Final Project – Hardy – CIDM-6330

Contents

Problem Statement

As a software developer, I notice when systems aren't ideally optimized. One example that I ran into is the Boys and Girls Club, which my daughter is engaged with. I have no easy way to know what kind of activities are available to her, what I have paid for, who are emergency contacts, etc. As a parent living in the 21st century, these types of shortcomings catch my attention since I design systems to address problems like these for a living.

Not knowing what types of activities are available could lead to missed opportunities for children to enrich their lives with activities that will help them learn new skills. Additionally, such a system would ensure that payments are made more convenient for staff and guardians, and having accurate records of what's owed and paid is of great value.

By providing a self-service option, staff will have more time to engage with students versus accepting payments or telling guardians about available opportunities for their children. A system like this will help reduce frustration for guardians and staff and provide a better experience for the children!

Goal

To provide a system that would allow users to access information about the people associated with the Boys and Girls Club. The types of activities that are available, who have been enrolled in them, and payments that have been applied for those activities. The system would ideally have a web interface that can be accessed via mobile for ease of use. Guardians would be enabled to self-service

Solution

The system I had in mind would essentially be a mini CRM (Customer Relationship Management) program that would be targeted toward facilitating activity scheduling, portal access, and employee management. This is ideally suited for a software solution because it would enable both employees and guardians to have access to the same base information about the types of activities that are available. It would also provide access for as long as the system is online, which enables busy parents to find out more information and sign their child up for activities when it's convenient for them versus when making a frantic pickup or drop-off. Both a regular web and mobile interface could be utilized for ease of access to the information available. The information could also be exported or imported (e.g., payments) so the information is represented accurately across systems.

I thought through the issue in some depth for assignment 3, and the actual domain model has not changed much since that original submission. I will include the model in the appendix.

In the original design I put together, I put together an end-to-end system that would allow for:

- User Registration
- User Authentication
- People management:
    - Student
    - Guardians (for students)
    - Employee
- Government Id's associated with people

- Addresses associated with people
- Contact information associated with people
- Payments made
- Student Enrollments
- Activities (e.g., Soccer, After School care)
- Enrollment Periods for activities
- Scheduled activity (something happening at a specific date and time during the enrollment period)

With these areas, the guardians and the employees of the Boys and Girls Club would experience an enhanced ability.

Advantages- Guardians

With the system, guardians would be empowered to do the following:

- Manage Emergency Contacts that are allowed to pick up their children
- Input contact information for themselves, emergency contacts, and the students
- View available activities for the student
- View activities that the student is enrolled in
- View the payment history for the student's activities

Some other items that could be included in the future include the ability to facilitate communication between the staff and guardians. Communications from the staff about activities that the students are enrolled in (e.g., an away game for Soccer).

Advantages- Staff

There are several advantages that could be achieved for the staff with such a system:

- The ability to see who is scheduled for what activities from a staffing perspective and how many students are enrolled
- Determine future staffing levels based on enrollment
- Quickly find a student's guardian information
- Provide invoices for guardians
- Schedule courses for the future
- See guardians who are behind on payments

As stated previously, facilitating communication between staff and guardians could be a second phase item that could foster better relationships.

System Design

For this project, I focused on a specific area of the problem, the ability to enroll students in activities for a particular period of time.

Domain

The "Enrollment Handling" app will follow many of the strictures of the allocations app.

Models

I designed it to have the following models for this app:

- Activity: An activity that can be enrolled into. When enrolling a student, we ensure that the activity exists and is active.
- Enrollment Period: This is the period in time that represents the activity. We track which activity it is associated with, class capacity, and the current enrollment level.
- Enrollment: This represents the student's enrollment. In addition to identifying the student, the price paid and the remaining balance are also tracked. We also validate that the related activity is still active and that the capacity of the enrollment period won't be exceeded if the enrollment occurs.

Events

These events are sent out to subscribers to let them know about specific changes that were implemented:

- New Enrollment: This event would be picked up by the system that tracks enrollment periods to increase the current enrollment. It communicates the student enrolled, the enrollment period, and the Id of the enrollment record. The price and balance are also included, which could be used to track receipts for the activity for the period of time.
- Enrollment Changed: Similar to new enrollment, but it is sent whenever there's a change to the enrollment.
- Payment Applied: This system is notified whenever a payment is applied, then updates the balance. This message is effectively a cross-acknowledgment that the balance has been updated.

Commands

The following commands are what can be initialized.

- Activity: Tracked as a read-only view so we can use it as a cross-check to make sure that we only enroll students into activities that are still active.
    - Add: Command will be initiated if a subscribed message **activity_added_published** is received that a new activity has been added. The activity will be inserted into the read-only view.
    - Change: Command will be initiated if a subscribed message **activity_updated_published** is received that an activity has been changed. The read-only version of the activity in the view will be updated.
    - Delete: Command will be initiated if a subscribed message **activity_deleted_published** is received that an activity has been changed. It will result in the removal of the read-only activity from the view.
- Enrollment: Tracked as both a read-only view as well as an active session table.
    - Enroll Student: The command is instigated by the API, **/enroll/add**. It uses a post with information about the enrollment to initiate the enrollment.
    - Change Student Enrollment: The command is instigated by the API, **/enroll/<enrollment_id>/edit**. It uses a post with information about how the enrollment should be modified.

    - Apply Payment: The command is Instigated by a subscription to **enrollment_payment_published**, which will communicate the enrollment Id and the amount that needs to be applied.
- Enrollment Period: Tracked as a read-only view, we can use it to cross-check enrollments to ensure that the connection to activity is accurate and that the capacity will not be exceeded if an enrollment occurs.
    - Add: The command will be initialized when receiving a message to the **enrollment_period_added_published** channel. It will insert an enrollment period record to the read-only view.
    - Update: The command will be initialized when receiving a message to the **enrollment_period_updated_published** channel. It will update the enrollment period record in the read-only view.
    - Remove: The command will be initialized when receiving a message to the **enrollment_period_deleted_published** channel. It will delete the enrollment period record from the read-only view.

Adapters

I followed the same pattern as allocate from an adaptor perspective. I will go into depth about the ORM and repository.

ORM

I created four tables in the ORM.

- Activities_view: Used as a read-only representation of the activities. It is used for cross-validation when creating an enrollment to ensure that activities are still active. It is set via an externally published event.
- Enrollments: Used to track the enrollments and make modifications.
- Enrollments_view: Used as a read-only representation of the enrollments table and is used for "gets" from external points.
- Enrollment_peroids_view: Used as a read-only representation of the enrollment periods. It is used for cross-validation when creating an enrollment. It is set via an externally published event.

Repository

I maintained most of the code as is, replacing obvious area's (e.g., products with enrollments) and added one additional function that looks up based on an enrollment period id and the student id, just in case we don't have the id of the enrollment.

Entry Points

There are two entry points

API (flask_app)

There are three methods defined here, all of the endpoints begin with **/enroll/**:

- add (post): This end point will add a new enrollment
- <enrollment_id>/get (get): This endpoint will retrieve the enrollment Id passed
- <enrollment_id>/edit (post): This endpoint will edit the enrollment

REDIS (redis_eventconsumer)

There are several subscriptions consumed in this area. Based on the type, the appropriate command is initialized. All of the commands can be found in the commands section.

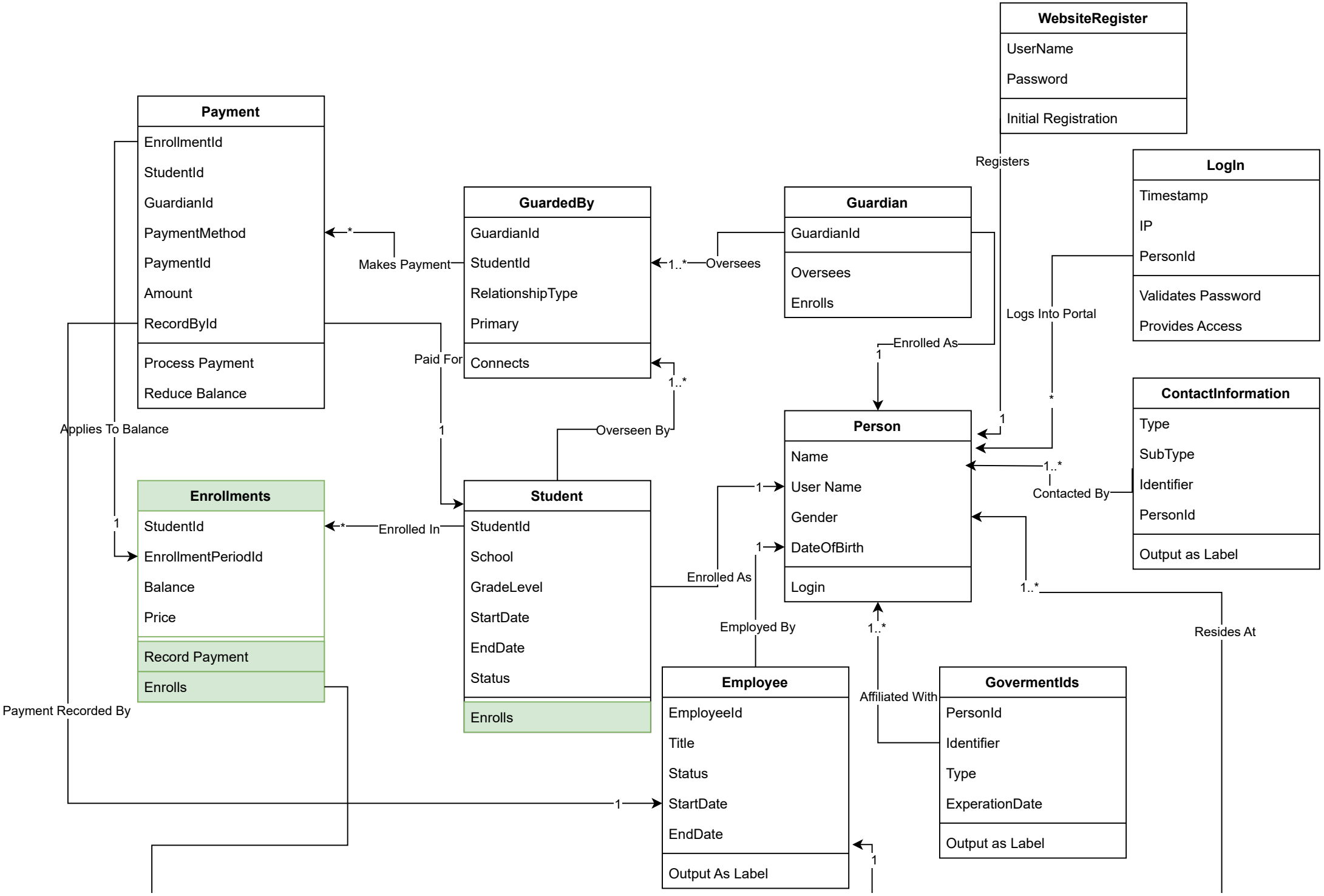
Service Layer

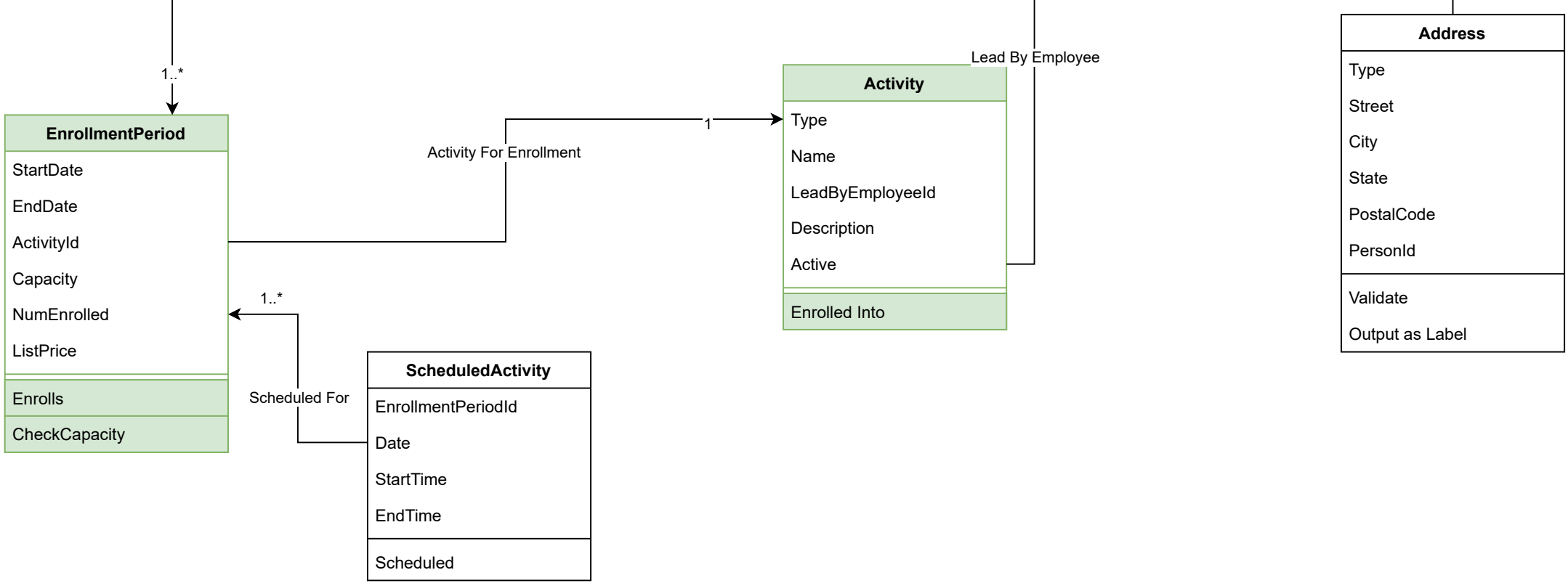
The service layer remained largely unchanged outside of the handler class. I made a few adjustments to refer to the correct terminology with the updated model (e.g., enrollments vs products). The handler class itself differed greatly from what was originally put together for allocations. Obviously, this is because the logic is completely different between the two. All of the code in the handler class is described as either a command or event since those areas are what cause the handler to be initialized, and the command or event call upon functions within the handler class itself.

Testing

I outlined two end-to-end tests that will test the API and external events. I was unable to get them running successfully, so I added commentary in general on what would be tested. I also added a unit test for handlers (attempting to follow the same pattern as the allocation); however, I ran into issues getting the tests up and running. The basic logic is outlined from within the unit tests themselves. Mainly I attempted to test some of the basic scenarios. I didn't go into too much detail outside of handlers because I felt like the areas that would be getting tested elsewhere would be covered with the handler tests.

Apendix


WebsiteRegister
UserName
Password
Initial Registration
Registers
Payment
EnrollmentId
StudentId
GuardianId
PaymentMethod
PaymentId
Amount
RecordById
Process Payment
Reduce Balance
Makes Payment
GuardedBy
GuardianId
StudentId
RelationshipType
Primary
Connects
Oversees
Guardian
GuardianId
Oversees
Enrolls
LogIn
Timestamp
IP
PersonId
Validates Password
Provides Access
Logs Into Portal
Enrolled As
Paid For
Applies To Balance
Overseen By
Person
Name
User Name
Gender
DateOfBirth
Login
ContactInformation
Type
SubType
Identifier
PersonId
Output as Label
Contacted By
Enrollments
StudentId
EnrollmentPeriodId
Balance
Price
Record Payment
Enrolls
Enrolled In
Student
StudentId
School
GradeLevel
StartDate
EndDate
Status
Enrolls
Enrolled As
Employed By
Resides At
Payment Recorded By
Employee
EmployeeId
Title
Status
StartDate
EndDate
Output As Label
Affiliated With
GovermentIds
PersonId
Identifier
Type
ExperationDate
Output as Label


1..*
Lead By Employee
Activity
Type
Name
LeadByEmployeeId
Description
Active
Enrolled Into
1
Activity For Enrollment
EnrollmentPeriod
StartDate
EndDate
ActivityId
Capacity
NumEnrolled
ListPrice
Enrolls
CheckCapacity
1..*
Scheduled For
ScheduledActivity
EnrollmentPeriodId
Date
StartTime
EndTime
Scheduled
Address
Type
Street
City
State
PostalCode
PersonId
Validate
Output as Label