

Рис. 1 Uml диаграмма развёртывания информационной системы по сбору, хранению и обработке агрометеорологических данных.

В текущем разделе представлено детальное описание системы оперативного мониторинга данных метеостанции.

Цели:

* Отображение данных метеостанции на веб сайте, доступ к которому осуществляется через интернет;
* Проведение описательных статистик по запрошенным данным с метеостанции.

Задачи:

* Разработка информационной системы по сбору, хранению и обработке агрометеорологических данных;
* Развертывание информационной системе на сервере;
* Размещение сайта (представления) системы в интернете.

В отделе 170 АФИ в лаборатории «Сектор средств инструментального контроля» была разработана метеостанция, которая может измерять следующие параметры:

* Температура поверхности земли;
* Солнечное излучение;
* Влажность воздуха;
* Температура воздуха;
* Направление ветра;
* Скорость ветра;
* Влагомер.

Данная метеостанция находится на опытном полигоне в Меньково и имеет источник постоянного питания в виде солнечной батареи. Собранные данные метеостанция отправляет по интернету в хранилище данных, которое физически находится в АФИ, в качестве SQL-сервера выступает PostgreSQL.

SQL сервер содержит единственную таблицу, содержащую измерения под названием measurement и имеет следующую структуру:

**create** **table** measurement

{

id serial **primary** **key**,

measure\_date **date**,

measure\_hour **integer**,

measure\_min **integer**,

t\_pov **numeric**,

far **numeric**,

rh **numeric**,

t **numeric**,

winddir **numeric**,

wind **numeric**

rain **numeric**

}

Где:

measure\_date – дата измерения (ГГГГ-ММ-ДД);

measure\_hour – час измерения (чч);

measure\_min – минута измерения (мм);

t\_pow – температура поверхности почвы (градусы цельсия);

far – датчик освещённости (люкс);

rh – влажность воздуха ();

t – температура воздуха (градусы цельсия);

winddir – направление ветра (градусы цельсия);

wind – скорость ветра (м/с).

rain – количество осадков (мм)

Необходимо предоставить пользователю доступ к просмотру данных метеостанции и для этого необходимо создать клиент-серверную архитектуру, где на стороне сервера будут обрабатываться и храниться данные, а на стороне клиента данные будут представляться. Подобную проблему решает архитектурный паттерн MVC. Термин модель-представление-контроллер (model-view-controller – MVC) был в употреблении с конца 1970-х годов и происходит из проекта Smalltalk в Xerox PARC, где он был задуман как способ организации ряда ранних приложений с графическим пользовательским интерфейсом, и нюансы первоначального паттерна MVC применимы к приложениям – и особенно хорошо к веб-приложениям.

Если оперировать высокоуровневыми понятиями, то паттерн MVC означает, что приложение MVC будет разделено, по крайней мере, на три части.

* Модели, содержащие, или представляющие данные, с которыми работают пользователи.
* Представления, используемые для визуализации некоторой части модели в виде пользовательского интерфейса.
* Контроллеры, которые обрабатывают входящие запросы, выполняют операции с моделью и выбирают представления для визуализации пользователю.

Каждая порция архитектуры MVC четко определена и самодостаточна: такое положение вещей называют разделением обязанностей. Логика, которая манипулирует данными в модели, содержится только в модели. Логика, отображающая данные, присутствует только в представлении. Код, который обрабатывает пользовательские запросы и ввод, находится только в контроллере. Благодаря ясному разделению между частями приложение будет легче сопровождать и расширять на протяжении его времени существования вне зависимости от того, насколько большим он станет.

Модели (M в MVC) содержат данные, с которыми работают пользователи. Существуют два обширных типа моделей: модели представлений, которые выражают сами данные, передаваемые из контроллера в представление, и модели предметной области, которые содержат данные в предметной области наряду с операциями, трансформациями и правилами для создания, хранения и манипулирования данными, вместе называемыми логикой модели. Предметная область в текущем проекте – это агрометеорологические данные.

Контроллеры являются «соединительной тканью» паттерна MVC, исполняя роль каналов между моделью данных и представлениями. Контроллеры определяют действия, предоставляющие бизнес-логику, которая оперирует на модели данных и обеспечивает представления данными, подлежащими отображению для пользователя.

Представления содержат логику, которая требуется для отображения данных пользователю или для сбора данных от пользователя, так что они могут быть обработаны каким-то действием контроллера.

Данный паттерн реализован на платформе ASP .NET 6, то есть он является кроссплатформенным и реализует библиотеку классов, наследуемую от Microsoft.AspnetCore.Mvc.

MVC – не единственный архитектурный паттерн, программного обеспечения. Есть много других паттернов, и некоторые из них чрезвычайно популярны или, по крайней мере, были таковыми. Преимущества MVC можно обнаружить, сравнивая его с альтернативами.

Один из наиболее распространенных паттернов проектирования известен как «Интеллектуальный пользовательский интерфейс» (Smart UI). Большинству программистов приходилось создавать приложение с интеллектуальным пользовательским интерфейсом на том или ином этапе своей профессиональной деятельности. При построении приложения с интеллектуальным пользовательским интерфейсом разработчики конструируют интерфейс, часто перетаскивая набор компонентов либо элементов управления на поверхность проектирования или холст. Элементы управления сообщают о взаимодействии с пользователем, инициируя события для щелчков кнопками мыши, нажатий клавиш на клавиатуре, перемещений курсора и т.д. Разработчик добавляет код реакции на такие события в набор обработчиков событий, которые являются небольшими блоками кода, вызываемыми при выдаче специфического события в определенном компоненте. В конечном итоге получается монолитное приложение.

Код, который поддерживает пользовательский интерфейс и реализует бизнес-логику, перемешан между собой без какого-либо разделения обязанностей. Код, который определяет приемлемые значения для вводимых данных и запрашивает данные или модифицирует, например, расчетный счет пользователя, оказывается размещенным в небольших фрагментах, связанных друг с другом в предполагаемом порядке поступления событий.

Интеллектуальные пользовательские интерфейсы идеальны для простых проектов, т.к. позволяют добиться неплохих результатов достаточно быстро (по сравнению с разработкой MVC, которая требует определенных начальных затрат, прежде чем будут доставлены результаты). Интеллектуальные пользовательские интерфейсы также подходят для построения прототипов пользовательских интерфейсов. Их инструменты визуального конструирования могут оказаться по-настоящему удобными, когда необходимо обсуждать с заказчиком требования к внешнему виду и потоку пользовательского интерфейса, то инструмент интеллектуального пользовательского интерфейса может быть быстрым и удобным способом для генерации и проверки различных идей.

Самый крупный недостаток интеллектуальных пользовательских интерфейсов связан с тем, что их трудно сопровождать и расширять. Смешивание кода модели предметной области и бизнес-логики с кодом пользовательского интерфейса приводит к дублирования кода, когда один и тот же фрагмент бизнес-логики копируется и вставляется для поддержки вновь добавленного компонента. Нахождение всех дублированных фрагментов и применение к ним исправления может быть непростой задачей. Добавление новой функции может оказаться практически невозможным без нарушения работы каких-то существующих функций. Тестирование приложения с интеллектуальным пользовательским интерфейсом также может быть затруднено. Единственный способ тестирования – эмуляция взаимодействия с пользователем, что является далеким от идеала и трудно реализуемым фундаментом для обеспечения полного покрытия тестами.

В мире MVC интеллектуальный пользовательский интерфейс часто называют антипаттерном, т.е. чем-то таким, чего следует избегать любой ценой. Такая антипатия возникает (по крайней мере частично) оттого, что к инфраструктуре MVC обращаются в поисках альтернативы после множества не слишком успешных попыток разработки и сопровождения приложений с интеллектуальным пользовательским интерфейсом, которые попросту вышли из-под контроля.

Тем не менее, полный отказ от паттерна «Интеллектуальный пользовательский интерфейс» будет заблуждением. Далеко не все в нем настолько плохо, и с данным подходом связаны также положительные аспекты. Приложения с интеллектуальным пользовательским интерфейсом быстро и легко разрабатывать. Производители компонентов и инструментов проектирования приложили немало усилий, чтобы сделать процесс разработки приятным занятием, поэтому даже совершенно неопытный программист за считанные часы может получить профессионально выглядящий и достаточно функциональный результат.

Наибольшая слабость приложений с интеллектуальным пользовательским интерфейсом – низкое удобство сопровождения – не проявляется при мелких объемах разработки. Если необходимо создать несложный инструмент для небольшой аудитории, то приложение с интеллектуальным пользовательским интерфейсом может оказаться идеальным решением. Просто отсутствует дополнительная сложность, присущая разработке приложения MVC.

Также существует другая разновидность трёхуровневой архитектуры для клиент-серверных приложений: паттерн «молель-представление-презентатор» (model-view-presenter – MVP). Она разработан для того, чтобы облегчить согласование с поддерживающими состояние платформами графического пользовательского интерфейса, таким как Windows Forms или ASP .NET WebForms. Это достойная попытка извлечь лучшее из паттерна интеллектуального пользовательского интерфейса, избежав проблем, которые он обычно привносит.

В паттерне MVP презентатор имеет те же обязанности, что и контроллер MVC, но он также более непосредственно связан с представлением, сохраняющим информацию о состоянии, напрямую управляя значениями, которые отображаются в компонентах пользовательского интерфейса в соответствии с вводом и действиями пользователя. Существуют две реализации паттерна MVP:

* Реализация пассивного представления, в которой представление не содержит никакой логики. Такое представление служит контейнером для элементов управления пользовательского интерфейса, которым напрямую управляет презентатор.
* Реализация координирующего контроллера, в которой представление может отвечать за определенные элементы логики презентации, такие как привязка данных, и получать ссылку на источник данных от моделей предметной области.

Отличие между указанными двумя реализациями касается уровня интеллектуальности представления. В любом случае презентатор отделен от инфраструктуры графического пользовательского интерфейса, что делает логику презентатора более простой и подходящей для модульного тестирования.

Разработку проекта было решено реализовывать с помощью паттерна MVC.

Таким образом проект содержит следующую структуру:

Объект wwwroot, в котором находятся css-файлы, которые определяют стили для сайта (его цвета, отступы и декоративные изображения), и JavaScript-файлы, они не отвечают за логику сайта, это задача контроллера, JS файлы необходимы только лишь для отображения графиков по измеренным данным, и эта задача реализуется с помощью библиотеки ApexCharts. Данная библиотека представляет собой высокоуровневое решение для отображения графической статистики, это необходимо чтобы пользователь мог в интерактивном режиме ознакомиться с данными.

Объект модели, в котором находятся классы, предоставляющие интерфейс для работы контроллера. Поскольку в базе данных находится одна таблица, в модели необходимо реализовать всего одну сущность – измерение. Код выглядит следующим образом:

namespace ServerMVC.Models

{

public class Measurement

{

public DateTime? measure\_date { get; set; }

public int? measure\_hour { get; set;}

public int? measure\_min { get; set;}

public int? t\_pow { get; set;}

public int? far { get; set; }

public int? rh { get; set; }

public int? t { get; set; }

public int? winddir { get; set; }

public int? wind { get; set; }

public int? rain { get; set; }

}

}

Как можно понять из листинга, все поля класса идентичны с колонками базы данных. По сути, можно сказать, что экземпляры класса Measurement будут непосредственно отображаться на веб-сайте.

Объект контроллер. Контроллер реализует функции, которые трансформируют данные, для дальнейшей передачи в представление. В этом объекте будут формироваться с помощью циклов или лямбд данные метеостанции за запрошенный пользователем период.

Объект представление реализует конечное представление данных для пользователя. Физически в MVC ASP .NET объект представления является файлом с расширением cshtml. Из этого названия можно сделать вывод, что эта сущность является смесью C# кода и HTML разметки. В объекте представления между обычной HTML разметкой можно вставлять куски C# кода, которые выполняют интерактивную функцию: отображение данных из контроллера непосредственно в тегах, реализация форм ввода данных для отправки GET и POST запросов.

Также в проекте содержатся конфигурационные файлы, которые позволяют настроить поведение веб сервера в общем.