# App Development Frameworks
# Project 1: Console Spring Project

R00159222 - Zachary Dair
zachary.dair@mycit.ie

## Pre-Project Information:

The assignment was done in Intellij, using JDK (OpenJDK 15).

## Database Design

The database consists of three tables:

1. Household (contains ID, eircode, address)
2. Person (contains ID, name, age, occupation)
3. OccupantRecords (contains ID, householdID, personID)

We use the junction table OccupantRecords to store the foreign keys of the household and the person(s) living in that house.

### Design Considerations:

All ID fields auto increment, the eircode and address are set as not null, and there is no unique checking to ensure no eircode is the same, as I believe some buildings can have the same eircode.

The personName field is also left as a single string allowing the user to input first, middle and last names if they wish.

## Project Structure

The structure follows convention that was shown in the lectures

A main source code package:

src/main/java/com.dair (is the main overall package storing the source code)

Inside this package there are several other packages:

1. Classes
2. Dao
3. Rowmappers
4. Service

And 3 classes, that run aspects of the project:

Main (Handles looping the application), Menu (Handles displaying the main menu and getting an input option) and Controller (Handles the processing of each specific task from the menu)

src/main/resources (contains beans.xml for bean creation, data.sql for populating the database, and schema.sql for creating the database structure)

src/test/java/classTests and src/test/java/databaseTests (for the unit tests files)

## Project Checklist

- **Menu System** (Found in the Menu Class and in Main as switch cases)
- **Searching by Eircode** (Found in Controller, called from Main)
- **Adding a new household and Occupants**
    - Takes input for Household, uses the DB returned ID to add a primary key
    - Loops allowing multiple occupants to be added
    - Each occupant is added to the occupant records table, with foreign keys
- **Adding a new person and assign to a household**
    - Takes input for Person, uses the DB returned ID to add a primary key
    - Asks if the user wants to add a new household, or view existing
    - Either creates a house and updates occupantRecords, or just update the records
- **Move a person**
    - Displays existing people, prompts for ID
    - Displays existing houses, prompts for ID
    - Finds and updates the occupantRecords for that personID with the new houseID

    - Prompts option to display new household details
- **Delete a household**
    - Displays existing houses, prompts for ID
    - Retrieves any occupant's IDs for that household
    - Removes each occupantRecord entry for the houseID and personID
    - Removes each person from the person table by ID
    - Finally removes the Household
- **Delete a person**
    - Displays existing people, prompts for ID
    - Finds and removes the occupantRecords for that personID
- **Display Statistics**
    - Uses SQL AVG query to find the average age and of the persons table
    - Uses SQL COUNT query to count all personID's where the occupation is set to pre-school, and again for scholar, and returns these values
    - Uses SQL COUNT query to count all personID's with an age greater than or equal to 65
    - Each of these stats are then displayed, along with a total student count (pre-school + scholar)
    - Total count of households, and persons

When creating a person if the age is between 0 and 5 it set's the occupation to 'Pre-School'
If the age is above 5 but less than 18 the occupation is set to 'Scholar'
When adding an occupation for any other age, the user is asked if they are a scholar to allow for normalized 'Scholar' entries in the database for the statistics.

**The Unit tests** are found in src/test/java/classTests and src/test/java/databaseTests
**BaseClassTests**, four basic object creation tests for household and person objects.
**CoreDatabaseTests**, two tests checking the tables are created and populated

**HouseholdDaoTests**, three tests finding total count, finding by ID and removing by ID(including occupants) ***DAO Layer Tests**

**PersonServiceTests**, three tests deleting all persons (including occupantRecords), finding the average age when persons is empty (normally throws a null pointer exception, but our service layer handles that logic), searching for a job that is not in persons table. ***Service Layer Tests**