# POS Rental Agreement System Architecture Specification

This document describes the architecture that is to be implemented to satisfy the requirements in the POS rental system functional specification. This system is a back-end system that performs 3 basic functions: (1) providing up-to-date options for the tools to be rented, (2) validating entries from the front-end POS system, and (3) generating a rental agreement based on input from the front-end POS system.

The system to be implemented is represented in the diagram below.

***Fig 1.*** *POS System Data Flow*

*A diagram of a diagram

Description automatically generated with medium confidence*

The class diagram below shows the skeleton framework for the system:

***Fig 2.*** *Class Diagram*

A screenshot of a computer

Description automatically generated

## The Checkout object

The main class that handles logic for the back end system is the Checkout class. This class exposes the methods getAvailableTools() and generateAgreement(), which are described below and which provide the business logic for getting options to display to the user and creating a final tool rental agreement.

***Fig 3.*** *Checkout class*

Public class Checkout {

public ToolRet getAvailableTools ();

public RentalAgreement generateAgreement(RentalParams params);

}

## Get Available Tools for the front-end POS System

In order to avoid any unnecessary direct ties between the front-end system and the database there will be a simple back-end method to return the current list of tools available for rental. The tools will be returned as a collection of objects where each tool object contains information as indicated in the following class definitions:

***Fig 4.*** *ToolCode class*

public class ToolCode

{

String ToolCode; // Unique code identifying the specific tool

String ToolType; // Type of tool, used to look up charge

// specifics

String Brand; // Tool brand name

}

*Note above that the “ToolType” field is listed as a simple string. It is expected that the available types of tools will be stored in a table with additional metadata about each tool type for maintenance and flexibility. However, for the purposes of this simple API the “Name” for the tool type is the only field used here to keep the logic simple for the front-end system.*

The returned collection is wrapped in a simple return object that also indicates a status code and error or informational message if needed.

***Fig 5.*** *ToolRet and BaseRet classes*

public class ToolRet extends BaseRet

{

ArrayList<ToolCode> Tools;

}

public class BaseRet

{

boolean Success; // True if the given method succeeded with

// no issues

String Message; // Message to display to the user to help

// resolve any issues that occurred}

The following pseudo-code shows how this method might be implemented:

public ToolRet getAvailableTools ()

{

// Perform any necessary validation / authentication checks

// Retrieve all data from the tool code table (tool code,

// tool type, brand)

// Create a ToolCode object for each and add it to our

// collection to return

// Return the full collection or set an error status and

// return message if something went wrong

}

## Validate Rental Agreement Input Parameters

This routine will be called in-line by the method that generates the rental agreement. It is used to verify that the input parameters are in valid ranges so the user can easily correct any issues and retry.

private BaseRet validateInput (RentalParams paramsIn)

{

// Set an error condition if # of rental days in the input

// is < 1

// Set an error condition if the discount % is < 0 or > 100

}

## Generate Rental Agreement

This method performs the main work of the module. It takes the input from the POS front-end, validates it, and generates a rental agreement using the rules from the POS system functional specification.

***Fig 6.*** *RentalAgreement class*

public class RentalAgreement

{

BaseRet Status; // Success/ Failure of any operation that

// returns this class

ToolCode CodeInfo; // Full tool code info for the specified

// tool to rent

ToolType ToolIn; // Full tool type info (e.g. charge rules)

// for the type of tool to rent

RentalParams ParamsIn; // Parameters entered at the POS system

// for this rental request

LocalDate DueDate; // Date the tool will be due based on

// start date and # rental days

int ChargeDays; // Total # of days we'll charge for after

// considering all factors

double PreDiscCharge; // Rental amount prior to applying

// discount

double DiscAmt; // Discount $$

double FinalCharge; // Final charge after discount

public void performCalcs () // Calculate final charge

// information

public void printToConsole() // Diagnostic print of all rental

// agreement values

}

public RentalAgreement generateAgreement (RentalParams paramsIn)

{

// Call the Validate method on input params and return an

// error if needed

// Create a rental agreement object and load it with

// the input parameters, the full ToolCode information

// from the dataset for the given tool, and the ToolType

// (charge) information for the given tool

// Call the performCalcs() method on the rental agreement

// and return the resulting object

}

### RentalAgreement – performCalcs()

This is the skeleton for the method that performs the rental agreement calculations:

public void performCalcs ()

{

// Calculate the due date using the checkout date from the

// input and the # of days requested

// Calculate the # of days to charge for the rental by walking

// forward from the start date to the end date and only

// adding days that match the type of days to charge for the

// given tool type (weekday, weekend, holiday)

// Calculate pre-discount charge (remaining days \* rental

// charge from the table) – round half-up to cents

// Calculate discount amount (entered discount % \*

// pre-discount charge) – round half-up to cents

// Calculate final charge (pre-discount – discount)

}

### RentalAgreement – printToConsole()

For diagnostics, the rental agreement object will implement a method to print all current properties to the console. Formatting of types is as follows:

*Date – mm/dd/yy*

*Currency - $#,###.## (i.e. US formatting)*

*Percent - ##%*