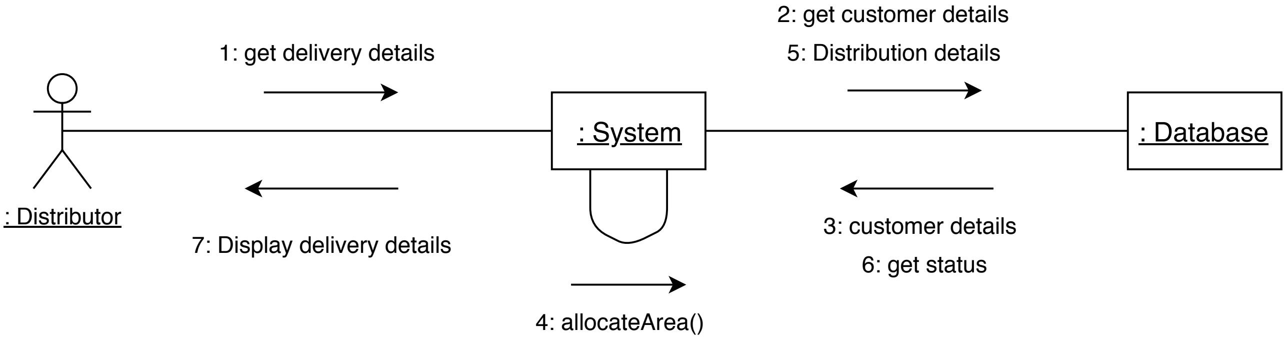
Edit Publication details



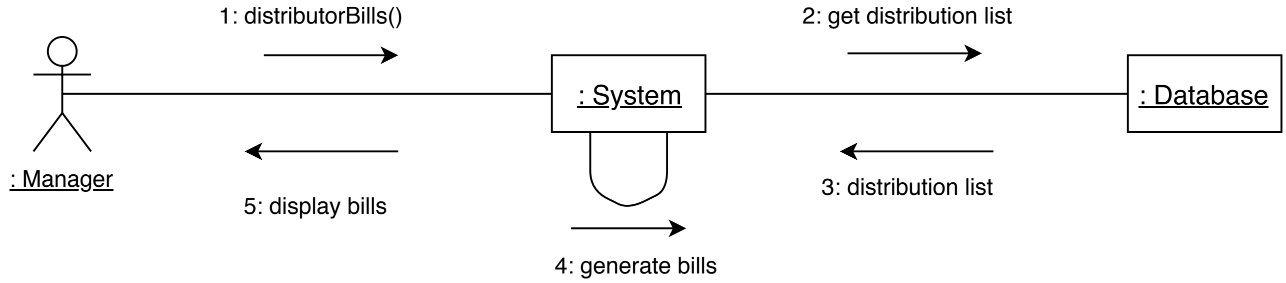
View Summary



Delivery details



Distributor payment

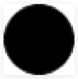


STATE CHART DIAGRAM

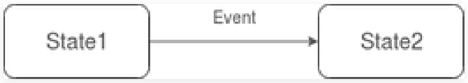
State chart diagram is one of the five UML diagrams used to model the dynamic nature of a system. They define different states of an object during its lifetime and these states are changed by events. State chart diagrams are useful to model the reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

Basic components of a state chart diagram

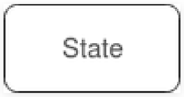
- **Initial state –** We use a black filled circle represent the initial state of a System or a class.



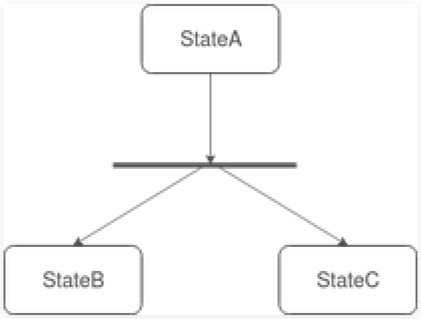
- **Transition –** We use a solid arrow to represent the transition or change of control from one state to another. The arrow is labeled with the event which causes the change in state.



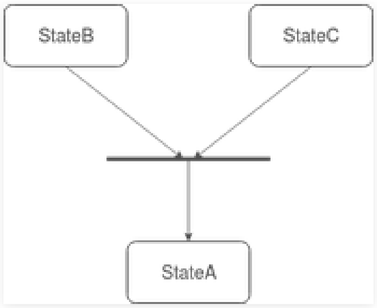
- **State –** We use a rounded rectangle to represent a state. A state represents the conditions or circumstances of an object of a class at an instant of time.



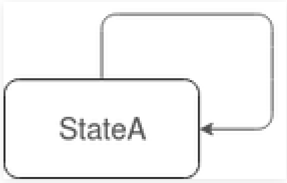
- **Fork –** We use a rounded solid rectangular bar to represent a Fork notation with incoming arrow from the parent state and outgoing arrows towards the newly created states. We use the fork notation to represent a state splitting into two or more concurrent states



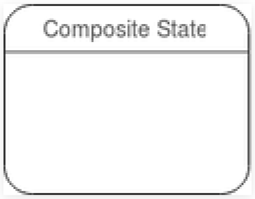
- **Join –** We use a rounded solid rectangular bar to represent a Join notation with incoming arrows from the joining states and outgoing arrow towards the common goal state. We use the join notation when two or more states concurrently converge into one on the occurrence of an event or events.



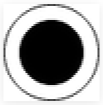
- **Self-transition –** We use a solid arrow pointing back to the state itself to represent a self-transition. There might be scenarios when the state of the object does not change upon the occurrence of an event. We use self-transitions to represent such cases.



- **Composite state –** We use a rounded rectangle to represent a composite state also. We represent a state with internal activities using a composite state

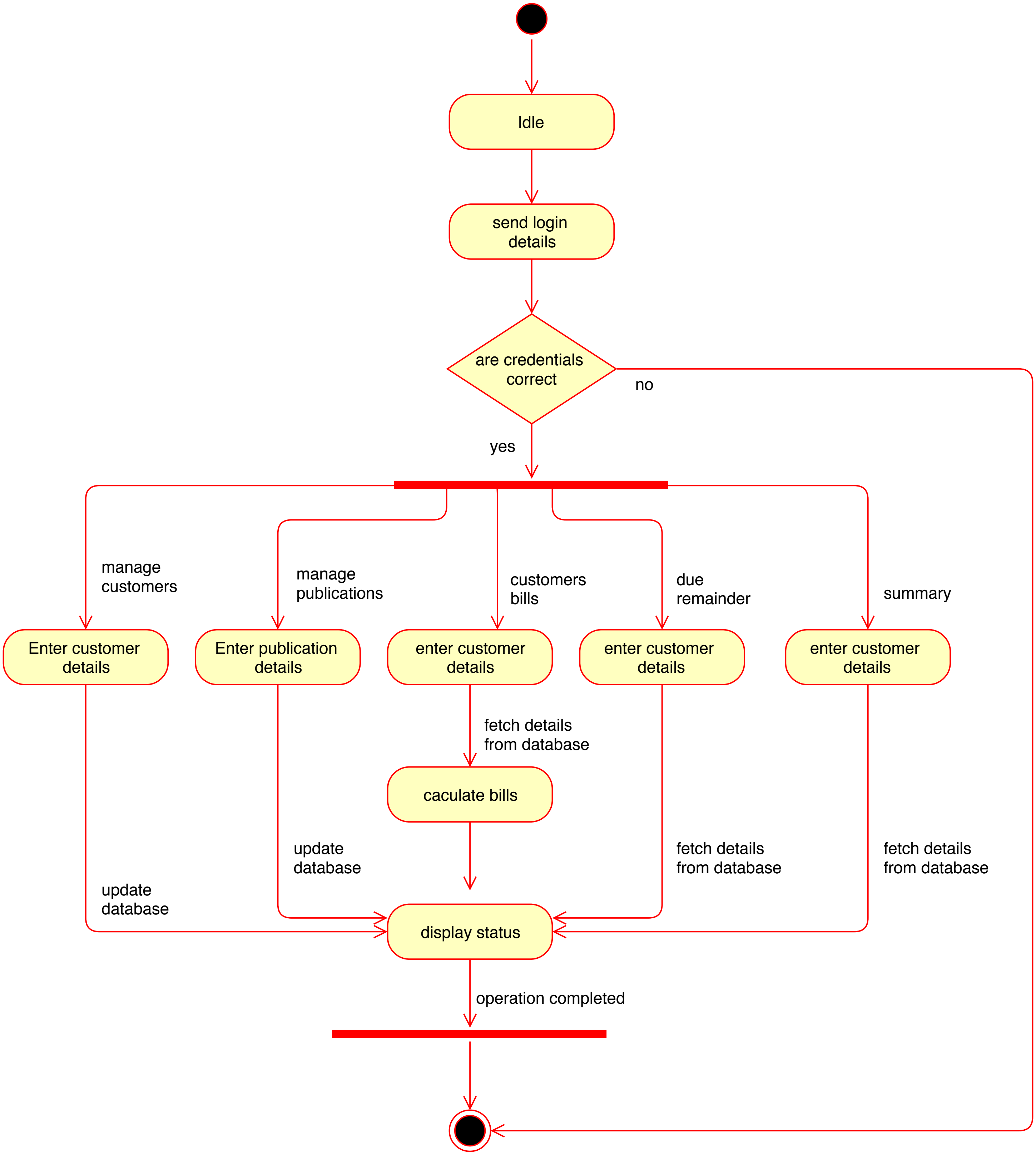


- **Final state –** We use a filled circle within a circle notation to represent the final state in a state machine diagram.

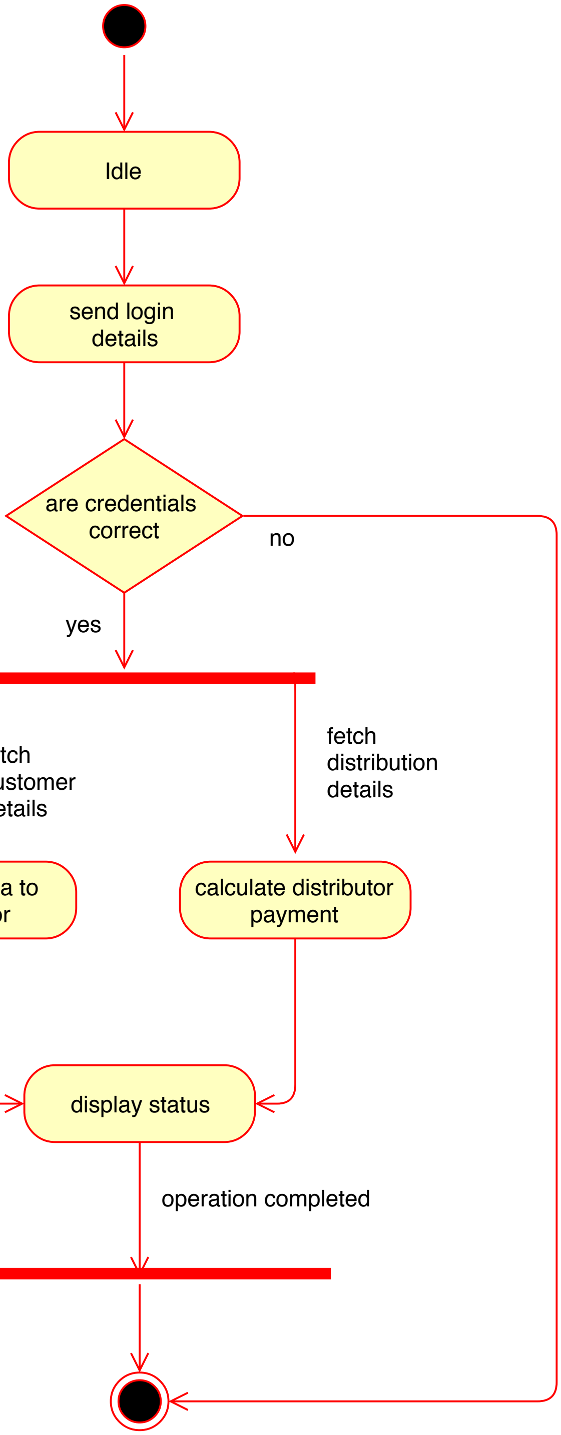


STATECHART DIAGRAM FOR NEWSPAPER AGENCY AUTOMATION SOFTWARE

Statechart diagram for Manager



Statechart diagram for Distributor

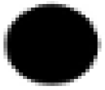
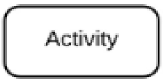
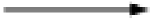
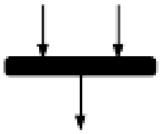
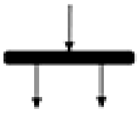
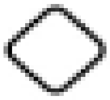


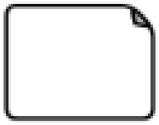
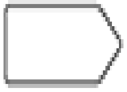
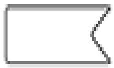
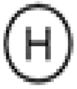
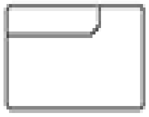
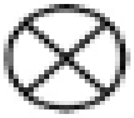
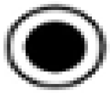
ACTIVITY DIAGRAM

Activity diagram is another important behavioral diagram in UML diagram to describe dynamic aspects of the system. Activity diagram is essentially an advanced version of flow chart that modeling the flow from one activity to another activity.

Activity diagram symbols

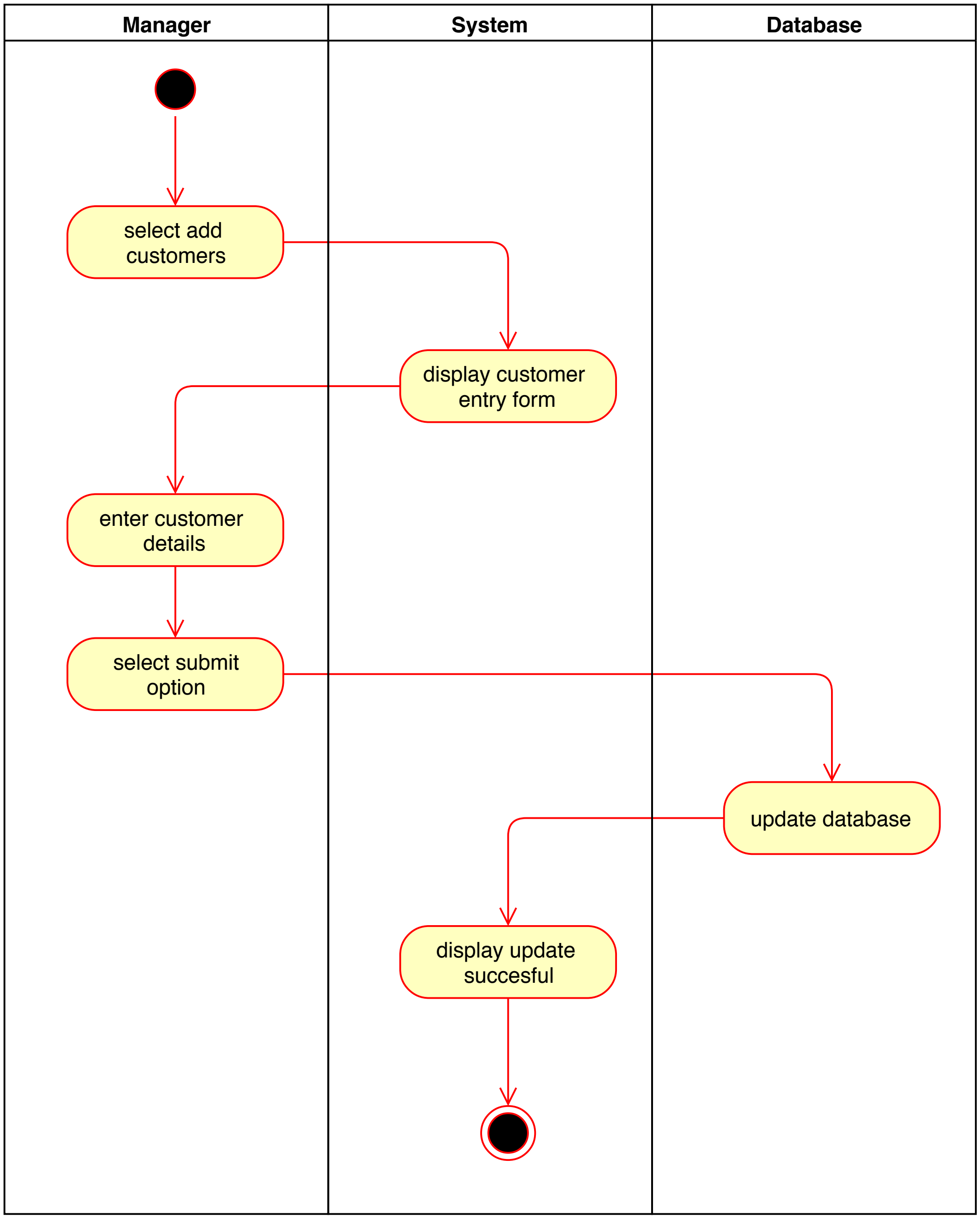
These activity diagram shapes and symbols are some of the most common types you'll find in UML diagrams.

Symbol	Name	Description
	Start symbol	Represents the beginning of a process or workflow in an activity diagram. It can be used by itself or with a note symbol that explains the starting point.
	Activity symbol	Indicates the activities that make up a modeled process. These symbols, which include short descriptions within the shape, are the main building blocks of an activity diagram.
	Connector symbol	Shows the directional flow, or control flow, of the activity. An incoming arrow starts a step of an activity; once the step is completed, the flow continues with the outgoing arrow.
	Joint symbol/ Synchronization bar	Combines two concurrent activities and re-introduces them to a flow where only one activity occurs at a time. Represented with a thick vertical or horizontal line.
	Fork symbol	Splits a single activity flow into two concurrent activities. Symbolized with multiple arrowed lines from a join.
	Decision symbol	Represents a decision and always has at least two paths branching out with condition text to allow users to view options. This symbol represents the branching or merging of various flows with the symbol acting as a frame or container.

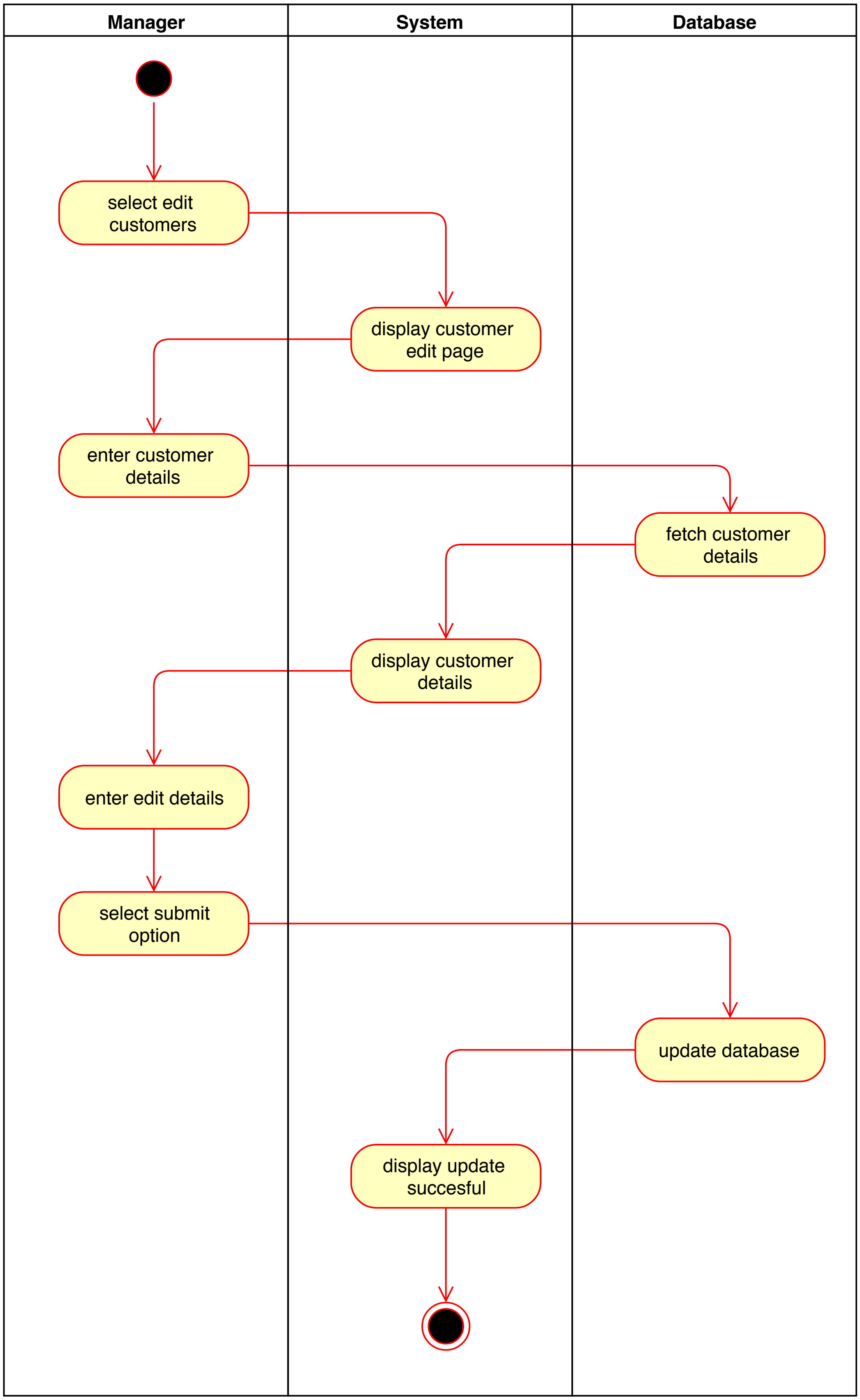
Symbol	Name	Description
	Note symbol	Allows the diagram creators or collaborators to communicate additional messages that don't fit within the diagram itself. Leave notes for added clarity and specification.
	Send signal symbol	Indicates that a signal is being sent to a receiving activity.
	Receive signal symbol	Demonstrates the acceptance of an event. After the event is received, the flow that comes from this action is completed.
	Shallow history pseudo state symbol	Represents a transition that invokes the last active state.
	Option loop symbol	Allows the creator to model a repetitive sequence within the option loop symbol.
	Flow final symbol	Represents the end of a specific process flow. This symbol shouldn't represent the end of all flows in an activity; in that instance, you would use the end symbol. The flow final symbol should be placed at the end of a process in a single activity flow.
[Condition]	Condition text	Placed next to a decision marker to let you know under what condition an activity flow should split off in that direction.
	End symbol	Marks the end state of an activity and represents the completion of all flows of a process.

ACTIVITY DIAGRAM FOR NEWSPAPER AGENCY AUTOMATION SOFTWARE

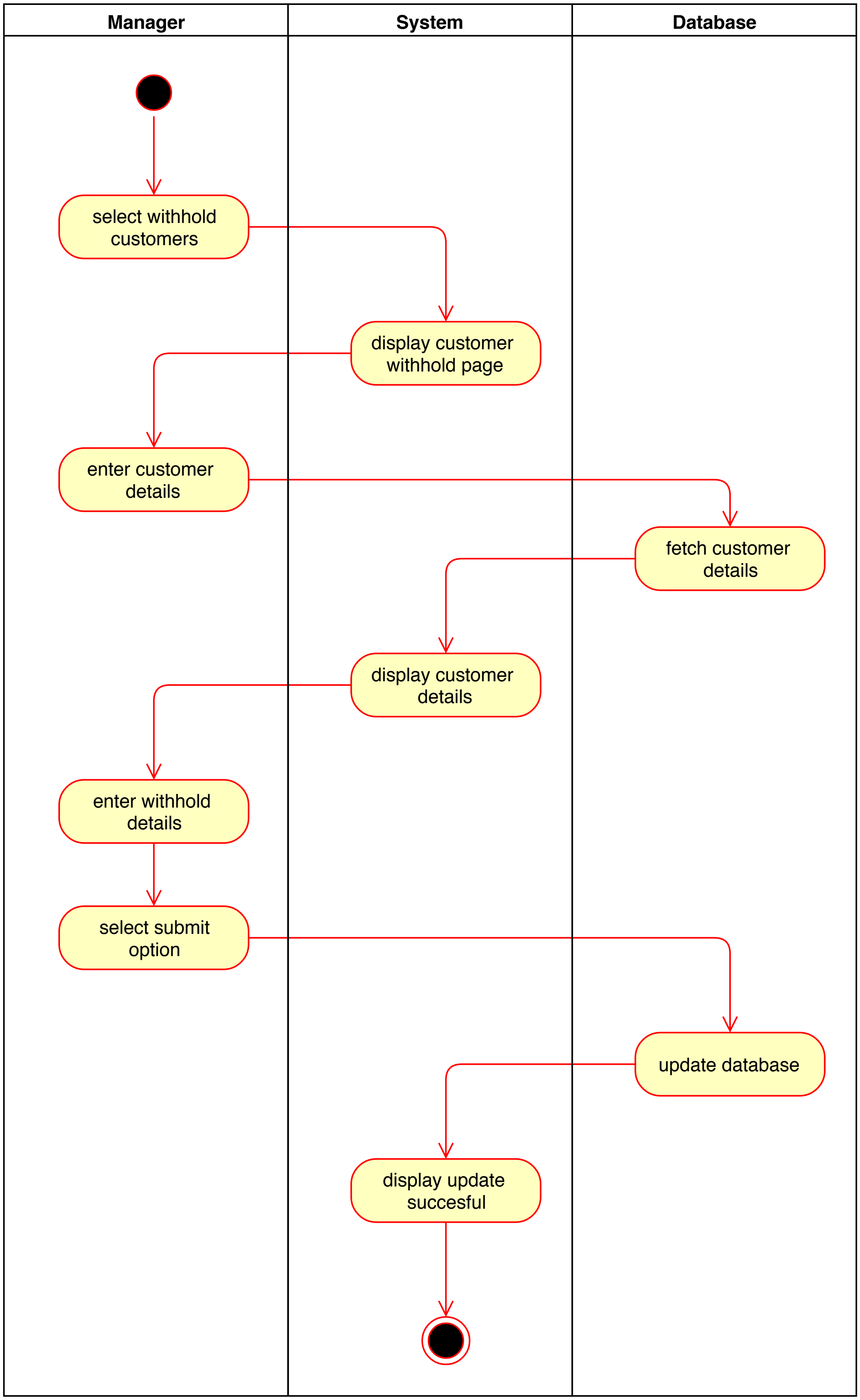
Add Customers



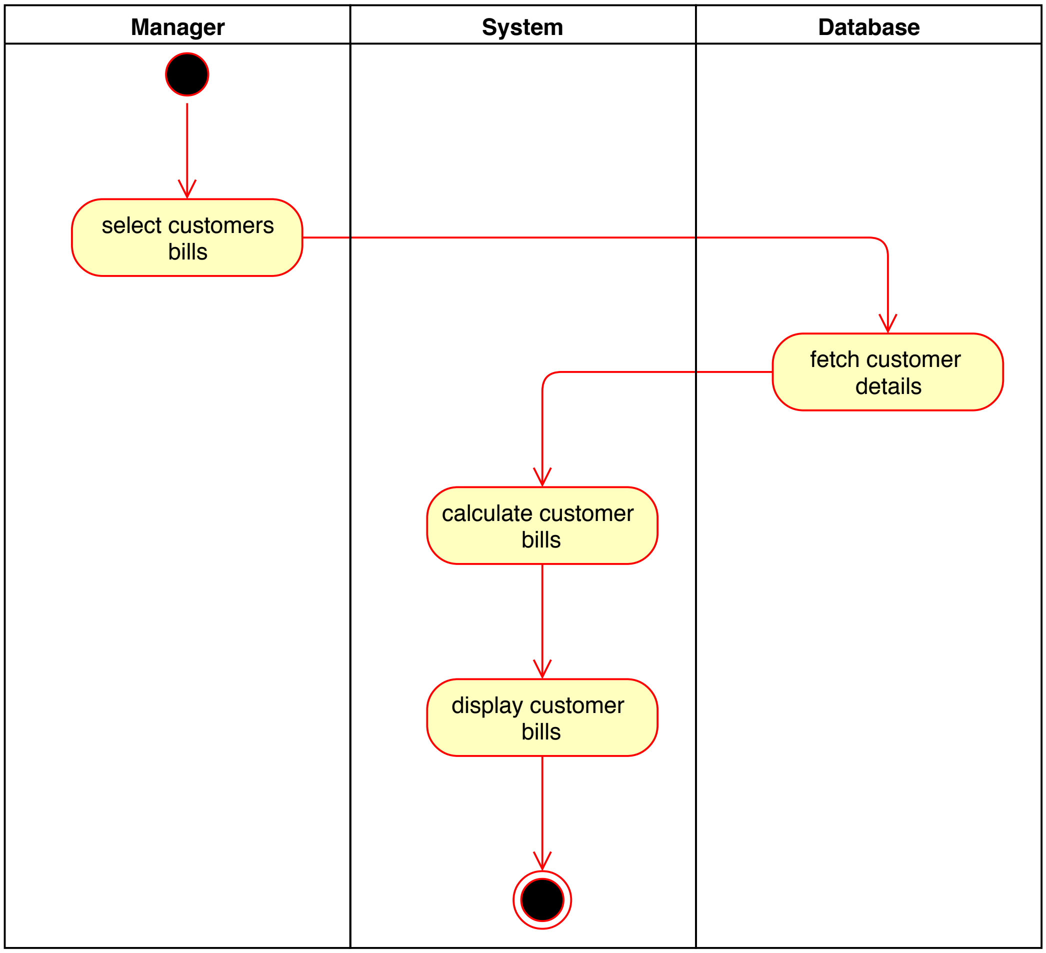
Edit Customer details



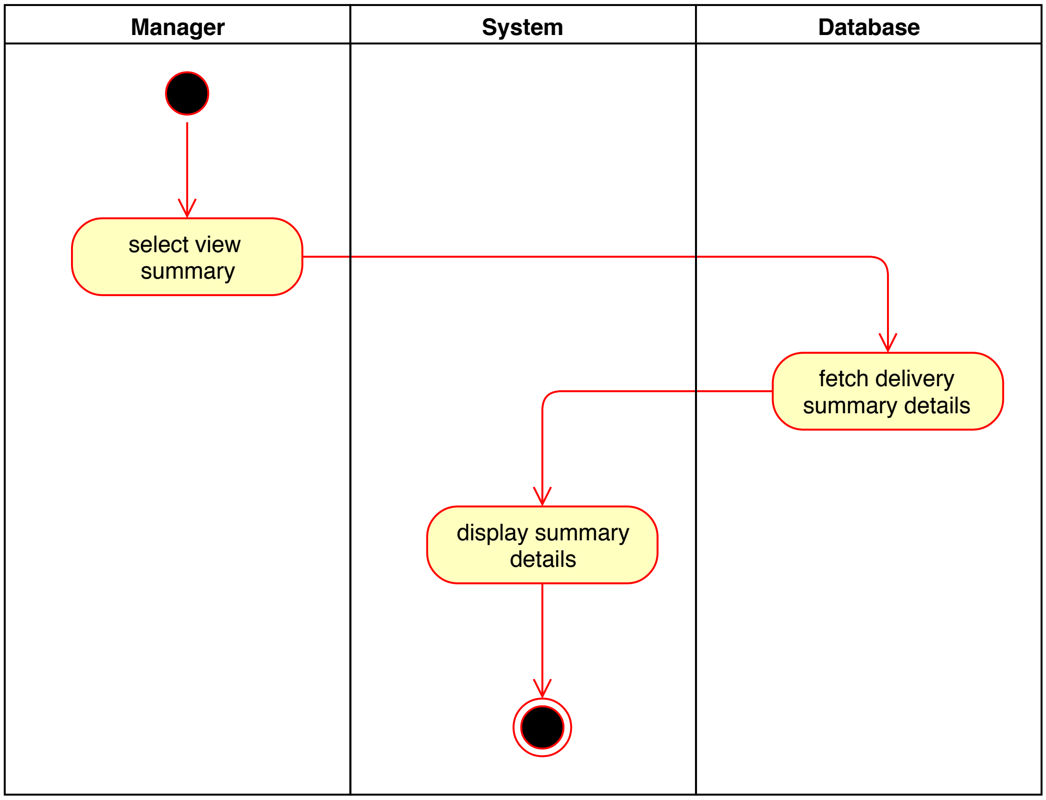
Withhold Customers



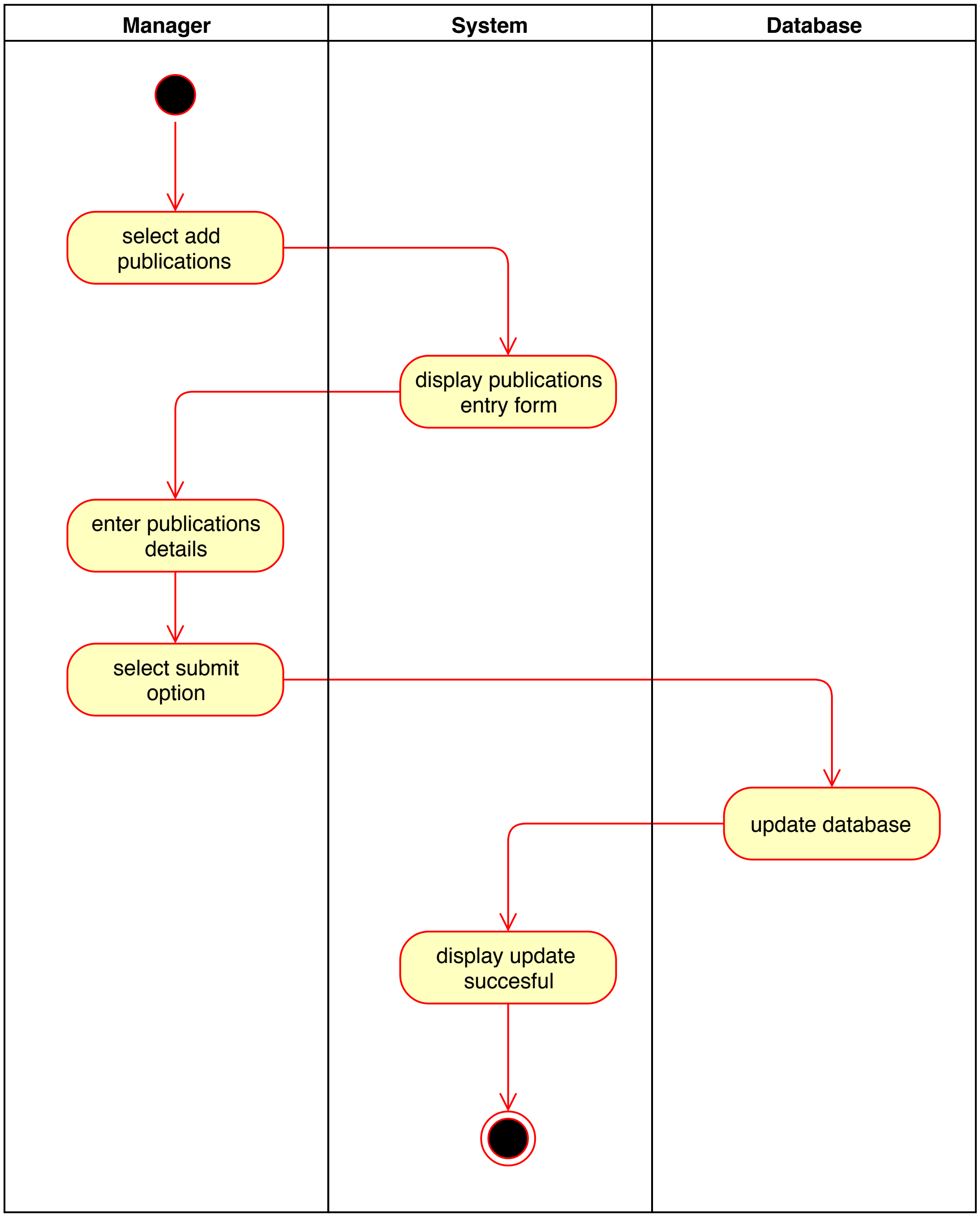
Customer Bills



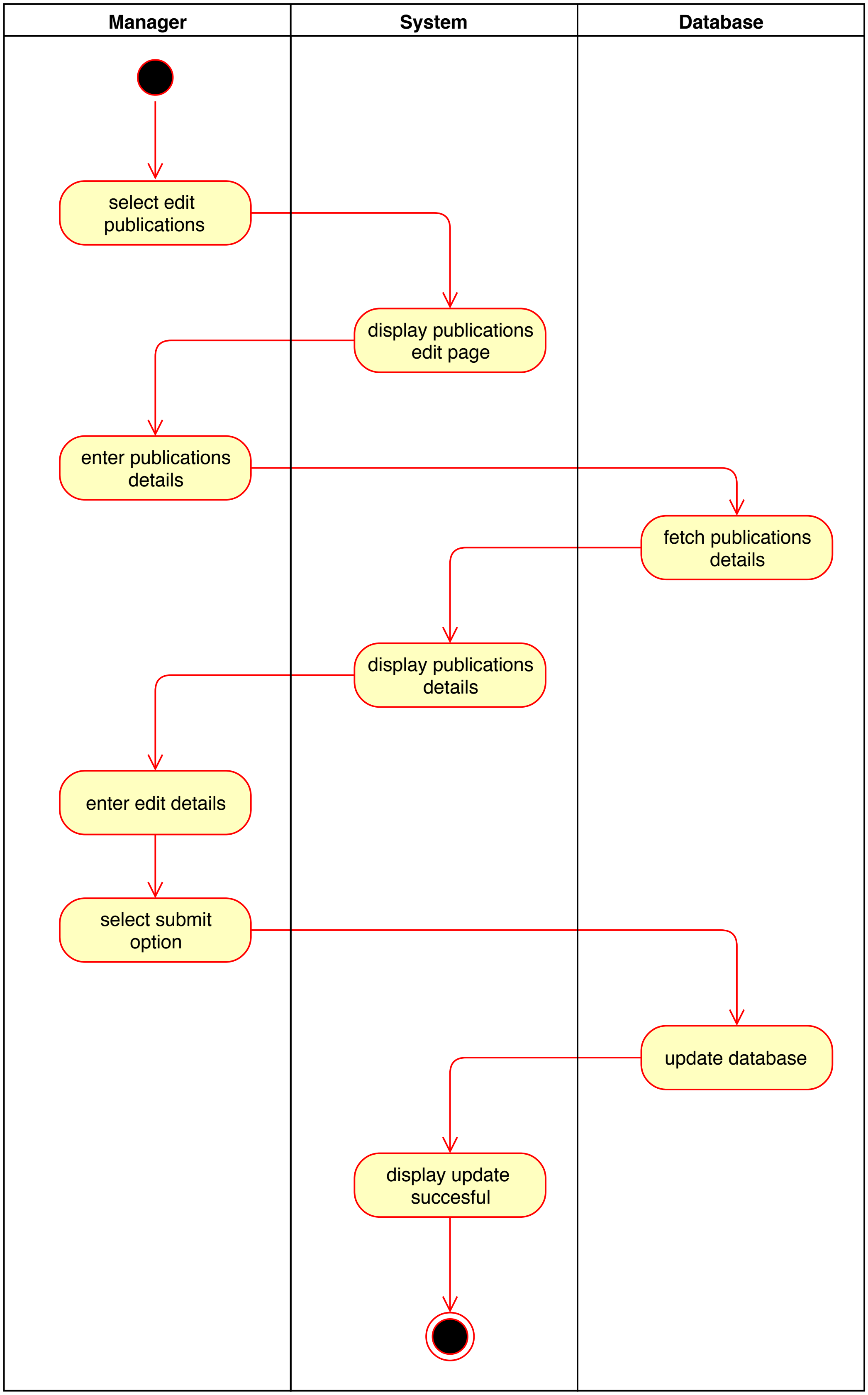
View Summary



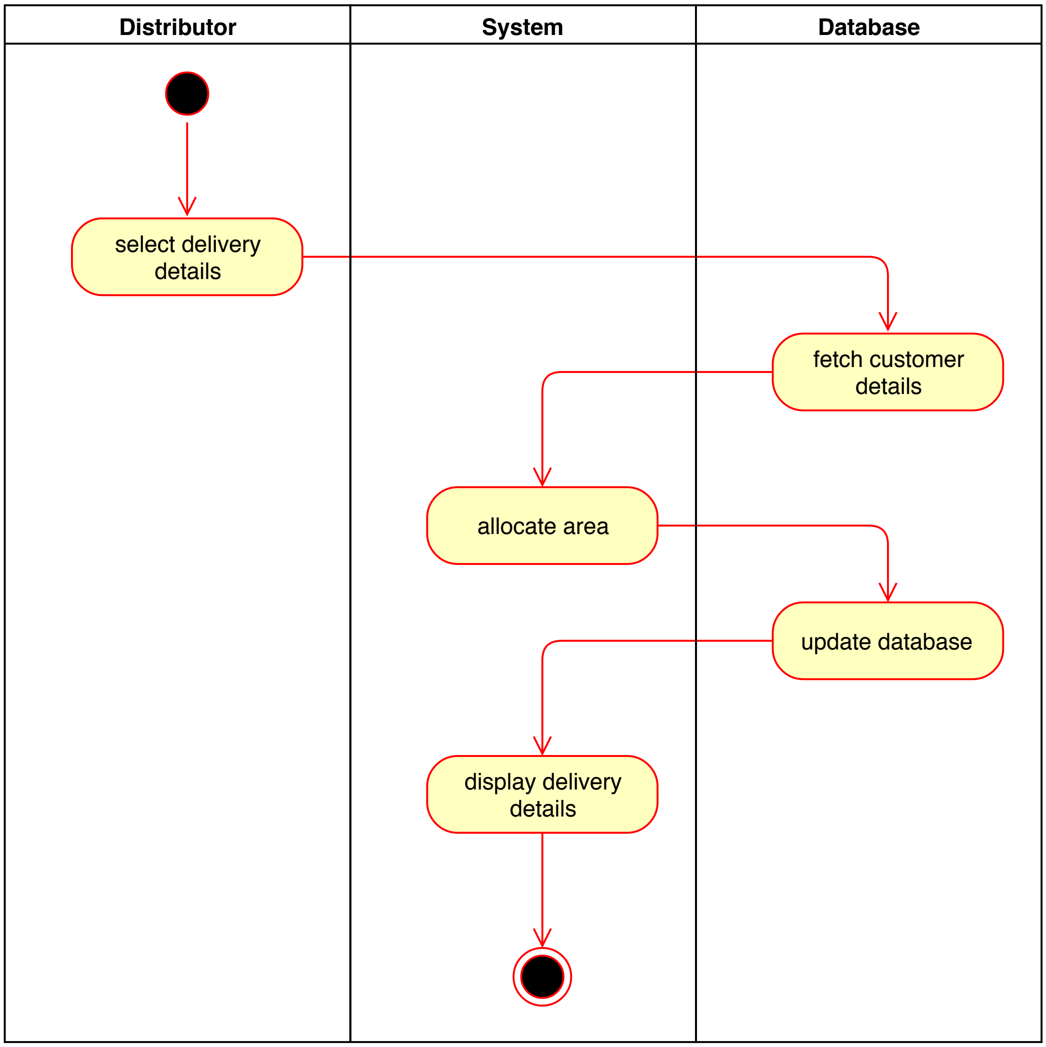
Add Publications



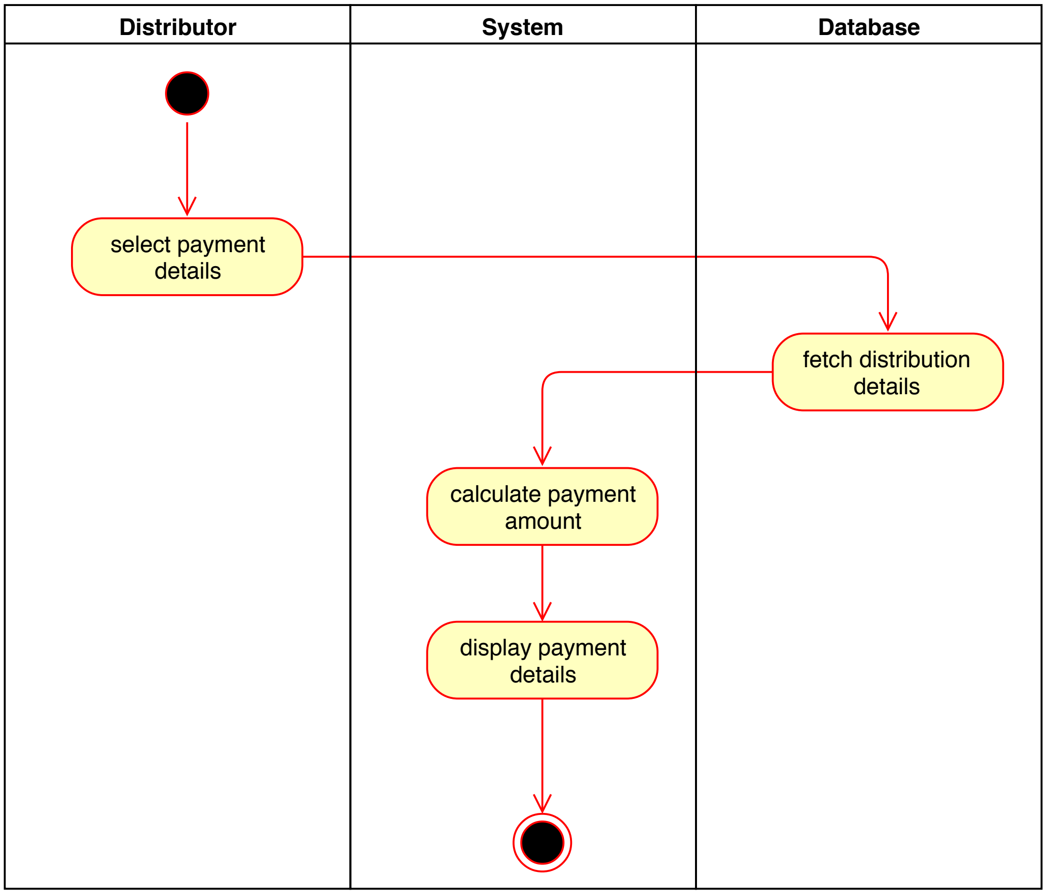
Edit Publication details



Delivery details



Distributor payment



IMPLEMENTATION

Implementation of the functionalities mentioned in SRS document is done using python framework, DjangoRest.

STEPS:

1. New python environment for the project is created.
2. Dependencies – DjangoRest, pandas, fpdf, MySQL required for the project are installed.
3. A database is created in MySQL and linked to the project through the python file ‘settings.py’.
4. The python file ‘models.py’ contains code for creating tables in the database
5. The python file ‘serializers.py’ contains code for serialising the user input.
6. The python file ‘views.py’ contains code for functionalities of the project.
7. The python file ‘urls.py’ contains code for urls required for the project.

CODE:

settings.py

```
"""

Django settings for news project.

"""

import os

# Build paths inside the project like this: os.path.join(BASE_DIR,
...)

BASE_DIR =
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

SECRET_KEY = '%p_4c&s7((gy&^n=q@t61x5&8w#t(5n_ponwbc0xf%tbj03ea@'

DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',
    'rest_framework',
    'agency.apps.AgencyConfig',
    'rest_framework.authtoken',
    'multiselectfield'
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'news.urls'

TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',

'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]


WSGI_APPLICATION = 'news.wsgi.application'
# Database
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
            'NAME': "newsAgency",
            'USER': "root",
            'PASSWORD': 'hacker',
            'HOST': 'localhost',
            'PORT': '3306',
    }
}


# Password validation
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_L10N = True
USE_TZ = False

STATIC_URL = '/static/'
```

urls.py - news

```
from django.contrib import admin
from django.urls import path

from django.conf.urls import include, url
from rest_framework.authtoken import views
from django.conf.urls.static import static
from django.conf import settings
from rest_framework.schemas import get_schema_view

urlpatterns = [
    url(r'agency/',include('agency.urls')),
    url(r'^admin/', admin.site.urls),
]
```

wsgi.py

```
import os
from django.core.wsgi import get_wsgi_application
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'news.settings')
application = get_wsgi_application()
```

apps.py

```
from django.apps import AppConfig
class AgencyConfig(AppConfig):
    name = 'agency'
```

models.py

```
from django.db import models
from django.db.models.signals import post_save
from django.dispatch import receiver
from rest_framework.authtoken.models import Token
from django.conf import settings
from django.contrib.auth.models import User
from multiselectfield import MultiSelectField
# from django.db import models
from django.contrib.auth.models import AbstractUser
# from datetime import datetime
from django.utils import timezone
from django.core.validators import RegexValidator

a_numeric = RegexValidator(r'^[a-zA-Z]*$', 'only alphabets are
allowed')
d_ten = RegexValidator(r'^[0-9]{10}$', 'enter 10 digits')
d_six = RegexValidator(r'^[0-9]{6}$', 'enter 6 digits')

class Distributors(models.Model):
    distributor_name = models.CharField(max_length=200)
    address = models.CharField(max_length=200)
    d_phone = models.CharField(max_length=200, validators=[d_ten])

class Publications(models.Model):
    paper_name = models.CharField(max_length=200,
validators=[a_numeric])
```

```
    language = models.CharField(max_length=200,
validators=[a_numeric])
    description = models.CharField(max_length=200)
    price = models.FloatField()

    def __str__(self):
        return self.paper_name

    class Meta:
        ordering = ('paper_name',)

class Customers(models.Model):
    customer_name = models.CharField(max_length=200,
validators=[a_numeric])
    address = models.CharField(max_length=200)
    pincode = models.CharField(max_length=6, default='',
validators=[d_six])
    phone = models.CharField(max_length=200, validators=[d_ten])
    subscription =
models.ManyToManyField('Publications',default='')
    due = models.FloatField(default=0)

class Subscript(models.Model):
    phone = models.CharField(max_length=200)
    subscription = models.ManyToManyField('Publications',
default='')

class WithHold(models.Model):
    customer_id = models.ForeignKey('Customers',
on_delete=models.CASCADE)
    from_date = models.DateField(default=timezone.now)
    to_date = models.DateField(default=timezone.now)

class Summary(models.Model):
    distributor_id = models.ForeignKey('Distributors',
on_delete=models.CASCADE)
    customer_id = models.ForeignKey('Customers',
on_delete=models.CASCADE)
    date = models.DateField(default=timezone.now)
```

serializers.py

```
from rest_framework import serializers, fields
from django import forms
from agency.models import Customers, Distributors, Publications,
WithHold, Summary, Subscript
from django.contrib.auth.models import User
from rest_framework import permissions
from django.contrib.auth import get_user_model


class CustomersSerializer(serializers.ModelSerializer):
    # subscription = fields.MultipleChoiceField(choices=CHOICES)


    class Meta:
        model = Customers
        fields = ('id', 'customer_name', 'address', 'pincode',
'phone', 'subscription')




class EditCustomersSerializer(serializers.ModelSerializer):
    # subscription = fields.MultipleChoiceField(choices=CHOICES)


    class Meta:
        model = Subscript
        fields = ('id','phone', 'subscription')


class EditPubicationSerializer(serializers.ModelSerializer):


    class Meta:
        model = Publications
        fields = ('id','paper_name', 'price')

class DistributorsSerializer(serializers.ModelSerializer):


    class Meta:
        model = Distributors
        fields = ('id','distributor_name', 'address', 'd_phone')



class PublicationsSerializer(serializers.ModelSerializer):


    class Meta:
        model = Publications
        fields = ('id','paper_name', 'language', 'description', 
'price')

class WithHoldSerializer(serializers.ModelSerializer):
    phone = serializers.IntegerField()
    class Meta:
        model = WithHold
        fields = ('id','phone', 'from_date', 'to_date')



class SummarySerializer(serializers.ModelSerializer):


    class Meta:
        model = Summary
        fields = ('id', 'date')

class AllocationSerializer(serializers.ModelSerializer):
    distributor_name = serializers.CharField(max_length=200)
    class Meta:
        model = Summary
        fields = ('id', 'distributor_name', 'date')

class DistributorPaymentSerializer(serializers.ModelSerializer):
    month = serializers.IntegerField(default=1)
    year = serializers.IntegerField(default=2018)

    class Meta:
        model = Distributors
        fields = ('id', 'distributor_name', 'month', 'year')

class CustomerBillsSerializer(serializers.ModelSerializer):
    month = serializers.IntegerField(default=1)
    year = serializers.IntegerField(default=2018)

    class Meta:
        model = Customers
        fields = ('id', 'month', 'year')

class PaymentReceiptSerializer(serializers.ModelSerializer):
    month = serializers.IntegerField(default=1)
    year = serializers.IntegerField(default=2018)
    amount = serializers.IntegerField(default=0)

    class Meta:
        model = Customers
        fields = ('id', 'phone', 'month', 'year','amount')
```

fuctions.py

```
import math
def partition_list(a, k):
    #check degenerate conditions
    if k <= 1: return [a]
    if k >= len(a): return [[x] for x in a]
    partition_between = []
    for i in range(k-1):
        partition_between.append((i+1)*len(a)/k)
    average_height = float(sum(a))/k
    best_score = None
    best_partitions = None
    count = 0
    no_improvements_count = 0
    while True:
        partitions = []
        index = 0
        for div in partition_between:
            partitions.append(a[index:int(div)])
            index = int(div)
        partitions.append(a[index:])
        worst_height_diff = 0
        worst_partition_index = -1
        for p in partitions:
            height_diff = average_height - sum(p)
            if abs(height_diff) > abs(worst_height_diff):
                worst_height_diff = height_diff
                worst_partition_index = partitions.index(p)
        if best_score is None or abs(worst_height_diff) <
best_score:
            best_score = abs(worst_height_diff)
            best_partitions = partitions
            no_improvements_count = 0
        else:
            no_improvements_count += 1
        if worst_height_diff == 0 or no_improvements_count > 5 or
count > 100:
            return best_partitions
        count += 1
        if worst_partition_index == 0:
            if worst_height_diff < 0: partition_between[0] -= 1
            else: partition_between[0] += 1
        elif worst_partition_index == len(partitions)-1:
            if worst_height_diff < 0: partition_between[-1] += 1
            else: partition_between[-1] -= 1
        else:
            left_bound = worst_partition_index - 1
            right_bound = worst_partition_index
            if worst_height_diff < 0:
                if sum(partitions[worst_partition_index-1]) >
sum(partitions[worst_partition_index+1]):
                    partition_between[right_bound] -= 1
                else:
                    partition_between[left_bound] += 1
            else:
                if sum(partitions[worst_partition_index-1]) >
sum(partitions[worst_partition_index+1]):
                    partition_between[left_bound] -= 1
                else:
                    partition_between[right_bound] += 1
```

views.py

```
from django.http import HttpResponse
from agency.models import Customers, Distributors, Publications,
WithHold, Summary, Subscript
from agency.serializers import CustomersSerializer,
DistributorsSerializer, PublicationsSerializer, WithHoldSerializer,
SummarySerializer
from agency.serializers import EditCustomersSerializer,
EditPubicationSerializer, AllocationSerializer,
DistributorPaymentSerializer
from agency.serializers import CustomerBillsSerializer,
PaymentReceiptSerializer
from rest_framework import generics
from django.contrib.auth.models import User
from rest_framework import permissions
from rest_framework.parsers import FormParser, MultiPartParser
from rest_framework import permissions
from django.contrib.auth import get_user_model
from django.contrib.auth.decorators import login_required
from django.views.static import serve
from django.conf import settings
from rest_framework.views import APIView
from rest_framework.authtoken.models import Token
from rest_framework.response import Response
from rest_framework.authentication import TokenAuthentication
from rest_framework.permissions import IsAuthenticated
from django.shortcuts import redirect
from django.template.loader import get_template


import numpy as np
import pickle
from django.db.models import Count
from agency.functions import partition_list
from datetime import datetime
```

```
import pandas as pd
import fpdf
import pdfkit, json
import calendar


def index(request):
    return HttpResponse("Welcome to Newspaper Automation Agency System")


class AddCustomer(generics.CreateAPIView):
    queryset = Customers.objects.all()
    serializer_class = CustomersSerializer


class EditCustomerSubscription(generics.CreateAPIView):
    serializer_class = EditCustomersSerializer

    def create(self, request, *args, **kargs):
        serializer = EditCustomersSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()

        serializer1 = EditCustomersSerializer(Subscript.objects.get(phone=request.data['phone']))
        h = serializer1.data['subscription']

        Subscript.objects.all().delete()

        t = Customers.objects.get(phone=request.data['phone'])
        t.subscription.set(h)
        t.save()

        return Response("update successful")
```

```
class AddPublication(generics.CreateAPIView):
    queryset = Publications.objects.all()
    serializer_class = PublicationsSerializer


class EditPublication(generics.CreateAPIView):
    serializer_class = EditPubicationSerializer


    def create(self, request, *args, **kargs):



Publications.objects.filter(paper_name=request.data['paper_name']).
update(price=request.data['price'])
        return Response("update successful")


class AddDistributor(generics.CreateAPIView):
    queryset = Distributors.objects.all()
    serializer_class = DistributorsSerializer


class WithHoldCustomer(generics.CreateAPIView):
    serializer_class = WithHoldSerializer


    def create(self, request, *args, **kargs):
        withH = WithHold(from_date=request.data['from_date'],
to_date=request.data['to_date'])
        withH.customer_id_id =
Customers.objects.get(phone=request.data['phone']).id
        withH.save()
        return Response("update")



class AllocateArea(generics.CreateAPIView):
    queryset = Summary.objects.all()
    serializer_class = SummarySerializer
    def create(self, request, *args, **kargs):
        p =
Customers.objects.values('pincode').order_by('pincode').annotate(co
unt=Count('pincode'))
        count_list = list()
        pincode_list = list()
        for i in p:
            count_list.append(int(i['count']))
            pincode_list.append(i['pincode'])
        d_count = Distributors.objects.all().count()
        sol = partition_list(count_list, d_count)

        distributor = Distributors.objects.all()
        index = 0
        pin_no = 0
        for d in distributor:
            for i in range(len(sol[index])):
                ctms =
Customers.objects.filter(pincode=pincode_list[pin_no])
                for c in ctms:
                    hold =
WithHold.objects.filter(customer_id_id=c.id)
                    if hold:
                        for date in hold:
                            t_date =
pd.to_datetime(request.data['date']).date()
                            s_date = date.from_date
                            e_date = date.to_date
                            if s_date <= t_date <= e_date:
                                None
                            else:
                                summ =
Summary(distributor_id_id=d.id, customer_id_id=c.id,
date=request.data['date'])
                                summ.save()
                    else:
                        summ = Summary(distributor_id_id=d.id,
customer_id_id=c.id, date=request.data['date'])
                        summ.save()
                pin_no = pin_no + 1
            index = index + 1


        return Response("areas allocated")


class DistributorAllocation(generics.CreateAPIView):
    queryset = Customers.objects.all()
    serializer_class = AllocationSerializer


    def post(self, request, *args, **kargs):
        d_id =
Distributors.objects.get(distributor_name=request.data['distributor
_name']).id
        ctms =
Summary.objects.filter(distributor_id_id=d_id).filter(date=request.
data['date'])
        c_list = list()
        for c in ctms:
            customer =
CustomersSerializer(Customers.objects.get(id=c.customer_id_id))
            cus = dict()
            cus['customer_name'] = customer.data['customer_name']
            cus['address'] = customer.data['address']
            cus['pincode'] = customer.data['pincode']
            cus['phone'] = customer.data['phone']
            cus['subscriptions'] = list()
            for i in range(len(customer.data['subscription'])):

cus['subscriptions'].append(Publications.objects.get(id=customer.da
ta['subscription'][i]).paper_name)
            c_list.append(cus)
        return Response(c_list)


class DistributorPayment(generics.CreateAPIView):
    queryset = Customers.objects.all()
    serializer_class = DistributorPaymentSerializer


    def post(self, request, *args, **kargs):
        s_date = pd.to_datetime(request.data['year'] + '-' +
request.data['month'] + '-1' ).date()
        e_date = pd.to_datetime(request.data['year'] + '-' +
request.data['month'] + '-30').date()
        i =
Distributors.objects.get(distributor_name=request.data['distributor
_name']).id
        s =
Summary.objects.filter(distributor_id_id=i).filter(date__range=(s_d
ate, e_date))
        payment = 0
        for c in s:
            customer =
CustomersSerializer(Customers.objects.get(id=c.customer_id_id))
            for i in range(len(customer.data['subscription'])):
                payment = payment +
Publications.objects.get(id=customer.data['subscription'][i]).price


        return Response(payment/4)


class CustomersBills(generics.CreateAPIView):
    queryset = Customers.objects.all()
    serializer_class = CustomerBillsSerializer


    def post(self, request, *args, **kargs):
        s_date = pd.to_datetime(request.data['year'] + '-' +
request.data['month'] + '-1' ).date()
        e_date = pd.to_datetime(request.data['year'] + '-' +
request.data['month'] + '-30').date()
```

```
        customers = Customers.objects.all()
        for ctmr in customers:
            bill = dict()
            sub =
Summary.objects.filter(customer_id_id=ctmr.id).filter(date__range=(
s_date, e_date)).count()
            bill['Name'] = ctmr.customer_name
            bill['Month'] = request.data['month'] + '-' +
request.data['year']
            bill['Bill'] = dict()
            c =
CustomersSerializer(Customers.objects.get(phone=ctmr.phone))
            for i in range(len(c.data['subscription'])):
                name =
Publications.objects.get(id=c.data['subscription'][i]).paper_name
                price =
Publications.objects.get(id=c.data['subscription'][i]).price
                bill['Bill'][name] = price


            amt = 0
            bills = list()
            st = 'Name    ' + bill['Name']
            bills.append(st)
            st = 'Month   ' + bill['Month']
            bills.append(st)
            bills.append("Bills")
            for key, value in bill['Bill'].items():
                st = '        ' + key + '   :   ' + str(sub) + ' x
' + str(value)
                bills.append(st)
                amt = amt + sub * value
            due = ctmr.due
            amt = amt + due
            st = 'Due     ' + str(due)
            bills.append(st)
            st = 'Total Amount : Rs.' + str(amt)
            bills.append(st)

            Customers.objects.filter(id=ctmr.id).update(due=amt)

            if amt:
                pdf = fpdf.FPDF(format='letter')
                pdf.add_page()
                pdf.image('logo.jpg',0,0,210,45)
                pdf.set_font("Arial", size=12)
                pdf.cell(100)
                pdf.ln(30)
                pdf.write(bill)
                for i in bills:
                    pdf.write(5,i)
                    pdf.ln()
                p = "bills/" + bill['Month'] + "/" + str(ctmr.id) +
'_' + ctmr.customer_name + '.pdf'
                pdf.output(p)
        return Response("Bills saved")


class ViewSummary(generics.CreateAPIView):
    queryset = Summary.objects.all()
    serializer_class = CustomerBillsSerializer


    def post(self, request, *args, **kargs):
        s_date = pd.to_datetime(request.data['year'] + '-' +
request.data['month'] + '-1' ).date()
        e_date = pd.to_datetime(request.data['year'] + '-' +
request.data['month'] + '-30').date()


        month = "Month    :   " + request.data['month'] + '-' +
request.data['year'] + "\n\n"
        tit = "Customer phone publications received\n"
        customers = Customers.objects.all()
        summ = list()
        paper = dict()
        publica = Publications.objects.all()
        for p in publica:
            paper[p.paper_name] = 0
        for ctmr in customers:
            sub =
Summary.objects.filter(customer_id_id=ctmr.id).filter(date__range=(
s_date, e_date)).count()
            subscri = ''
            c =
CustomersSerializer(Customers.objects.get(phone=ctmr.phone))
            for i in range(len(c.data['subscription'])):
                name =
Publications.objects.get(id=c.data['subscription'][i]).paper_name
                paper[name] = paper[name] + sub
                subscri = subscri + ', ' + name + '-' + str(sub)

            st = ctmr.customer_name + "          " + ctmr.phone + "
" + subscri
            summ.append(st)

        n = "\n\nDistribution of publications : "
        summ.append(n)
        tot = 0
        for key, value in paper.items():
            n = key + "   :   \t" + str(value)
            tot = tot + value
            summ.append(n)
        n = "Total publications delivered : \t" + str(tot)
        summ.append(n)

        pdf = fpdf.FPDF(format='letter')
        pdf.add_page()
        pdf.image('logo.jpg',0,0,210,45)
        pdf.set_font("Arial", size=12)
        pdf.cell(100)
        pdf.ln(30)

        pdf.write(5, month)
        pdf.write(5,tit)
        for i in summ:
            pdf.write(5,i)
            pdf.ln()

        month = request.data['month'] + '-' + request.data['year']
        p = "bills/" + month + "/summary.pdf"
        pdf.output(p)
        return Response("Summary pdf saved")

class PaymentReceipts(generics.CreateAPIView):
    queryset = Customers.objects.all()
    serializer_class = PaymentReceiptSerializer

    def post(self, request, *args, **kargs):

        tit = "Payment Receipt\n\n"
        month = "Month     " + request.data['month'] + '-' +
request.data['year'] + "\n\n"

        customers =
Customers.objects.get(phone=request.data['phone'])
        bills = list()
        st = 'Name     ' + customers.customer_name
        bills.append(tit)
        bills.append(st)
        bills.append(month)


        due = "Due amount     " + str(customers.due)
        bills.append(due)
        due = "Received amount     " + str(request.data['amount'])
        bills.append(due)
        due = "Remaining due     " + str(int(customers.due) - 
int(request.data['amount']))
        bills.append(due)



Customers.objects.filter(id=customers.id).update(due=int(customers.
due) - int(request.data['amount']))


        pdf = fpdf.FPDF(format='letter')
        pdf.add_page()
        pdf.image('logo.jpg',0,0,210,45)
        pdf.set_font("Arial", size=12)
        pdf.cell(100)
        pdf.ln(30)


        for i in bills:
            pdf.write(5,i)
            pdf.ln()


        m =  request.data['month'] + '-' + request.data['year']
        p = "receipts/" + m + "/" + str(customers.id) + '_' + 
customers.customer_name + '.pdf'
        pdf.output(p)
        return Response("receipt pdf saved")
```

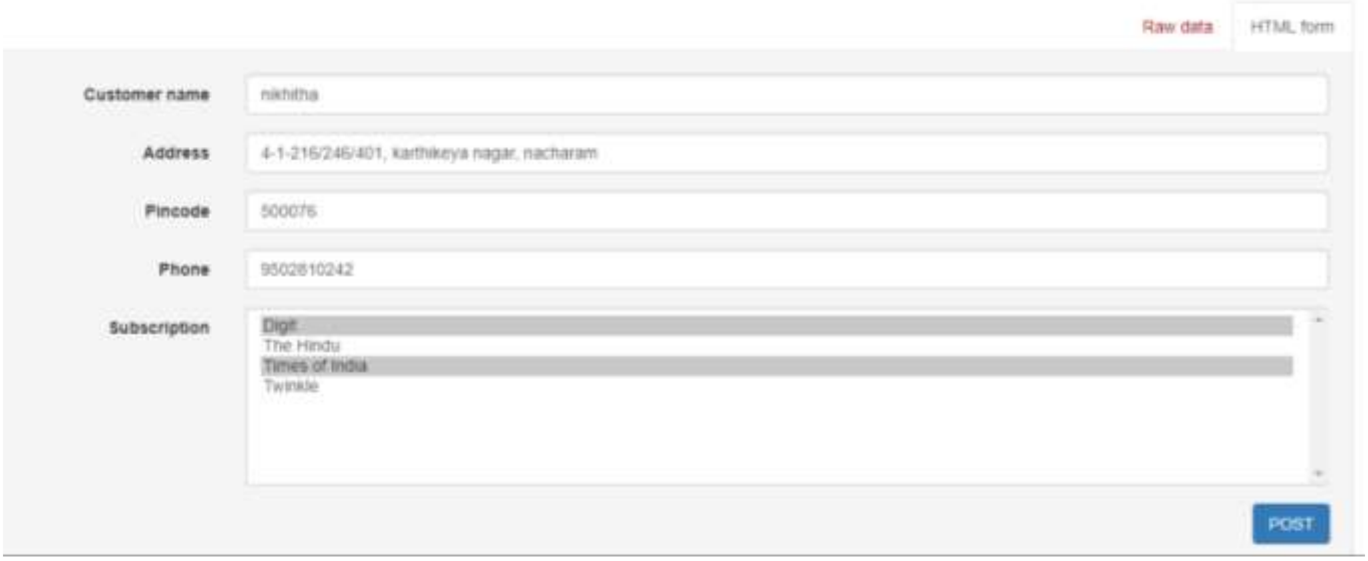
urls.py

```
from django.conf.urls import  url
from rest_framework.authtoken import views as tokenview
from rest_framework.urlpatterns import format_suffix_patterns
from agency import views
from django.conf.urls import include
from django.contrib.auth.decorators import login_required
from django.views.static import serve
from django.conf import settings
from django.conf.urls.static import static


urlpatterns = [
    url(r'^$', views.index, name='index'),
    url(r'^addCustomer/$', views.AddCustomer.as_view()),
    url(r'^editCustomer/$',
views.EditCustomerSubscription.as_view()),
    url(r'^addDistributor/$', views.AddDistributor.as_view()),
    url(r'^addPublication/$', views.AddPublication.as_view()),
    url(r'^editPublication/$', views.EditPublication.as_view()),
    url(r'^withHold/$', views.WithHoldCustomer.as_view()),
    url(r'^allocateArea/$', views.AllocateArea.as_view()),
    url(r'^distributorCheck/$',
views.DistributorAllocation.as_view()),
    url(r'^distributorPayment/$',
views.DistributorPayment.as_view()),
    url(r'^customerBills/$', views.CustomersBills.as_view()),
    url(r'^viewSummary/$', views.ViewSummary.as_view()),
    url(r'^paymentReceipts/$', views.PaymentReceipts.as_view()),


]


urlpatterns = format_suffix_patterns(urlpatterns)
urlpatterns += [
    url(r'^api-auth/', include('rest_framework.urls',
                               namespace='rest_framework')),
]
```

Output:

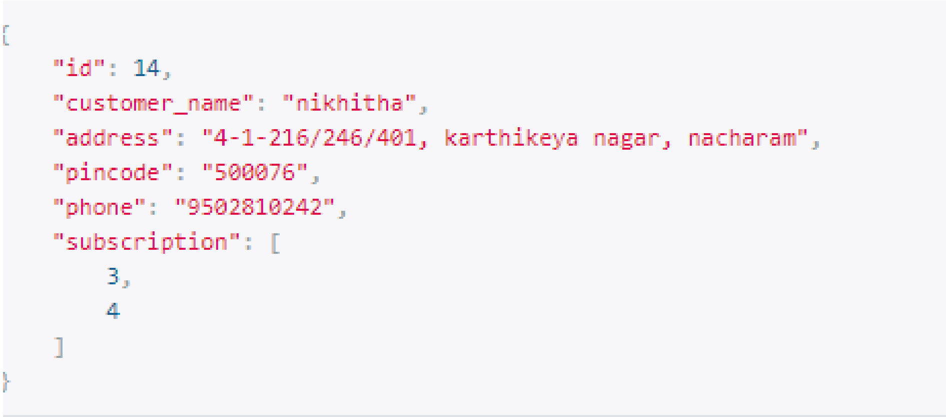
Add Customers

Input



Customer name	nikhitha
Address	4-1-216/246/401, karthikeya nagar, nacharam
Pincode	500076
Phone	9502810242
Subscription	Digit The Hindu Times of India Twinkle

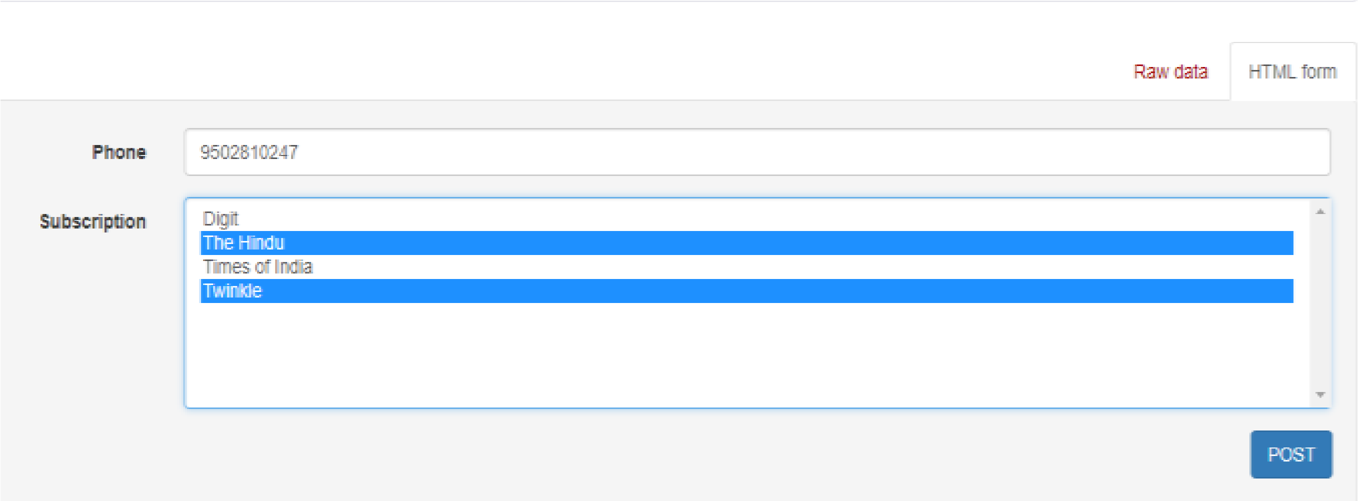
output



```
{
    "id": 14,
    "customer_name": "nikhitha",
    "address": "4-1-216/246/401, karthikeya nagar, nacharam",
    "pincode": "500076",
    "phone": "9502810242",
    "subscription": [
        3,
        4
    ]
}
```

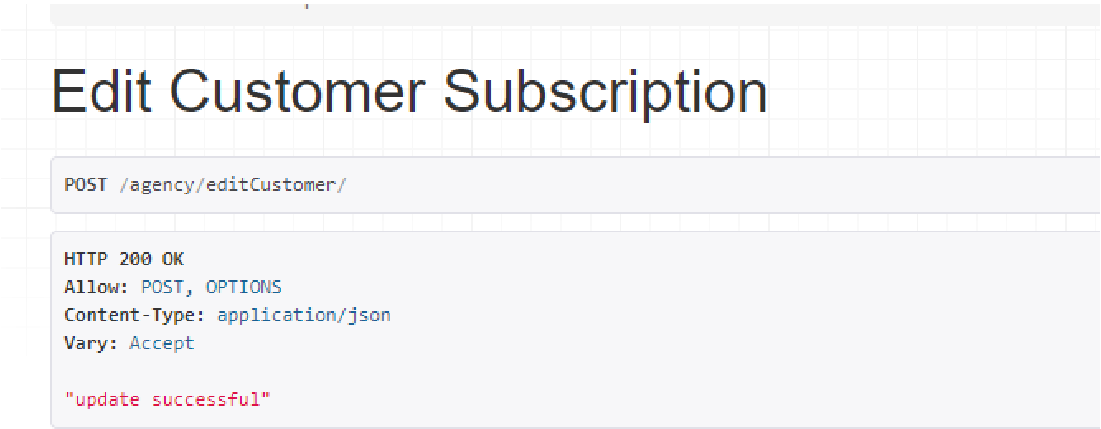
Edit customers

input



Phone	9502810247
Subscription	Digit [✓ The Hindu] Times of India [✓ Twinkle]

Output



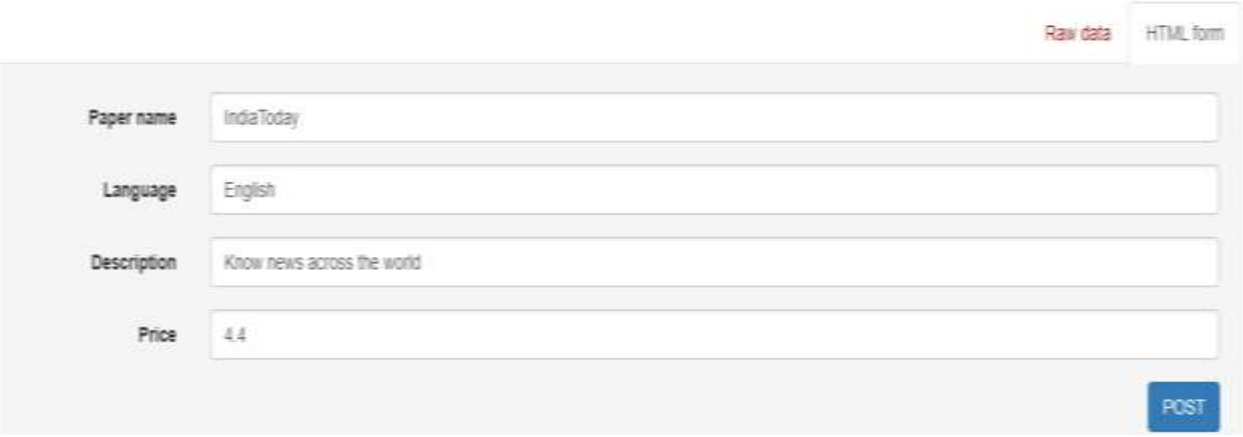
Edit Customer Subscription

```
POST /agency/editCustomer/
```

```
HTTP 200 OK
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

"update successful"
```

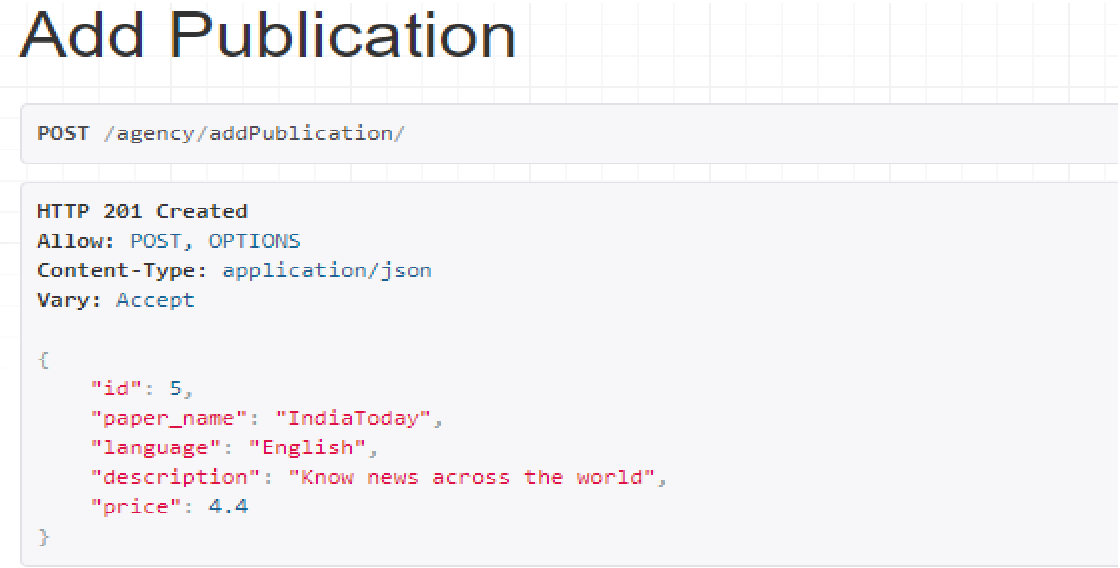
Add Publication

Input



Paper name	IndiaToday
Language	English
Description	Know news across the world
Price	4.4

Output



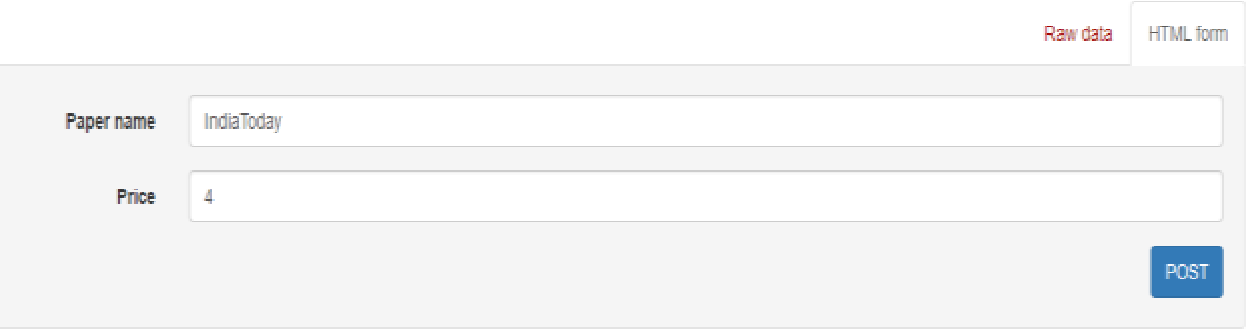
Add Publication

```
POST /agency/addPublication/
```

```
HTTP 201 Created
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "id": 5,
    "paper_name": "IndiaToday",
    "language": "English",
    "description": "Know news across the world",
    "price": 4.4
}
```

Edit Publication

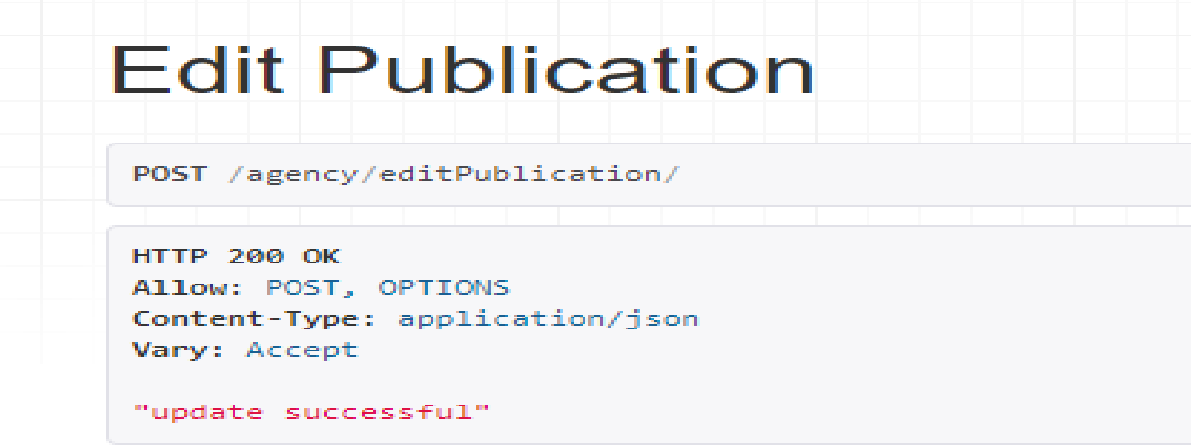
Input



Paper name: IndiaToday

Price: 4

Output

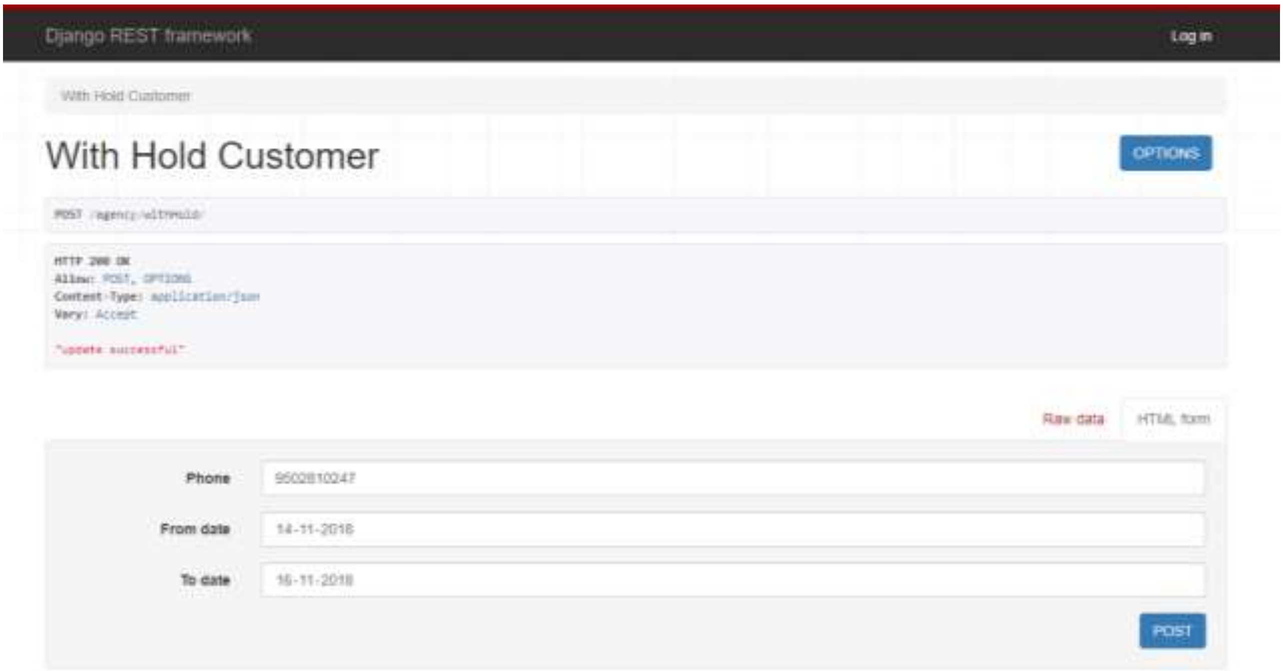


Edit Publication

```
POST /agency/editPublication/
```

```
HTTP 200 OK
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

"update successful"
```

WithHold Customers



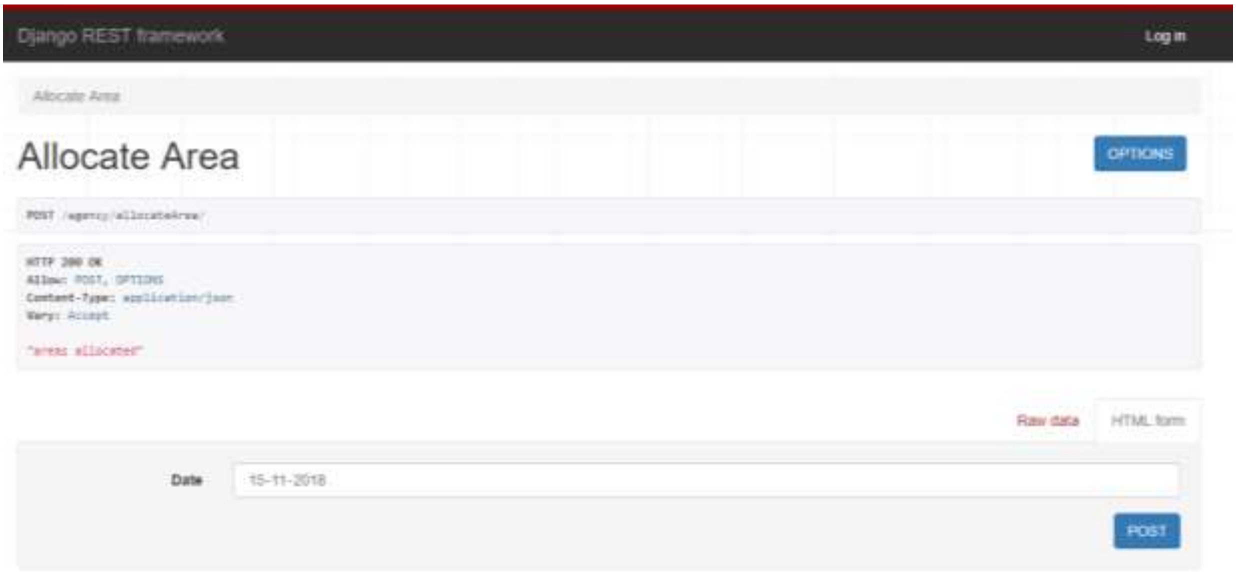
With Hold Customer

```
POST /agency/withHold/
```

```
HTTP 200 OK
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

"update successful"
```

Phone: 9502810247
From date: 14-11-2018
To date: 16-11-2018

Allocate Area



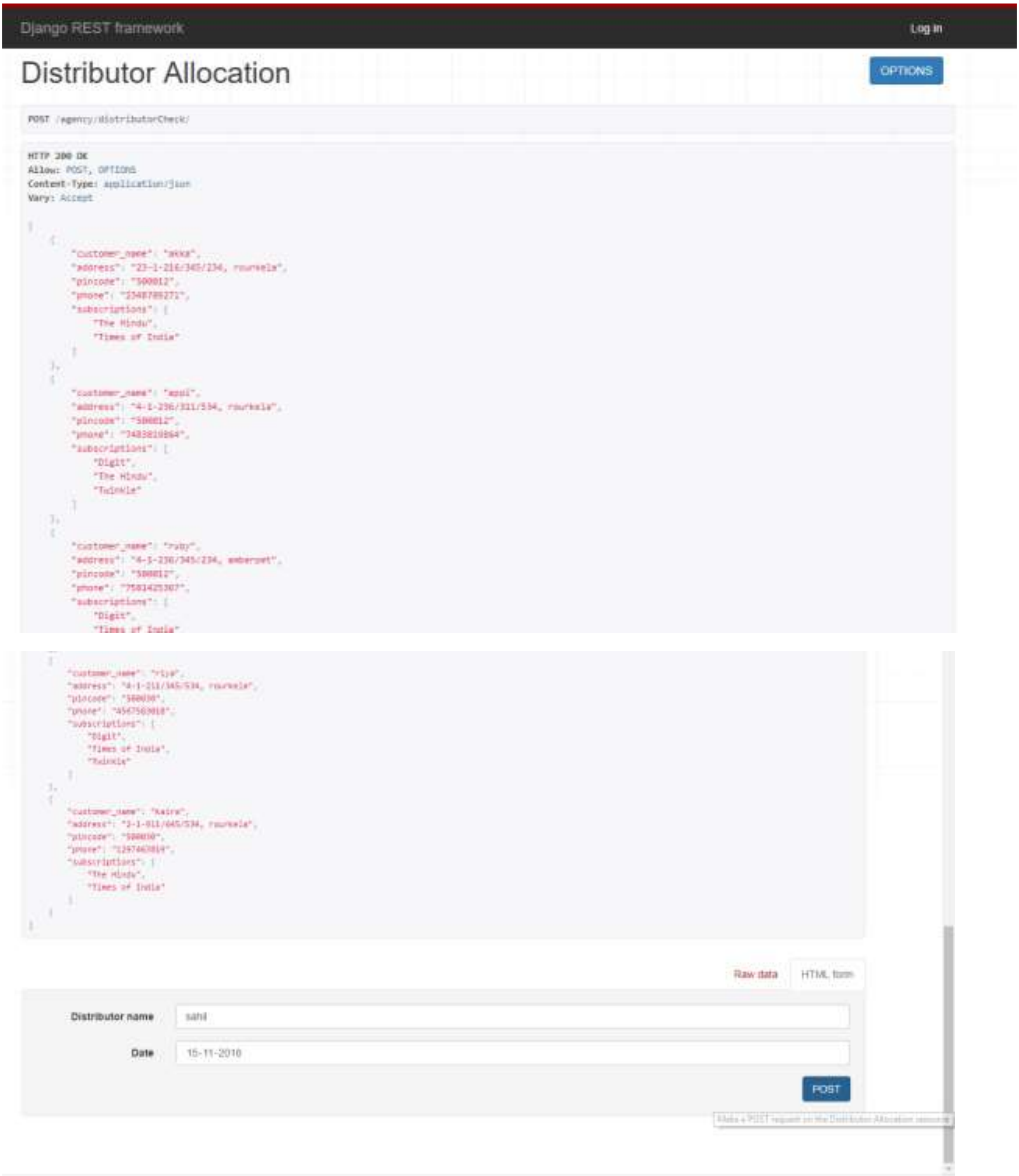
Allocate Area

```
POST /agency/allocateArea/
```

```
HTTP 200 OK
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

"areas allocated"
```

Date: 15-11-2018

Distributor Allocation



Distributor Allocation

```
POST /agency/distributorCheck/
```

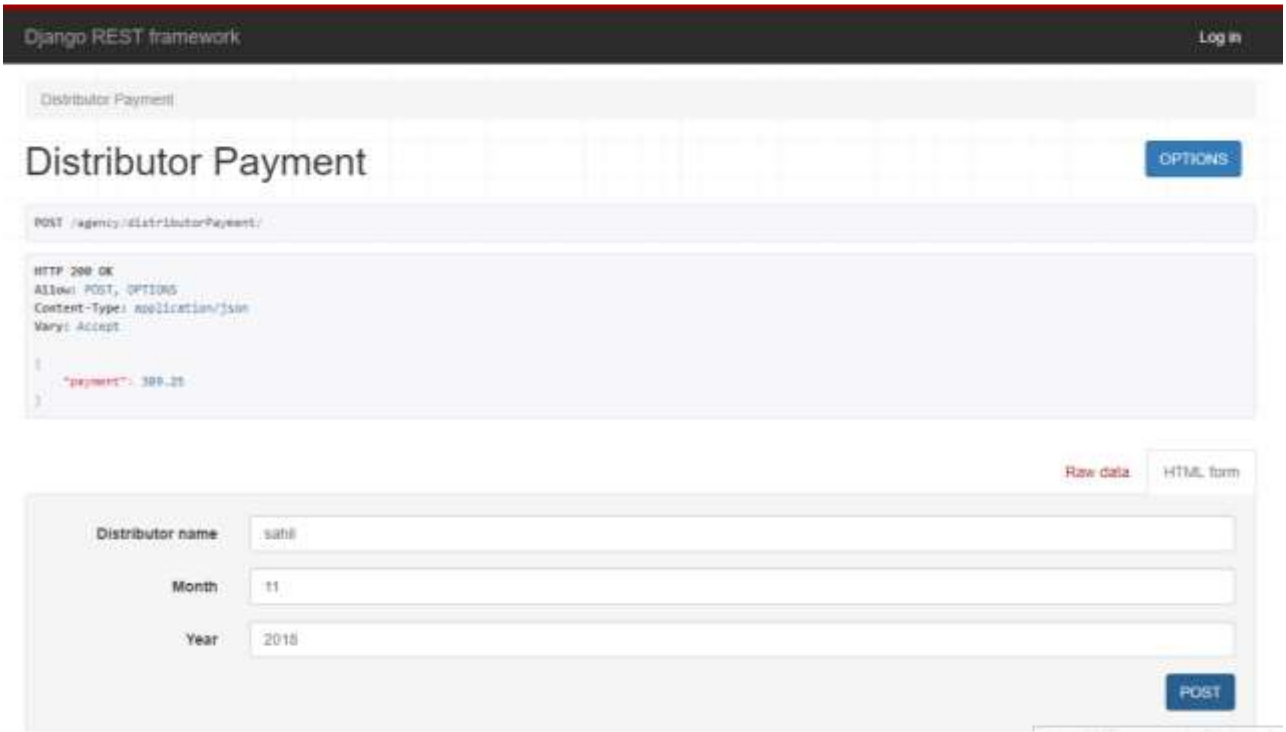
```
HTTP 200 OK
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "customer_name": "akka",
        "address": "23-1-216/345/234, rourkela",
        "pincode": "500012",
        "phone": "2348789271",
        "subscriptions": [
            "The Hindu",
            "Times of India"
        ]
    },
    {
        "customer_name": "apul",
        "address": "4-1-236/311/534, rourkela",
        "pincode": "500012",
        "phone": "7483819864",
        "subscriptions": [
            "Digit",
            "The Hindu",
            "Twinkle"
        ]
    },
    {
        "customer_name": "ruby",
        "address": "4-1-236/345/234, amberpet",
        "pincode": "500012",
        "phone": "7581425307",
        "subscriptions": [
            "Digit",
            "Times of India"
```

```
    {
        "customer_name": "riya",
        "address": "4-1-211/345/534, rourkela",
        "pincode": "500030",
        "phone": "4567569818",
        "subscriptions": [
            "Digit",
            "Times of India",
            "Twinkle"
        ]
    },
    {
        "customer_name": "kaira",
        "address": "3-1-811/645/534, rourkela",
        "pincode": "500030",
        "phone": "1297467819",
        "subscriptions": [
            "The Hindu",
            "Times of India"
        ]
    }
]
```

Distributor name: sahil

Date: 15-11-2018

Distributor payment



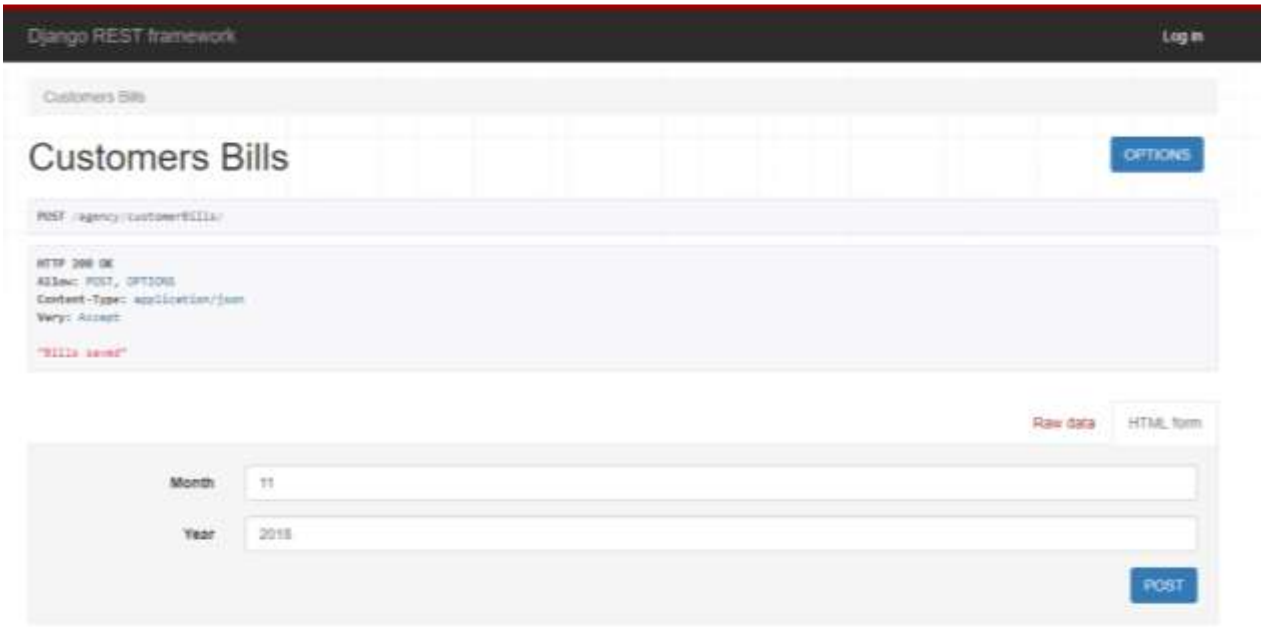
Distributor Payment

```
POST /agency/distributorPayment/
```

```
HTTP 200 OK
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "payment": 309.25
}
```

Distributor name	sahil
Month	11
Year	2018

Customer Bills

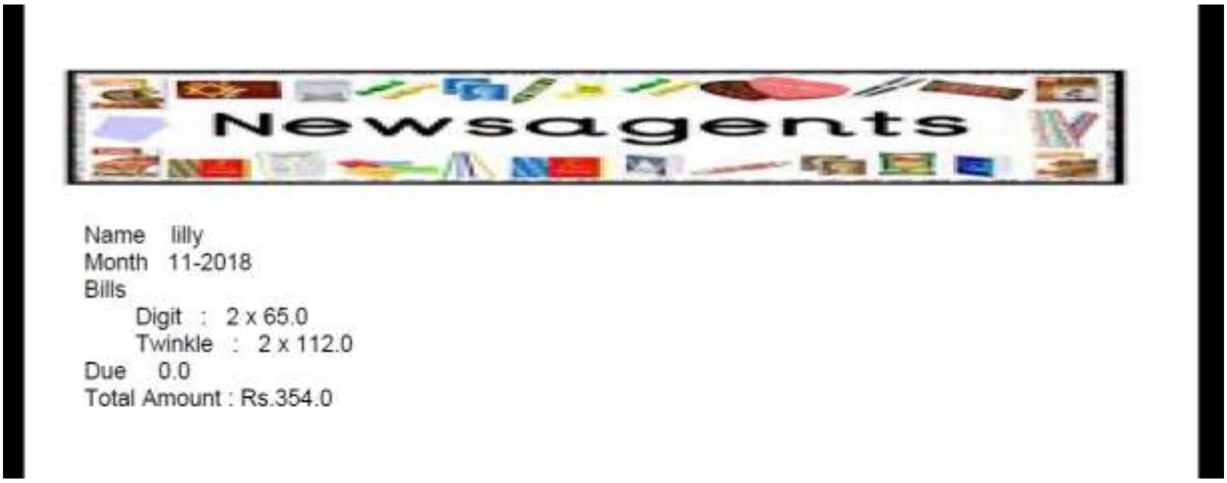


Customers Bills

```
POST /agency/customerBills/
```

```
HTTP 200 OK
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

"Bills saved"
```

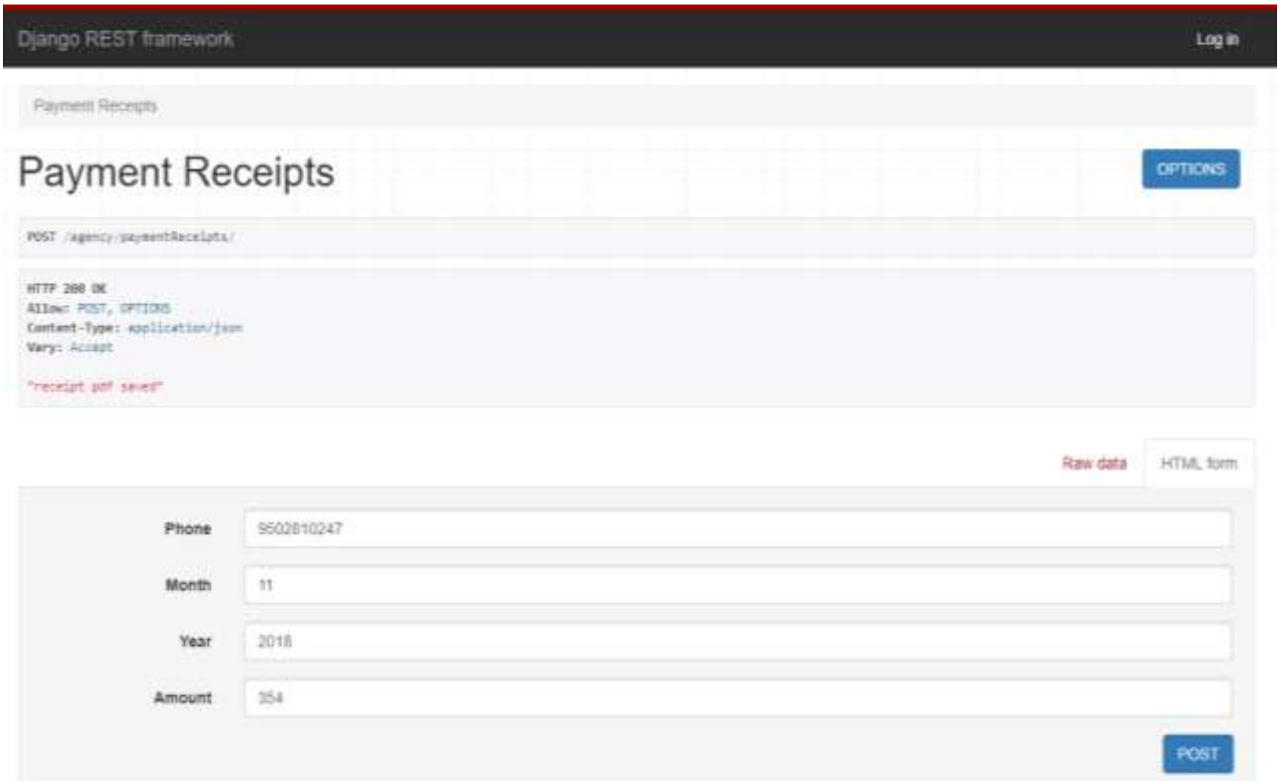
Month	11
Year	2018


Newsagents

Name lilly
Month 11-2018
Bills
Digit : 2 x 65.0
Twinkle : 2 x 112.0
Due 0.0
Total Amount : Rs.354.0

customer_bill.pdf

Customer Payment Receipts



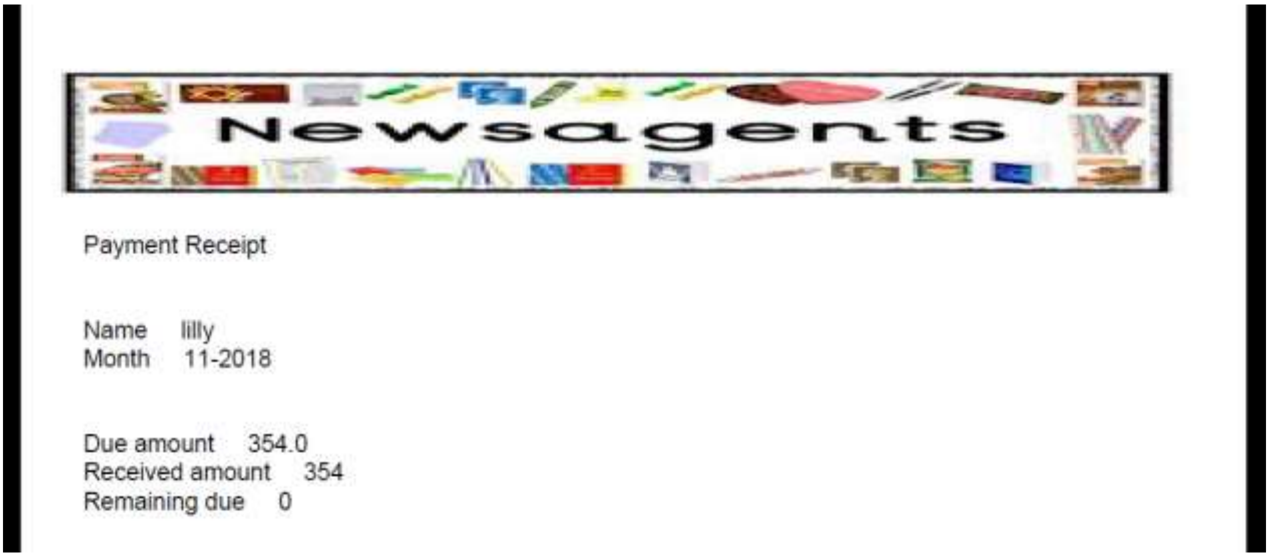
Payment Receipts

Payment Receipts

```
POST /agency/paymentReceipts/
```

```
HTTP 200 OK
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

"receipt pdf saved"
```

Phone	9502810247
Month	11
Year	2018
Amount	354



Newsagents

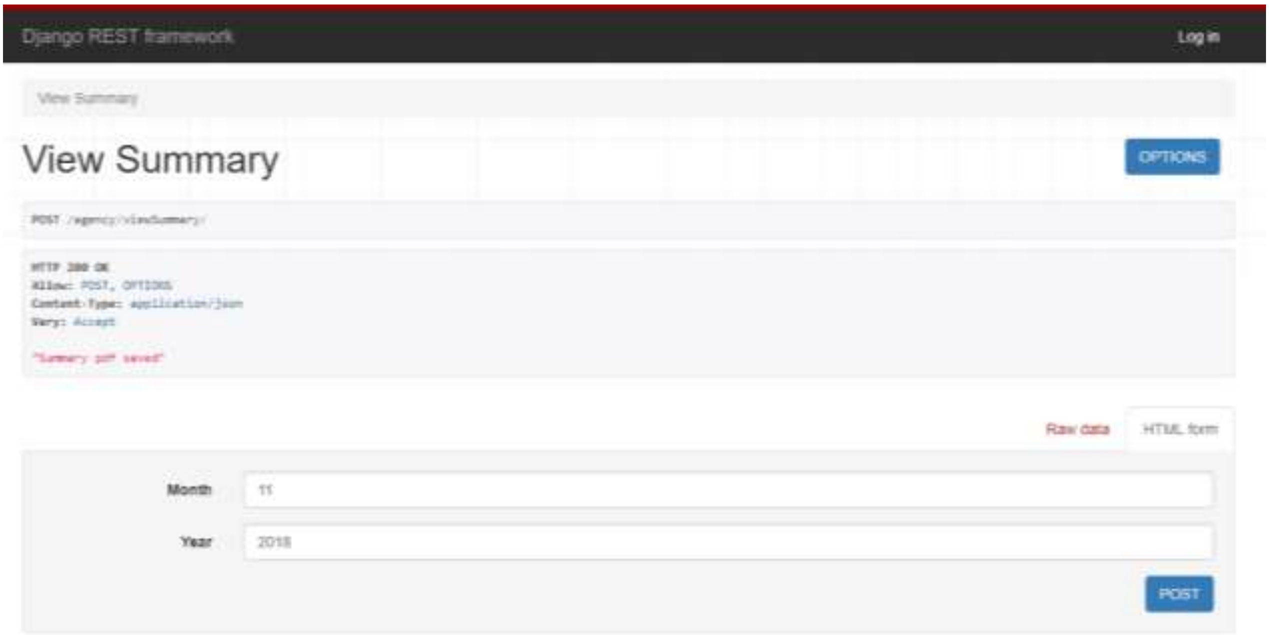
Payment Receipt

Name lilly
Month 11-2018

Due amount 354.0
Received amount 354
Remaining due 0

Receipt.pdf

View Summary



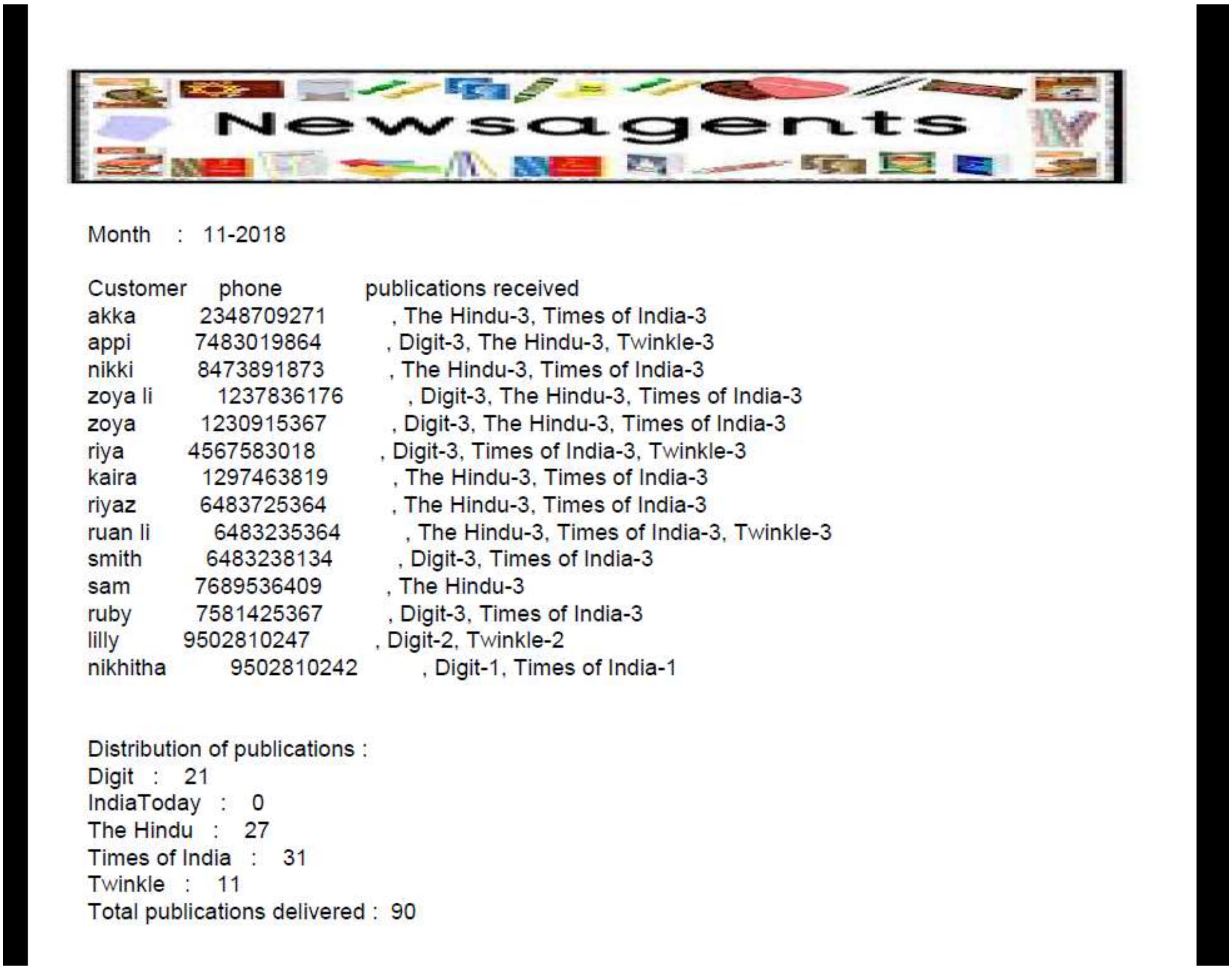
View Summary

```
POST /agency/viewSummary/
```

```
HTTP 200 OK
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

"Summary pdf saved"
```

Month 11

Year 2018



```
Month   :  11-2018

Customer    phone          publications received
akka        2348709271        , The Hindu-3, Times of India-3
appi        7483019864        , Digit-3, The Hindu-3, Twinkle-3
nikki       8473891873        , The Hindu-3, Times of India-3
zoya li        1237836176        , Digit-3, The Hindu-3, Times of India-3
zoya        1230915367        , Digit-3, The Hindu-3, Times of India-3
riya        4567583018        , Digit-3, Times of India-3, Twinkle-3
kaira       1297463819        , The Hindu-3, Times of India-3
riyaz       6483725364        , The Hindu-3, Times of India-3
ruan li        6483235364        , The Hindu-3, Times of India-3, Twinkle-3
smith       6483238134        , Digit-3, Times of India-3
sam         7689536409        , The Hindu-3
ruby        7581425367        , Digit-3, Times of India-3
lilly       9502810247        , Digit-2, Twinkle-2
nikhitha        9502810242        , Digit-1, Times of India-1


Distribution of publications :
Digit  :   21
IndiaToday  :   0
The Hindu  :   27
Times of India  :   31
Twinkle  :   11
Total publications delivered :  90
```

Summary.pdf

---------------- x x x ----------------