1. Task Entity
   1. Entity annotation: creates the table for task and the relations of Task
   2. @OneToOne annotation: 1 Task to 1 other entity relation
      1. @JoinColumns annotation: creates a column in Task table
   3. @ManyToOne annotation: many tasks to 1 assignment relation
      1. A many to one relation must be used with a one to many relation and the entity with the many to one relation is the owner of the relation where the primary and foreign exists in the Task entity
2. Canvasser Repository
   1. Our queries to the database regarding Canvasser functionality
   2. These queries are only our load/get, remove, and delete queries.
      1. Remove is for removing a relation from a entity
      2. Delete is for deleting a entity
      3. Did not use save as the query for that is 1 line
   3. Some of the queries are for the same entity but retrieve different relations
      1. As we wanted to reduce latency instead of retrieving all relations every single time
3. Create new assignment
   1. Start with screenshot of CM homepage
   2. estimateTasks
      1. for each location, calculate the distance from every other location and get the average, singleDistanceAvg
      2. get the average of the singleDistanceAverages
      3. the distances are calculated by the Manhattan distance using the Haversine formula
   3. set up json object for OR-Tools
   4. launchORT
      1. writes json object to a json file
      2. creates a promise object that calls back and spawns a child process to execute our OR-Tools program written in python
   5. OR-Tools
      1. Fills data
         1. Associates demand with estimated visit time
         2. Associates capacity with workday limit
      2. Create routing model with # locations, # canvassers, depot
         1. The depot is required for the constraint solver of OR-Tools
         2. Depot is the start and end node of each route
         3. This was an issue for our case so we added a duplicate location within the campaign and set it as the depot
            1. Which increases search accuracy rather than a random location
            2. Prevent search from calculating distance n travel time
      3. Create distance callback and set it as the function for SetArcCostEvaluatorOfAllVehicles which calculated the distance between each node
         1. In the callback, calculate the distance by using Manhattan distance
      4. Add constraint with demand callback
         1. Add the callback to add\_capacity\_constraints
         2. The callback calculates the total time consumed for traveling and servicing
         3. Travel time = Manhattan distance / travel speed
         4. Service time = global variable
         5. Remove the travel times for the deport, both start and end
      5. Set search option to first solution heuristic search
      6. For every route add to json object
      7. Dump json object into result json file
4. Assign Tasks
   1. Pass results from OR-Tools into createTasks
      1. Creates the task for the routes and assigns them to assignment and then to campaign
   2. assignTasks
      1. the canvasser’s available dates are not the current available dates. They are the initial date the canvasser chose for availability
         1. it must be cross referenced by the canvasser’s assigned dates
      2. for each canvasser, sort availableDates
         1. cross reference assigned dates and available dates
            1. if there are available dates not in assigned dates, those dates are the current available dates
      3. check if there are no available dates
         1. if then return with status 3 and null canvasser
      4. else assign the earliest date to tasks
      5. if signal == 2 then there are not enough available dates to assign a task
         1. return with status 2
      6. if earliest date is found, assign canvasser to task and update the current available dates and assigned dates