УДК 004.6

Поварницын Е.Н., студент

4 курс, факультет «Информационные системы и технологии»

Северный Арктический Федеральный Университет

Россия, г. Архангельск

Povarnitsyn E. N., student

4rd year, faculty of Information systems and technologies»

Northern Arctic Federal University

Russia, Arkhangelsk

РАЗРАБОТКА КЛИЕНТ-СЕРВЕРНОЙ АРХИТЕКТУРЫ ИС ДЛЯ РАСЧЁТА ЗАРАБОТНОЙ ПЛАТЫ С ИСПОЛЬЗОВАНИЕМ ПАТТЕРНА MVC

Development of a client-server architecture of IS for calculating wages using the MVC pattern

Аннотация:

Статья посвящается анализу разработки клиент-серверной архитектуры ИС. В ней детально рассматриваются все этапы разработки. А также генерацию базового кода для работы.

Ключевые слова: разработки, Java, MVC, программирование, анализ.

Annotation:

The article is devoted to the analysis of the development of the client-server architecture of the IS. It covers all stages of development in detail. As well as generating the basic code to work.

Keyword: development, Java, MVC, programming, analysis.

1 ВАРИАНТЫ ИСПОЛЬЗОВАНИЯ ИС

Осуществим анализ предметной области на основе объектно-ориентированного подхода с помощью программы Bouml. Для этого в данной программе создаём диаграмму вариантов использования.

В данной диаграмме создаём двух актёров. Administrator(Администратор) и Worker(Сотрудник). Данные действующие лица относятся к предметной области – системы управления организацией. Администратором будет тот, кто может управлять базами данных, редактировать их. Следовательно, Сотрудник тот, над кем будут происходить действия, то есть начисления заработной платы, списание налогов, должность и так далее. Создаём их, в соответствии с рисунком 1.

Рисунок 1 – Актёры Administrator и Worker

После создадим первый вариант использования «Authorization» (Авторизация пользователя). Она соответствует в ИС функции предоставление информационных ресурсов пользователям. Так как авторизация есть и у администратора и у сотрудника используем отношение ассоциации к обоим, в соответствии с рисунком 2.

Рисунок 2 – Authorization

После чего создаём вариант использования для администратора. Назовём его «Management account» (Управление аккаунтом). Администратор будет отвечать, как уже было сказано выше, за редактирование, удаление и создание новых пользователей. Используем между вариантом использования и администратором отношение ассоциацию, в соответствии с рисунком 3.

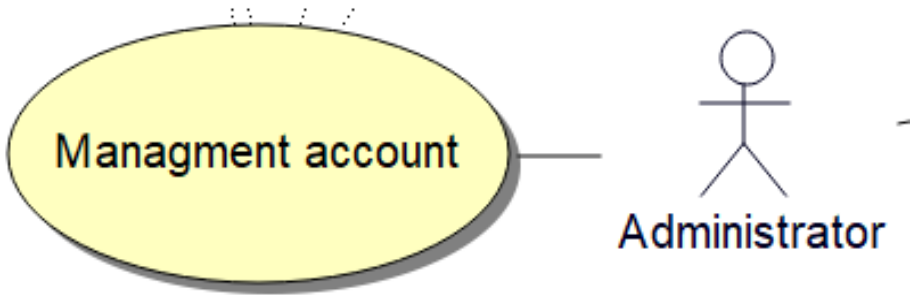

Рисунок 3 – Management Account

Вариант использования «Management account» включает в себя такие варианты использования как «Delete account»(Удаление аккаунта), «Change account»(Изменение аккаунта), «View account»(Просмотр аккаунта), «Add account»(Добавление аккаунта). В ИС управление аккаунтом соответствует функции сбора информации о пользователе. Зависимость с ними будет называться включение. Так же администратор будет управлять резервным копированием. Поэтому так же создадим вариант использования «Backup managment» (Управление резервным копированием») Он включает в себя «Delete backup»(Удаление резервного копирования), «Add backup»(Добавление резервного копирования) и «View backup»(Просмотр резервного копирования). Создадим данные варианты использования, в соответствии с рисунком 4.

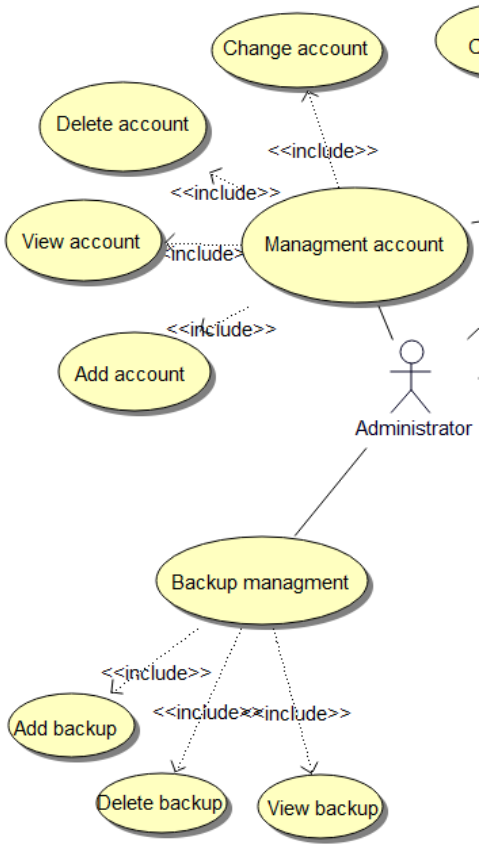

Рисунок 4 – Зависимость включение

Далее создаём варианты использования для сотрудника. Первый вариант использования будет «Post management»(Управление должностью). Используем между сотрудником и вариантом использования отношение ассоциацию и имеет отношение ассоциация с вариантом использования «Management history», которую мы создадим далее. Так же вариант использования «Post management» включает в себя такие варианты использования как «Change post»(Изменить должность), «Delete post»(Удалить должность), «Add post»(Добавить должность), «View post»(посмотреть должность). Создадим данный вариант использования, в соответствии с рисунком 5.

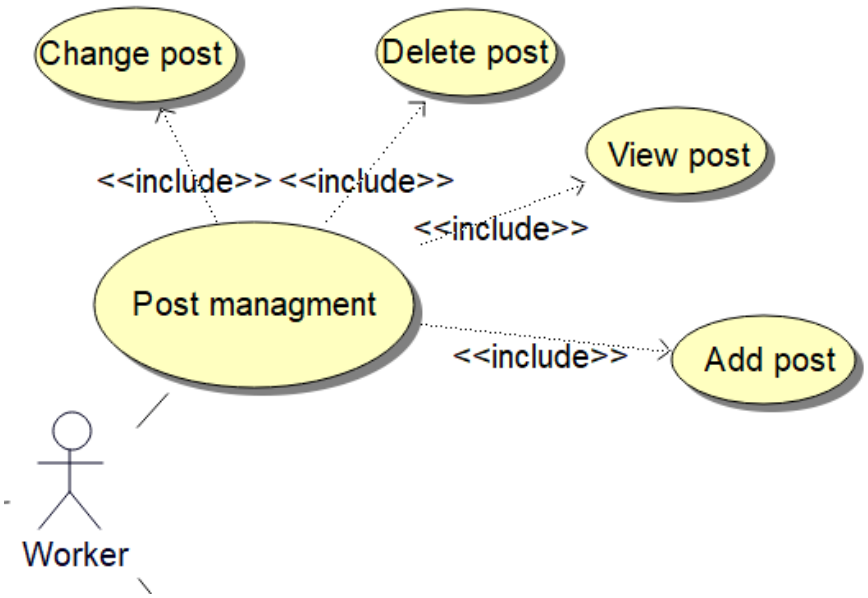

Рисунок 5 – Post management

Далее создаём вариант использования «Management worker»(Управление сотрудниками). Он будет относится к администратору. Для сотрудников определяется должность. Используем между сотрудником и вариантом использования отношение ассоциацию, и используем ассоциацию данного варианта использования с вариантом использования «Management account». Так же вариант использования «Management user» включает в себя такие варианты использования как «Change user»(Изменить пользователя), «Delete user»(Удалить пользователя), «Add user»(Добавить пользователя), «View user (посмотреть пользователя). Создадим данный вариант использования, в соответствии с рисунком 6.

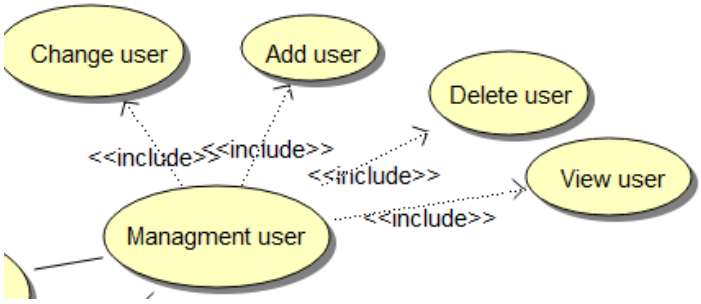

Рисунок 6 – «Management user»

Далее создаём вариант использования «Management fee and taxes»(Управление доплатной и налогами). Для сотрудников определяется должность. Используем между сотрудником и вариантом использования

отношение ассоциацию. Так же вариант использования «Management fee and taxes» включает в себя такие варианты использования как «Change fee and taxes»(Изменить доплату и налог), «Delete fee and taxes»(Удалить доплату и налог), «Add fee and taxes»(Добавить доплату и налог), «View fee and taxes (посмотреть доплату и налог). Создадим данный вариант использования, в соответствии с рисунком 7.

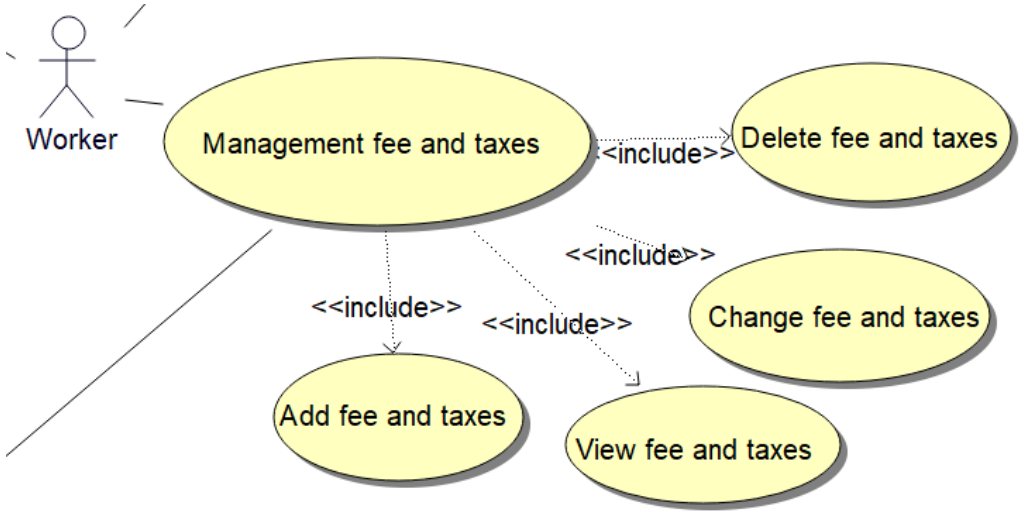

Рисунок 7 – Management fee and taxes

После чего создаём последний вариант использования «Managment history»(Управление доплатной и налогами). Используем между сотрудником и вариантом использования отношение ассоциацию и имеет отношение ассоциация с вариантом использования «Post managment» . Так же вариант использования «Managment history» включает в себя такие варианты использования как «Change history»(Изменить историю), «Delete history»(Удалить историю), «Add history»(Добавить историю), «View history (посмотреть историю). Создадим данный вариант использования, в соответствии с рисунком 8.

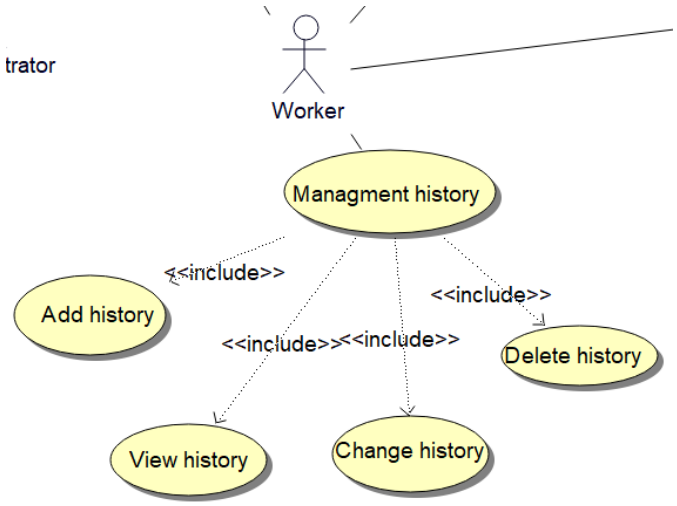

Рисунок 8 – Management history

После чего мы получаем полную диаграмму вариантов использования, в соответствии с рисунком 9.

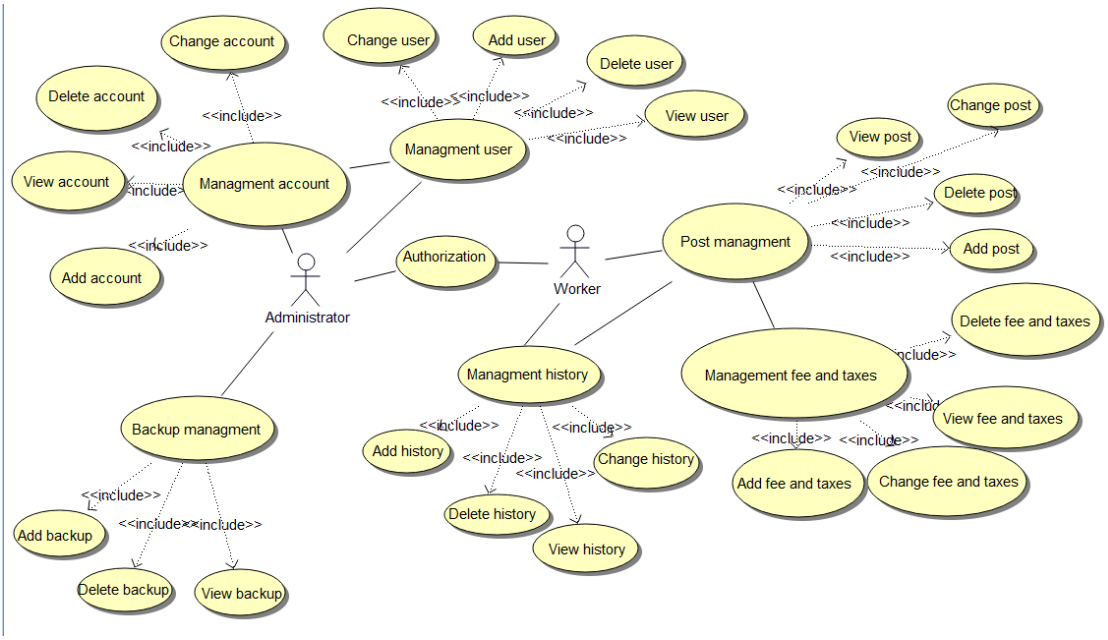

Рисунок 9 – Диаграмма вариантов использования

2 ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ИС

Создадим диаграмму классов для каждого варианта использования, которые используют отношение ассоциацию. Это такие варианты использования как «Management account», «Post managment», «Management fee and taxes», «Management history», «Management worker», «Backup managment».

Сначала, рассмотрим классовую диаграмму для «Management account». В нём создаём класс «Account». В этом классе добавляем необходимые классы и атрибуты. Добавляем атрибуты такие как id_account (номер аккаунта), login (логин пользователя), password (пароль пользователя). В операции добавляем геттеры атрибутов. Геттеры – это методы для считывания данных из атрибутов. Также же стоит к операциям отнести конструкторы. Один из них без аргументов и инициализирующий атрибуты значений по умолчанию, а второй с аргументами для каждого атрибута. После чего у нас получается класс в соответствии с рисунком 9.

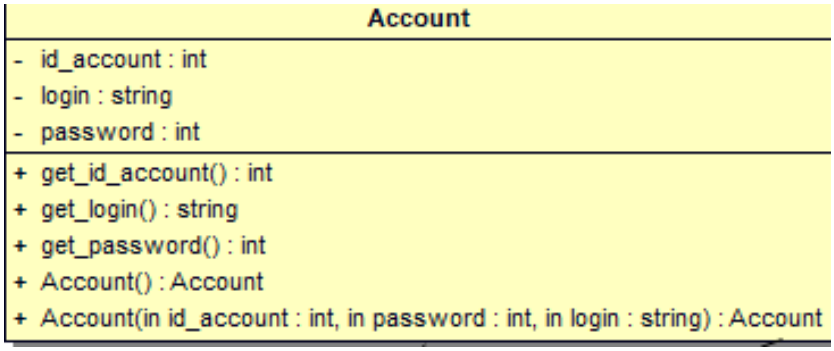

Рисунок 9 – Account

После чего создаём класс «Database». В этом классе добавляем необходимые классы и атрибуты. Добавляем атрибут conn, он позволяет хранить подключение к базе данных. Далее добавляем операции openConnection и closeConnection будут отвечать за открытие и закрытие базы данных, newAccount, updateAccount,deleteAccount и getAllAccount будут отвечать соответственно за добавление, обновление, удаление и просмотр данных об аккаунте, так же следует добавить Authorization. Следует это всё оформить, в соответствии с рисунком 10.

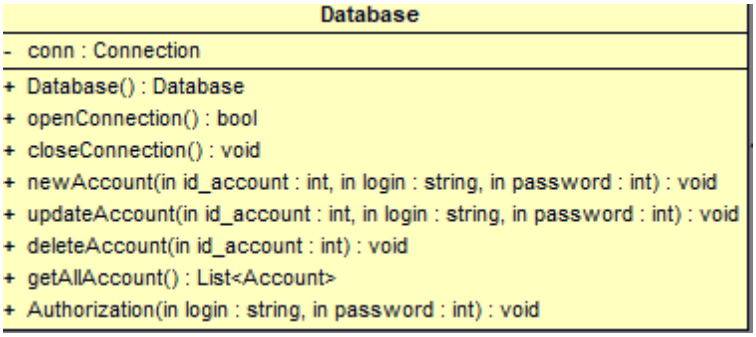

Рисунок 10 – Database

После создаём класс «View». В нём следует создать такие атрибуты как tvAccount, который будет позволять отображать данные всех аккаунтов в табличном виде, tfIdAccount, tfLogin, tfPassword они необходимы для редактирования соответствующих атрибутов клиента. Атрибуты bNew, bEdit, bLogin и bDelete для вызова действий добавлению, редактированию, логина и удаления данных. В итоге у нас должен получиться класс, в соответствии с рисунком 11.

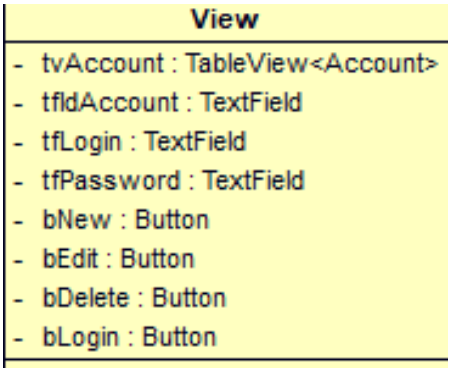

Рисунок 11 – View

И создаём последний класс «Controller». В нём создаём атрибуты db и v. Они связывают контроллер с классами модели и вида. Операции handleNew, handleUpdate и handleDelete, handleAuthorization отвечают за обработку нажатия на соответствующие кнопки в виде. И оператор initialize отвечает за инициализацию контроллера. Получаем класс, в соответствии с рисунком 12.

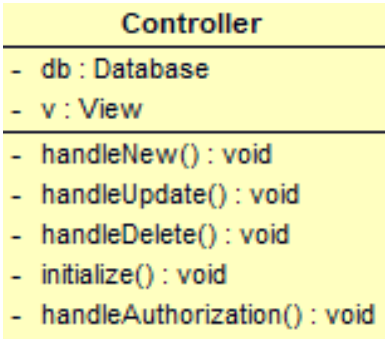

Рисунок 12 – Controller

После создания всех классов следует указать отношения между ними. Между классами Conrtoller и Database, Controller и View отношение направленной ассоциации, поскольку в классе Controller присутствуют два атрибута типа Database и View. Отношение зависимости указана между классами Database и Client, Controller и Account, поскольку в методах этих классов будет использоваться Account. В итоге получаем классовую диаграмму, в соответствии с рисунком 13.

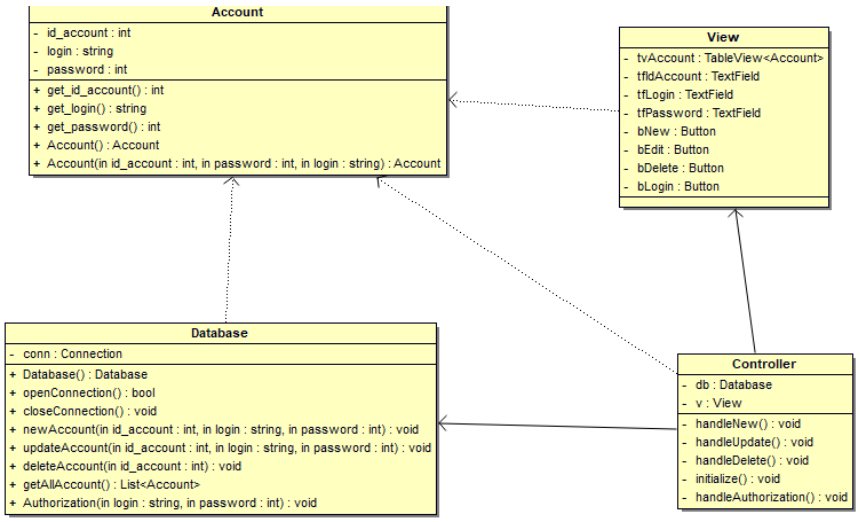

Рисунок 13 – Отношения между классами

После чего расставим стереотипы для классов. Классы, относящиеся к модели, будут иметь стереотип «entity» (сущность), это классы Account и Database, классы относящиеся к виду будут иметь стереотип «boundary» (граничный), это класс View, и классы относящиеся к контроллеру будут иметь стереотип control (управляющий). Вот мы и получаем готовую диаграмму класса для «Management account», в соответствии с рисунком 14.

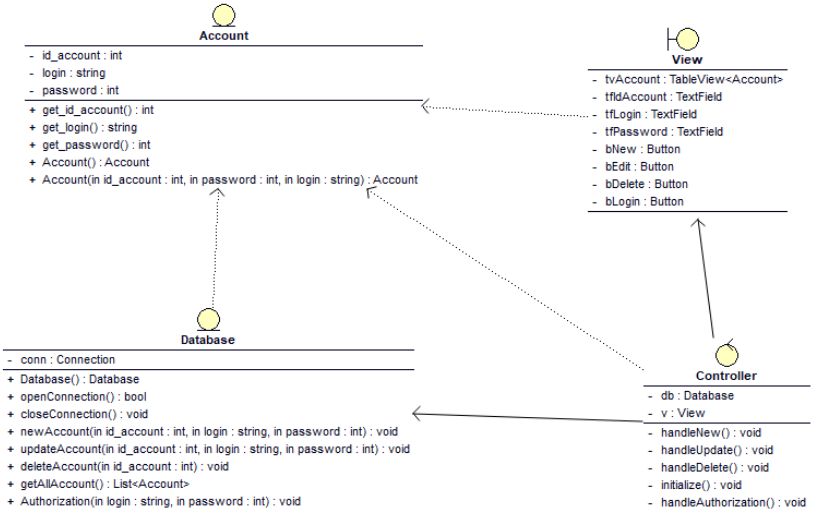

Рисунок 15 – Диаграмма классов для «Management account»

Далее разберём создание классовой диаграммы для «Post management». В Database нужно добавить newPost, updatePost,deletePost и getAllPost будут отвечать соответственно за добавление, обновление, удаление и просмотр данных об должности сотрудника и т.д. , так же стоит добавить атрибут conn. И добавить операции Database, openConnection, closeConnection. В View следует дописать свои атрибуты, которые будут отображать данные в табличном виде это: tfPost, tfDischarge, tfSalary tfUnion они отвечают за редактирование соответствующих атрибутов, post, discharge, salary, Union, которые мы добавим позже, также стоит написать атрибуты для вызова действий bEdit, bDelete, bNew. Controller будет аналогичен с прошлым Controller. После следует создать класс Post и в нём прописать атрибуты такие как post, который отвечает за должность и который использует связь с User, discharge, который отвечает за разряд сотрудника, и salary, который отвечает за заработанную плату сотрудника, Union, который отвечает за то что состоит ли сотрудник в Профсоюзе. Далее для них следует создать методы геттеры и конструкторы, которые мы разбирали выше. После чего добавляем недостающие классы, указываем отношения и расставляем стереотипы. В итоге у нас должна получиться классовая диаграмма, в соответствии с рисунком 16.

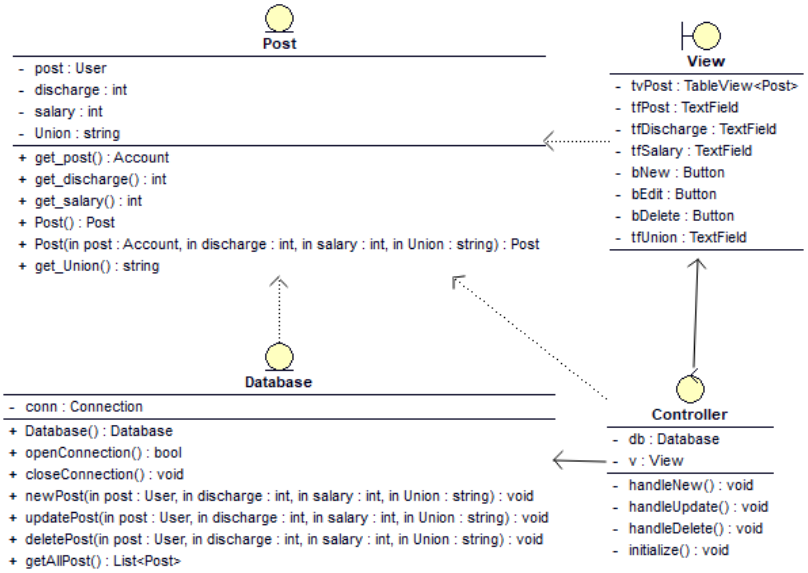

Рисунок 16 – Диаграмма классов для «Post management»

Далее разберём создание классовой диаграммы для «Management worker». В Database нужно добавить newUser, updateUser, deleteUser и getAllUser будут отвечать соответственно за добавление, обновление, удаление и просмотр данных о сотруднике, так же стоит добавить атрибут conn. И добавить операции Database, openConnection, closeConnection. В View следует дописать свои атрибуты, которые будут отображать данные в табличном виде это: tfFirstName, tfLastName, tfBirthday, tfPost, tfLogin, tfFullPayment, tfUnion они отвечают за редактирование соответствующих атрибутов first_name, last_name, birthday, post, login, fill_payment, Union которые мы добавил позже, также стоит написать атрибуты для вызова действий bEdit, bDelete, bNew. Controller будет аналогичен с прошлым Controller. После следует создать класс User и в нём прописать атрибуты такие как first_name, который за имя сотрудника, last_name, который отвечает за фамилию сотрудника birthday, который отвечает дату рождения сотрудника и post, который отвечает за должность сотрудника, login, который отвечает за логин сотрудника который связан с классом Account, full_payment, который отвечает за итоговую расчёт с сотрудником, Union, который отвечает за то состоит ли сотрудник в профсоюзе или нет, который связан с Post. Далее для них следует создать методы геттеры и конструкторы, которые мы разбирали выше. После чего добавляем недостающие классы, указываем отношения и расставляем стереотипы. В итоге у нас должна получиться классовая диаграмма, в соответствии с рисунком 17.

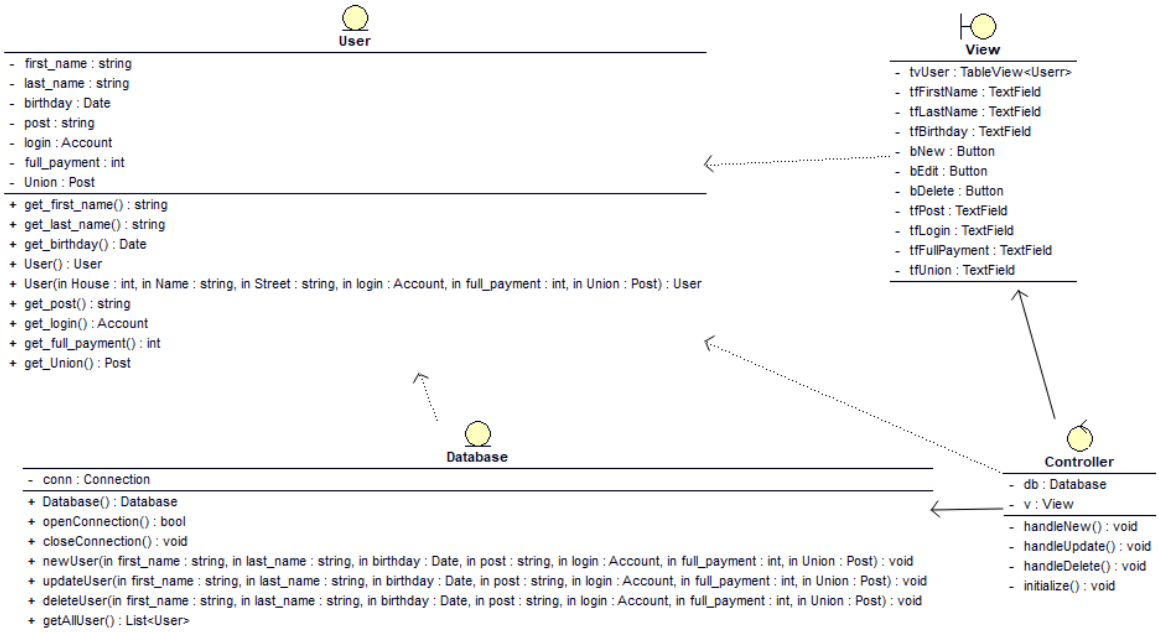

Рисунок 17 – Диаграмма классов для «Management user»

Далее разберём создание классовой диаграммы для «Management fee and taxes». В Database нужно добавить newFee_Taxes, updateFee_Taxes, deleteFee_Taxes и getAllFee_Taxes будут отвечать соответственно за добавление, обновление, удаление и просмотр данных об зарплате и налогах, так же стоит добавить атрибут conn. И добавить операции Database, openConnection, closeConnection. В View следует дописать свои атрибуты, которые будут отображать данные в табличном виде это: tfUralCoeficient, tfTax, tfPensionFund, tfTradeUnion, tfLastName они отвечают за редактирование соответствующих атрибутов ural_coeficient, tax, pension_fund, trade_Union, last_name которые мы добавил позже, также стоит написать атрибуты для вызова действий bEdit, bDelete, bNew. Controller будет аналогичен с прошлым Controller. После следует создать класс Fee_Taxes и в нём прописать атрибуты такие как ural_coeficient, который отвечает за уральский коэффициент, tax, который отвечает за подоходный налог, pension_fund, который отвечает за пенсионные выплаты, и trade_Union, который отвечает за отчисление профсоюзу, если человек состоит в нём, last_name, который отвечает за фамилию сотрудника, который связан с классом User. Далее для них следует создать методы геттеры и конструкторы, которые мы разбирали выше. После чего добавляем недостающие классы, указываем отношения и расставляем стереотипы. В итоге у нас должна получиться классовая диаграмма, в соответствии с рисунком 18.

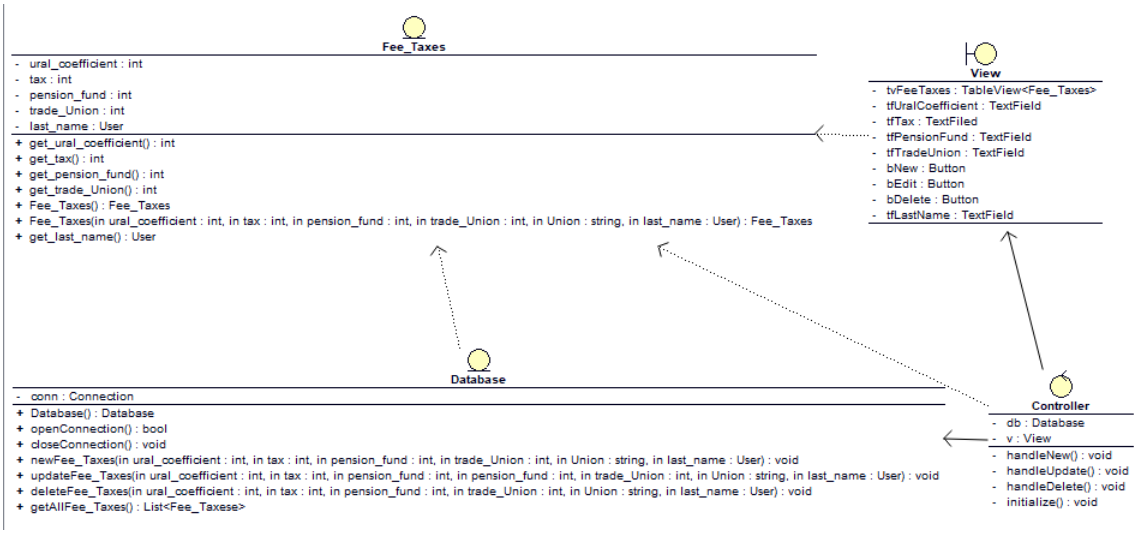

Рисунок 18 – Диаграмма классов для «Management fee and taxes»

Далее разберём создание классовой диаграммы для «Management history». В Database нужно добавить newHistory, updateHistory, deleteHistory и getAllHistory будут отвечать соответственно за добавление, обновление, удаление и просмотр данных об должности сотрудника и т.д., так же стоит добавить атрибут conn. И добавить операции Database, openConnection, closeConnection. В View следует дописать свои атрибуты, которые будут отображать данные в табличном виде это: tfTotalAccrued, tfWithheld, tfFullPayment, tfDate они отвечают за редактирование соответствующих атрибутов total_accrued, withheld, full_payment, date, которые мы пропишем позже, также стоит написать атрибуты для вызова действий bEdit, bDelete, bNew. Controller будет аналогичен с прошлым Controller. После следует создать класс History и в нём прописать атрибуты такие как total_accrued, который отвечает сколько всего начислено, withheld, который отвечает сколько удержано, и full_payment, сколько выплачено сотруднику после вычетов всех сервисов, который связан с классом User, date, который отвечает за период отчётности. Далее для них следует создать методы геттеры и конструкторы, которые мы разбирали выше. После чего добавляем недостающие классы, указываем отношения и расставляем стереотипы. В итоге у нас должна получиться классовая диаграмма, в соответствии с рисунком 19.

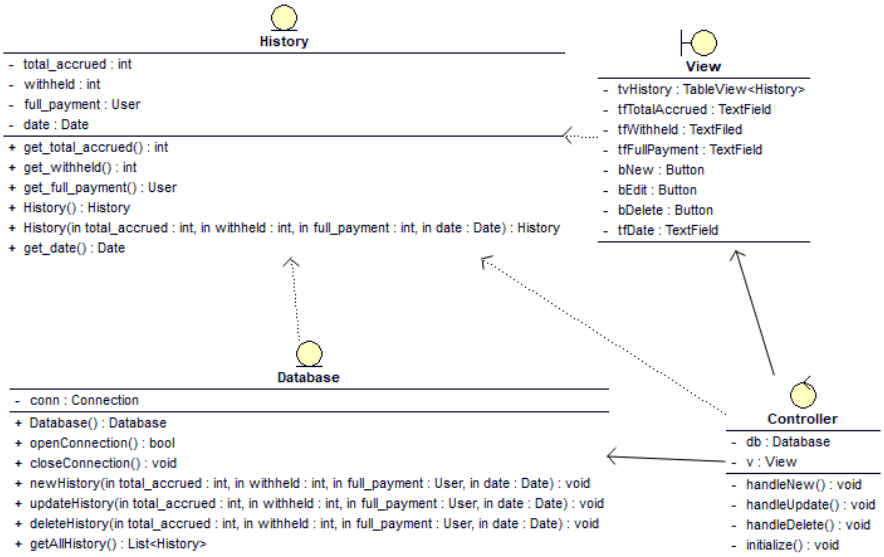

Рисунок 19 – Диаграмма классов для «Management history»

За авторизацию отвечает класс «Account» и метод «Authorization, который использует bLogin, который находиться в View.

Далее разберём создание классовой диаграммы для «Backup managment». В Database нужно добавить newBackup, deleteBackup и getAllBackup будут отвечать соответственно за добавление, обновление, удаление и просмотр данных об резервном копирование, так же стоит добавить атрибут conn. И добавить операции Database, openConnection, closeConnection. В View следует дописать свои атрибуты, которые будут отображать данные в табличном виде это: tfDataBackup, tfBackupDescription они отвечают за редактирование соответствующих атрибутов data_backup, backup_description, которые мы пропишем позже, также стоит написать атрибуты для вызова действий bEdit, bDelete, bNew. Controller будет аналогичен с прошлым Controller. После следует создать класс Backup и в нём прописать атрибуты такие как dataBackup, который отвечает за дату бэкапа, backup_description, который отвечает за описание бэкапа,. Далее для них следует создать методы геттеры и конструкторы, которые мы разбирали выше. После чего добавляем недостающие классы, указываем отношения и расставляем стереотипы. В итоге у нас должна получиться классовая диаграмма, в соответствии с рисунком 20.

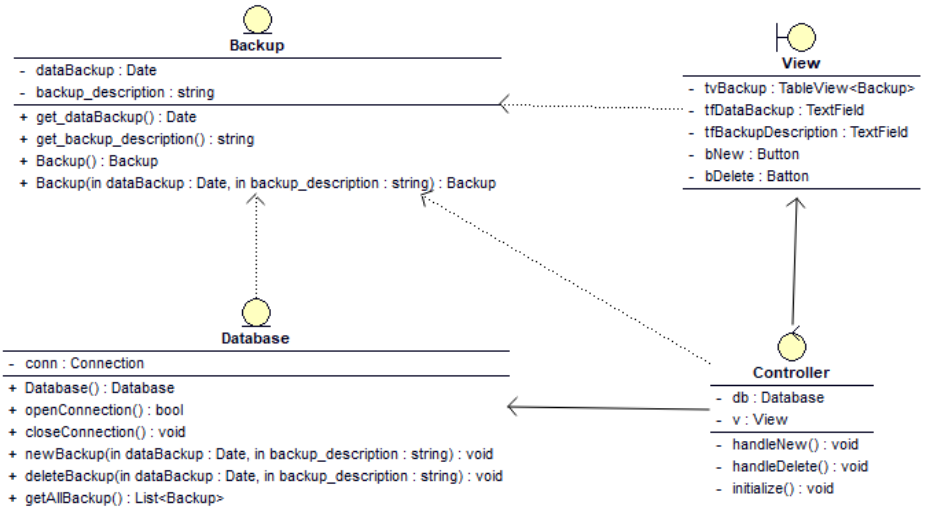

Рисунок 20 – Диаграмма классов для «Backup managment»

Создаём диаграмму компонентов и развёртывания.

Сначала следует создать «Component diagram» (диаграмму компонентов). Создадим для данной архитектурной системы application, которая будет являться клиентом и компонент database, который будет рассматриваться в качестве сервера и будет предоставлять нужную информацию клиенту. Далее выберем для application требуемые классы. Для меня это «Account», «Post», «Fee_Taxes», «Worker» и «History». Это означает, что компоненту, отвечающего за работу приложения, требуется соответствующая таблица в базе данных. Далее необходимо предоставить данные классы в компоненте database. Это означает, что компонент, отвечающий за базу данных, предоставляет таблицу для работы компонента, отвечающего за приложение. Далее отразим отношение на диаграмме компонентов. В нём один требует, а другой предоставляет. После чего зададим стереотипы для компонентов. Поскольку в компоненте database расположен класс сущность, то для данного компонента выберем соответствующий стереотип. Для компонента application выбран стереотип процесс для характеристики работающего приложения. После чего у нас должна получиться диаграмма компонентов, в соответствии с рисунком 21.

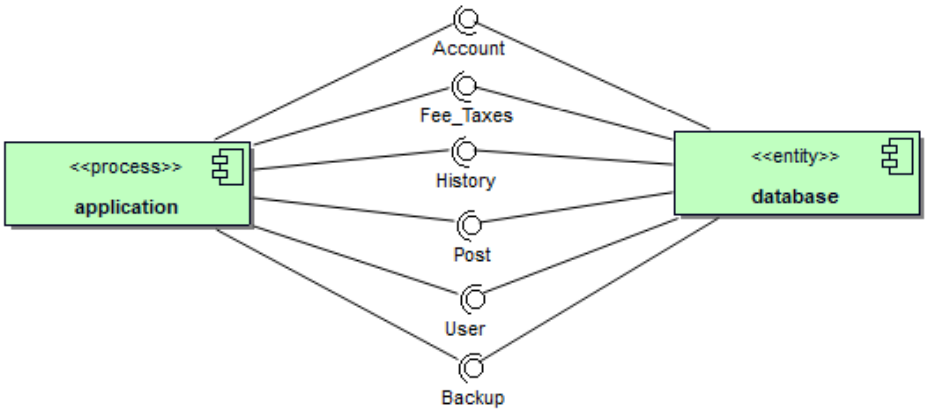

Рисунок 21 – Диаграмма компонентов

Далее перейдём к проектированию физической архитектуры, которая осуществляется с помощью «Deploy diagram» (диаграммы развёртывания). Создадим диаграмму развёртывания. В ней добавляем сервер баз данных, который будет называться «nix:DB-Server», и на котором будет размещён компонент database. После чего добавим узды для рабочих станций, назвав их «:Workstation» и соединим все узлы по топологии звезда. Далее разместим в узлах компоненты, в соответствии с рисунком 22.

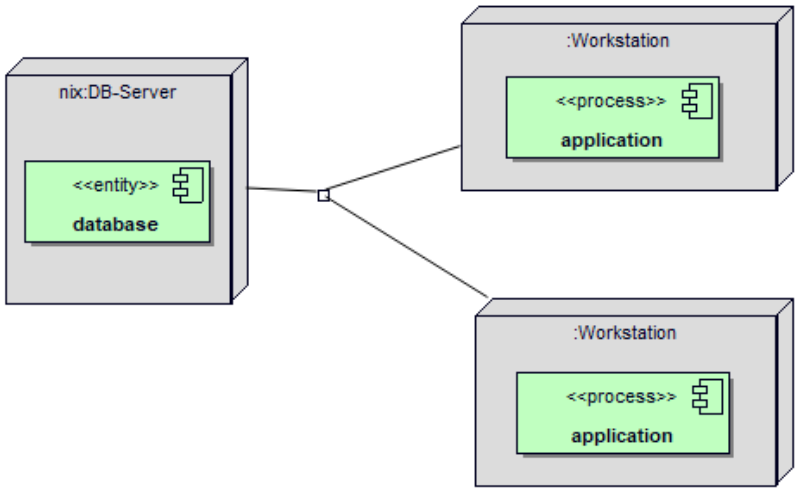

Рисунок 22 – Диаграмма развёртывания

После чего создаём артефакт для генерации кода java на основе классовой диаграммы. В нём необходимо выбрать с какими классами ассоциирован артефакт, в соответствии с рисунком 23.

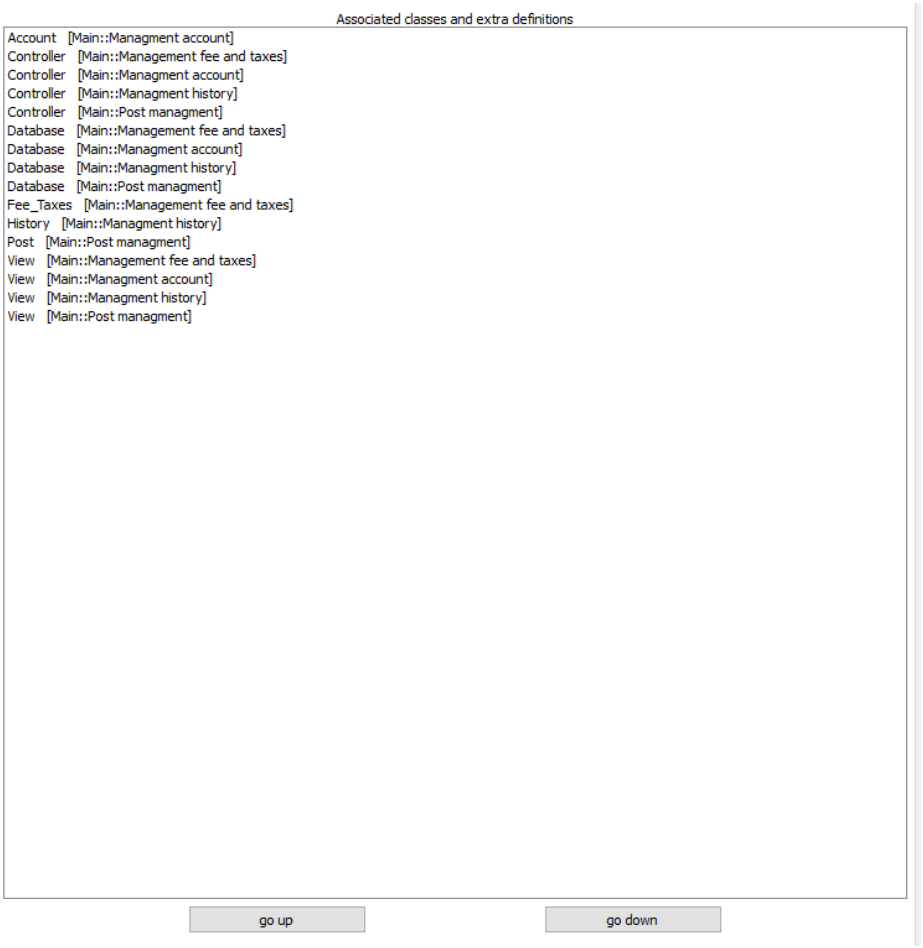

Рисунок 23 – Артефакт

После чего код начинает генерироваться.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Рыбальченко М.В. Архитектура информационных систем. Учебное пособие для ВУЗов [Текст]. – Юрайт,2016 г.

2 Трутнев Д.Р. Архитектуры информационных систем. Основы проектирования [Текст]. – ИТМО, 2012 г.