```
Api/management.py

def getAllSales():
    """
    Returns all sales in JSON format ready for table view
    :return: 200 if completed successfully
    """
def getXReport():
    """
    Retrieves the X report given the date range
    :return: 200 on completion
    """
def getZReport():
    """
    Retrieves the Z report based on reported items
    :return: 200 on success or 204 if there is nothing to report
    """
def getAndUpdateMenu():
    """
    Accepts various types of calls depending on requested action:
    GET returns the current state of the menu in table form
    POST updates a menu item's features such as ingredients and price
    DELETE removes a menu item
    PUT adds a menu item
    :return: 200 on success 400 on failure
    """
def getAndUpdateInventory():
    """
    Accepts various types of calls depending on requested action:
    GET returns the current state of the inventory in table form
    POST updates an ingredient's cost or low stock threshold
    DELETE removes an ingredient, requests the identifier and two booleans
regarding safe deletion of items
    PUT adds a menu item based on data
    :return:
    """
def restockInventory():
    """
    Attempts to restock the given inventory items listed, should be a series
of items in the form
    index: name and amount
    :return: 204 on success and 400 on fail
    """
def voidInventoryItem():
    """
    Attempts to void the given inventory item listed by the given amount
    {inventory_name, amount}
    :return: 204 on success and 400 on fail
    """
def viewTransactions():
    """
    Returns JSON table-ready form of the requested transactions. GET returns
the latest 500 and
    POST returns all within a date range
    :return: table-ready JSON containing each transaction received
    """
```

```
def getAndUpdateEmployees():
    """
    Modifies the employee list or retrieves it according to the request
method:
    GET retrieves all entries
    POST adds a new employee
    DELETE removes an employee
    :return: 200 on success, 400 on failure
    """

def getItem():
    """
    Gets the requested item from the database
    :return: The JSON of the item
    """
def getInvItem():
    """
    Gets the requested item from the database
    :return: The JSON of the item
    """


def getAllSales():
    """
    Returns all sales entries in the database

    :return: All sales collected by the database
    :rtype: list[Tuple[dateTime, boolean, integer, float, boolean]]
    """
```

```
models/management.py

def getAllSales():
    """
    Returns all sales entries in the database

    :return: All sales collected by the database
    :rtype: list[Tuple[dateTime, boolean, integer, float, boolean]]
    """

def matchItemToCategoryAndPrice() -> dict:
    """
    Returns a dictionary matching every item to a category and price
respectively in a tuple
    :return: A dictionary pairing to respective category and price for each
item
    :rtype: dict[str, Tuple[str, float]]
    """

def categorizeSales(startDate, endDate):
    """
    Browses the transactions during the date range, and condenses the
information organzied by category of each product
    Specifically meant for X and Z report, but could be used elsewhere

    :param startDate: A string representing the starting date to be
categorized or preset datetime
    :type startDate: str or datetime.datetime
    :param endDate: A string representing the ending date to be categorized
or preset datetime
    :type endDate: str or datetime.datetime
    :return: A dictionary matching each category to the money spent on items
of the category between the two dates
    :rtype: dict[str, float]
    """
def markDaysAsReported():
    """
    Helper function that marks days as reported so they won't be counted in
the next Z report
    """

def getZDateRange() -> tuple[datetime.datetime, datetime.datetime]:
    """
    Returns the date range needed to perform the Z report
    :return: A tuple of start date and ending date
    :rtype: Tuple[datetime, datetime]
    """

def viewTransactions(startDate=None, endDate=None):
    """
    Returns the list of transactions capping at either 500 or all entries if
start and end dates are provided
    :param startDate: starting date
    :type startDate: datetime.datetime or str
    :param endDate:  ending date
    :type endDate: datetime.datetime or str
```

```
    :return: The results from the performed query based on the date range
    :rtype: List[Tuple]
    """

def adjustPrice(identifier, price: float):
    """
    Adjusts the price of a given item

    :param identifier: Flexible identifier token can either be the item id or
name
    :type identifier: str or int
    :param price: The updated price
    :type price: float
    """


def getMenuItems():
    """
    Creates a JSON friendly dictionary from the database's menu
    :return: Dictionary modeling JSON return
    """
def getInventory():
    """
    Returns JSON formatted list of inventory items
    :return: JSON formatted list of inventory items
    :rtype: list[dict, ...]
    """
def addMenuItem(name: str, display: str, category: str, sized: bool,
ingredients: list[dict],
                price, autocalc=False):
    """
    Adds a menu item including the sized variants if applicable with the
given ingredients and price
    :param name: internal name of the item not including the size aka coffee-
tall -> coffee, size will be added in function
    :param display: external name of the item
    :param category: category of the drink. includes: espresso-drinks, tea-
hot-iced, bakery-coremark, coffee_hot_iced, add-on, frappuccino-and-blended
    :param sized: boolean representing if the drink has tall/grande/venti
variants or not
    :param ingredients: list containing pairs of ingredients and their
respective portions
    :param price: price of the item or tuple for the three different prices
    :param autocalc: flag that determines if pricing is auto-calculated for
sized drink at ratio of .85 and 1.25 for tall and venti respectively
    """


def removeMenuItem(identifier):
    """
    Remove a menu item from the menu
    :param identifier: a str for the item name or int for the item id
    :type identifier: int or str
    """
```

```
def updateMenuIngredients(id, newIngredients):
    """
    Updates the ingredients of the item with the given id to the updated list
    :param id: id of the menu item
    :param newIngredients: new ingredient list or lists if sized (3)
    :param sized: bool of if the item is sized
    """

def getItem(identifier):
    """
    Gets the price and ingredient of the given item
    :param identifier: a string or int representing the item_name or item_id
respectively
    :type identifier: int or str
    :return: Returns the price and ingredients of the given item
    """

def getIngredients(identifier):
    """
    Returns the ingredients and amounts from the item in the parameters
    :param identifier: a string or int representing the item_name or item_id
respectively
    :type identifier: int or str
    :return: returns the list of pairs of ingredients and their amounts
    """


def restockItem(identifier, amount):
    """
    Restocks the item given by identifier with the given amount of the item
in the database
    :param identifier: a string or int representing the item_name or item_id
respectively
    :type identifier: int or str
    :param amount: amount of item to be restocked
    """
def addInventoryItem(name: str, initialAmount: int, cost: float,
lowStockThreshold: float = 25):
    """
    Add a new inventory item. Checks for existing items and proper amounts
    :param name: name of ingredient
    :param initialAmount: initial amount of the item
    :param cost: cost to supply the item
    :param lowStockThreshold: threshold to be considered low stock
    """
def changeInventoryCost(identifier, cost: float):
    """
    Changes the inventory ingredient cost in the database
    :param identifier: a string or int representing the inventory_name or
inventory_id respectively
    :type identifier: int or str
    :param cost: new cost of the ingredient
    :type cost: float
    """

def changeLowStockThreshold(identifier, threshold: float):
    """
```

```
    Changes the low stock threshold of the given item
    :param identifier: a string or int representing the inventory_name or
inventory_id respectively
    :type identifier: int or str
    :param threshold: new threshold of low stock
    :type threshold: float or int
    """


def getLowStock():
    """
    Finds all items that are considered low stock (stock < minimum quantity)
    :return: Table of all low stock items and all columns from the database
    """


def removeIngredient(identifier, deleteIfStockLeft: bool = False,
deleteIfInMenu: bool = False):
    """
    Safely deletes an ingredient from the database. If deleteIfInMenu is
enabled, menu items that rely on the ingredient will also be removed. Use
with caution
    :param identifier: a string or int representing the inventory_name or
inventory_id respectively
    :type identifier: int or str
    :param deleteIfStockLeft: flag that will allow deletion if stock is still
in the system
    :type deleteIfStockLeft: bool
    :param deleteIfInMenu: flag that will allow deletion if menu items are
still in the menu, will also delete those menu items
    :type deleteIfInMenu: bool
    :return: a bool indicating if the item was removed and a string message
corresponding to the reason
    :rtype: tuple[bool, str]
    """

def voidItem(identifier, amount):
    """
    Void an amount of  the item given by identifier in the database
    :param identifier: a string or int representing the item_name or item_id
respectively
    :type identifier: int or str
    :param amount: amount of item to be restocked
    :type amount: float
    """



def getAmountOfInventroy(identifier):
    """
    Get the quantity of item in the database
    :param identifier: a string or int representing the item_name or item_id
respectively
    :type identifier: int or str
    """
```

```
def getEmployees():
    """
    Retrieves all the employees to be listed in the table
    :return: JSON formatted list of employees
    """

def addEmployee(name, email, management=False):
    """
    Adds employee into the database
    :param name: Name of the employee
    :type name: str
    :param management: boolean representing if the employee is management
    :type management: bool
    :param email: email of the employee to be used with OAuth
    :type email: str
    :return: boolean representing if successfully added and message
    :rtype tuple[bool,str]
    """

def removeEmployee(identifier):
    """
    Using the employee email or id, which is unique to each employee, removes
the given employee
    :param identifier: Either the email or id to find the employee by
    :type identifier: int or str
    """

def getInvItem(identifier):
    """
    Delivers the JSON of the requested inventory item
    :param identifier: name of the inventory item
    :type identifier: str
    :return: JSON formatted dict of the inventory item
    """
```

```
menu_board_view.js

function autoscroll()

/**
 * Automatically scrolls up and down the page,
 * with a 5-second pause upon reaching the top or bottom,
 * before the scroll reverses.
 */


function scrollUp()
/**
 * Scrolls up by the number of pixels specified by top
 */
function scrollDown()
/**
 * Scrolls down by the number of pixels specified by top
 */
```