Three-layered architecture with Sitecore

Sitecore User Group Switzerland 2014

Kevin Brechbühl

/ /ug/ch

About me

- Kevin Brechbühl
- Senior Application Architect at Unic
- Sitecore experience since 2010

- Twitter: @aquasonic
- Blog: http://ctor.io

What I’m going to show you

- Why are we talking about this topic?
- What do we want to achieve?
- Three-layered architecture
- Available frameworks
- Demo

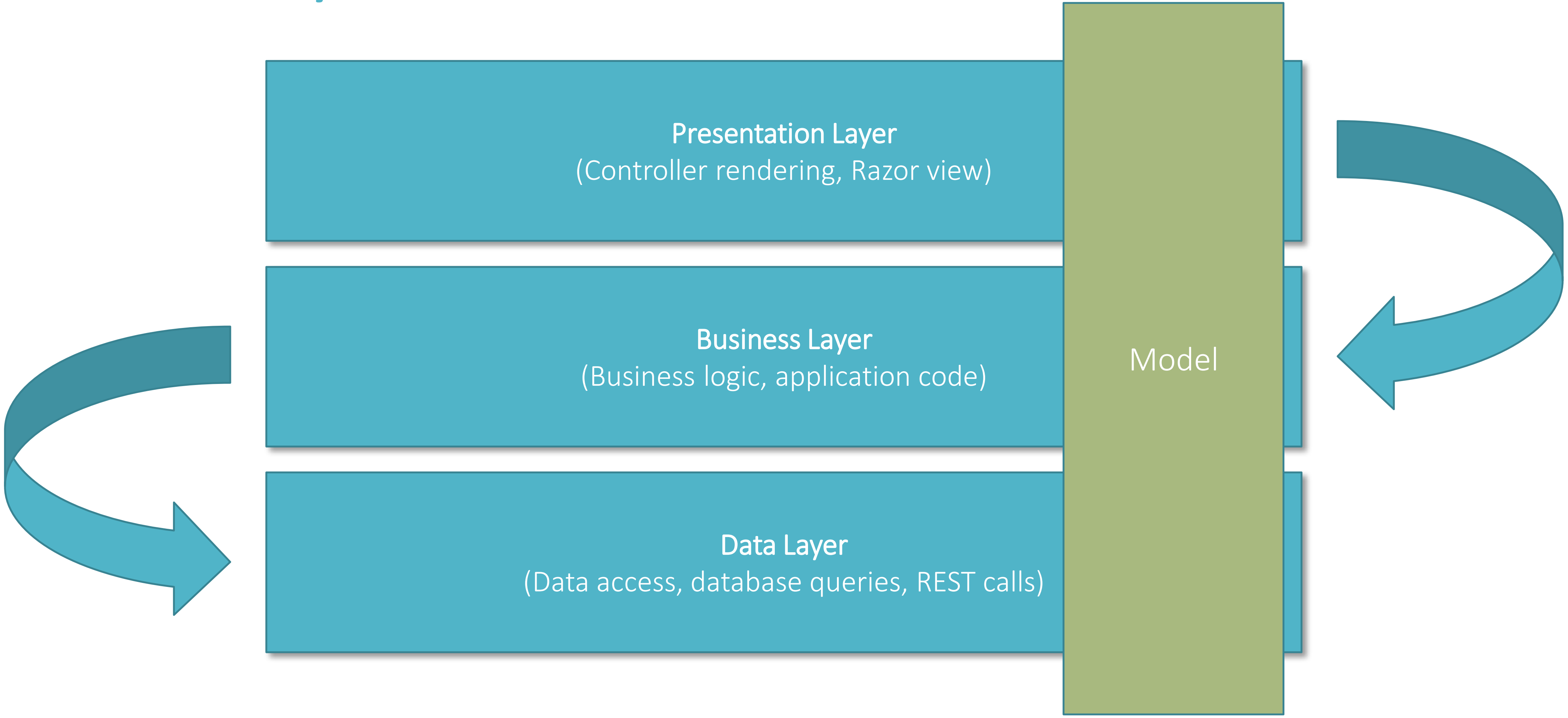
Why are we talking about this topic?

- Unstructured code (one class does everything)
- Code maintainability
- DRY = Don’t repeat yourself
- High developer skills needed
    - One application is implemented by one single developer
    - From presentation to data access
- Strong dependency on *HttpContext*, Sitecore and the databases
- Unit testing

What do we want to achieve?

- Separation of concerns
  - Code parts are responsible for a single concern
  - Clean, structured and maintainable code
  - Developer responsibilities (e.g. Frontend, database access, web services)
- Abstraction and decoupling
  - Abstract dependency to Sitecore
  - Mocking and unit testing
- Common solution: Three-layered architecture

Three-layered architecture


Presentation Layer
(Controller rendering, Razor view)
Business Layer
(Business logic, application code)
Data Layer
(Data access, database queries, REST calls)
Model

Architecture rules

- Each layer only has access to it’s underlying layers
- Never allow the architecture to call layers the other way around
- A layer never knows what the other layers do and what the data is used for

- Proposal
  - Classes from business layer gets the suffix *Service*
  - Classes from data layer gets the suffix *Repository*

Available frameworks

- Decouple presentation, data and application logic → Sitecore MVC
- Loose coupling and dependency injection → Ninject
- Sitecore abstraction → Glass Mapper
- Separation of concerns is given by the architecture

- Choose the frameworks you prefer

Sitecore MVC

- Actually not necessary for this architecture
- Is used for this session and demo

- Available since Sitecore 6.6
- Enabled by default since Sitecore 7.1

Ninject

- Very easy dependency injection framework
- Supports ASP.net MVC controller injection
- Fantastic logo and name ☺

- http://www.ninject.org/
- Available via NuGet

Glass Mapper

- ORM (object-relational mapping) for Sitecore
- Maps Sitecore items to strongly typed objects

- http://glass.lu/
- Available via NuGet

Demo

- Show list of news in a Sitecore MVC controller rendering
- Sort them descending by date
- Never use the *Sitecore.Context* to access data
- Go through the different layers to get and display the data
- Abstract different layers into different Visual Studio projects
- Simple unit tests

Recap

- It’s very easy to build a three-layered architecture
- Easy to understand and easy to maintain
- We can abstract a lot and have many advantages
- We can mock the data layer e.g. for unit tests

- It makes developer lifes easier and the code cleaner
- **Developers must understand the architecture and follow the rules**

Resources

- Blog post available at my blog: http://ctor.io/three-layered-architecture-with-sitecore
- Source is available on GitHub: https://github.com/aquasonic/SUGCH2014

Questions?