



MASARYKOVA UNIVERZITA

PV213 Enterprise Information Systems in Practice

02 – Architecture of the EIS in the standard environment

PV213 EIS in Practice: 02 – Architecture of the EIS in the
standard environment



MASARYKOVA UNIVERZITA

Tento projekt je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky.



INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

PV213 EIS in Practice: 02 – Architecture of the EIS in the standard environment

Tento projekt je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky.



INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Why we need SW architecture?

Why SW architecture is created?

- Increase product quality
 - Functional requirements
 - Non-functional requirements
- Understand product structure
- Speedup development time
- Minimize maintenance costs
- ...



Architect in history

The ideal architect should be a person of letters, a mathematician, familiar with historical studies, a diligent student of philosophy, acquainted with music, not ignorant of medicine, learned in the responses of jurisconsults, familiar with astronomy and astronomical calculations.

—Vitruvius, circa 25 BC

Vitruvius was a Roman architect and the author of the *De architectura*.

Pont du Gard, France



SW Architect - role in the team

Key responsibilities

- ❏ Creating and **documenting** architecture
 - ❏ High level
 - ❏ Design decisions (especially non functional requirements)
 - ❏ Guidelines
- ❏ **Communication** between different stakeholders
 - ❏ Requirements with customer / analyst / business people
 - ❏ Planning with project manager
 - ❏ SW architecture / design with developers, testers
- ❏ **Supervising** project from technical point of view

We can differentiate: business architect, enterprise architect, solutions architect, application architect

Non-functional requirements

Product qualities which are often overlooked but they are important part of the product.

- **Performance** - e.g. transactions per seconds
- **Scalability** - ability to “grow” with increasing load
- **Reliability** - ensures integrity and consistency
- **Availability** - e.g. in percentage (99,9%)
- **Security** - system and its data cannot be compromised
- **Maintainability** - ability to correct flaws without big impacts
- **Manageability** - monitor system health, change of configuration
- **Extensibility** - ability to add new functionality without big changes
- **Testability** - ability to test
- ...

Architecture types - Monolithic (1-tier)

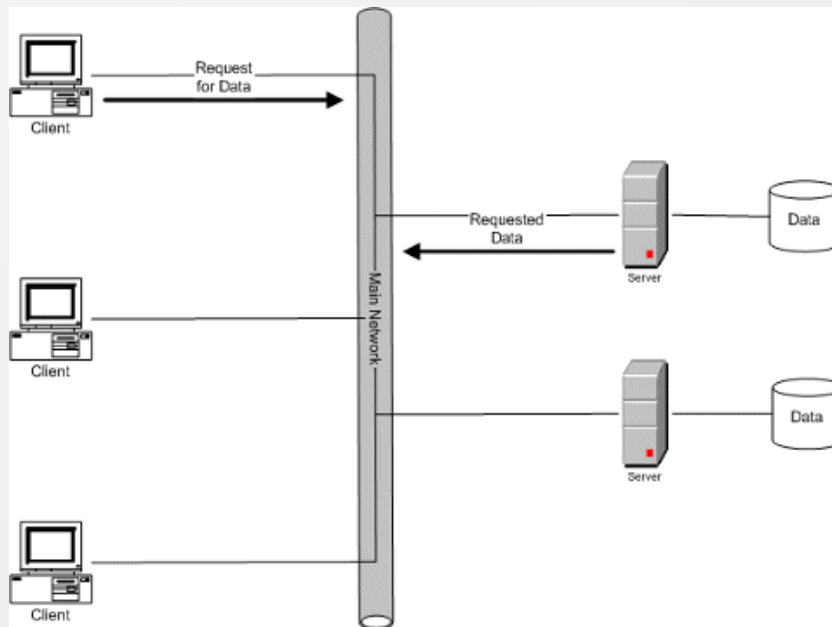
- ❏ First type in history (on mainframes)
- ❏ Everything in one monolithic application
 - ❏ User interface
 - ❏ Application (business) logic
 - ❏ Persistence logic (database)
- ❏ Simple
- ❏ Hard to fulfill some non-functional requirements
 - ❏ Non-scalable
 - ❏ Hard to extend
 - ❏ Hard to maintain
- ❏ Today valid only in very special cases - embedded SW with small resources (CPU power, memory, without network connectivity)

Architecture types - 2-tier I

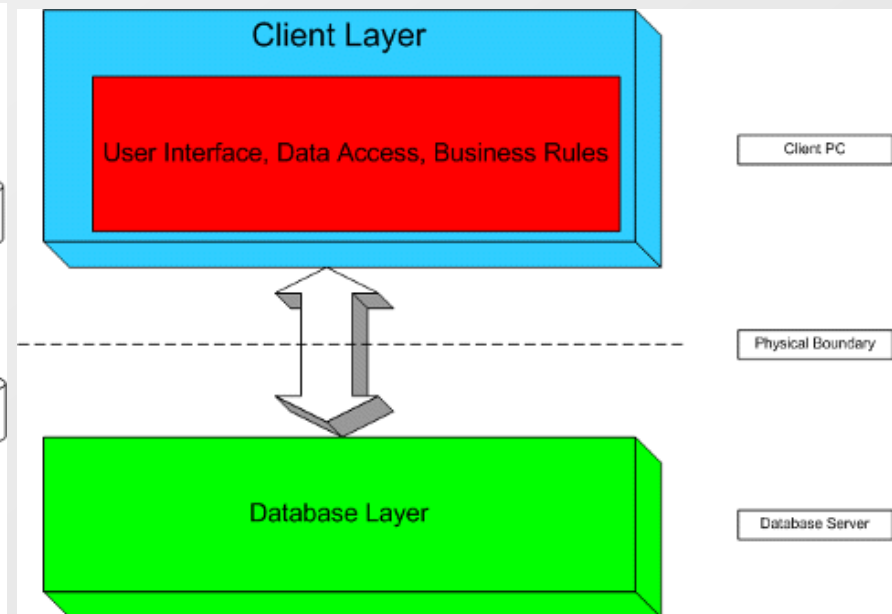
- ❏ Separation
 - ❏ First tier - User interface plus application logic
 - ❏ Second tier - Persistence (database)
- ❏ Also called client / server architecture (popular in 1980s)
- ❏ Allows to easily share data from different clients
- ❏ Still hard to fulfill some non-functional requirements
- ❏ Used mostly in desktop applications
 - ❏ Hard to change business logic (requires redeployment)
- ❏ Small modification for exceptional web applications
 - ❏ First tier - User interface
 - ❏ Second tier - Application logic and persistence
- ❏ Not recommended for new development

Architecture types - 2-tier II

Physical model



Logical model

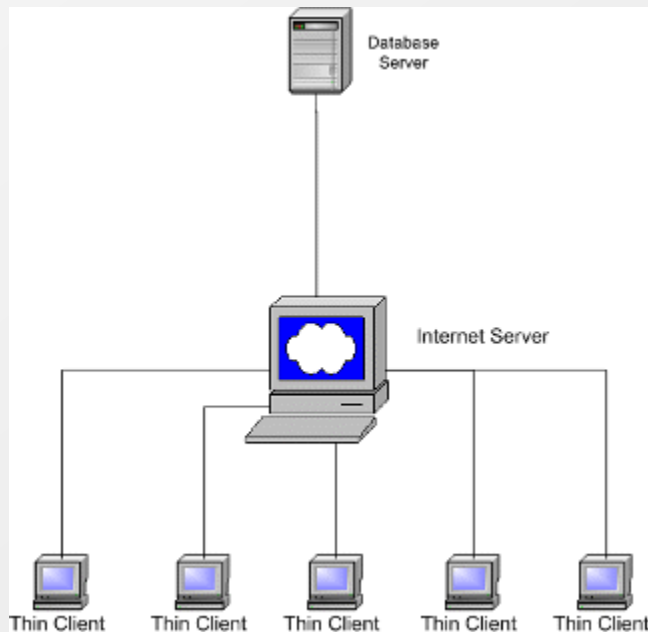


Architecture types - 3-tier (n-tier) I

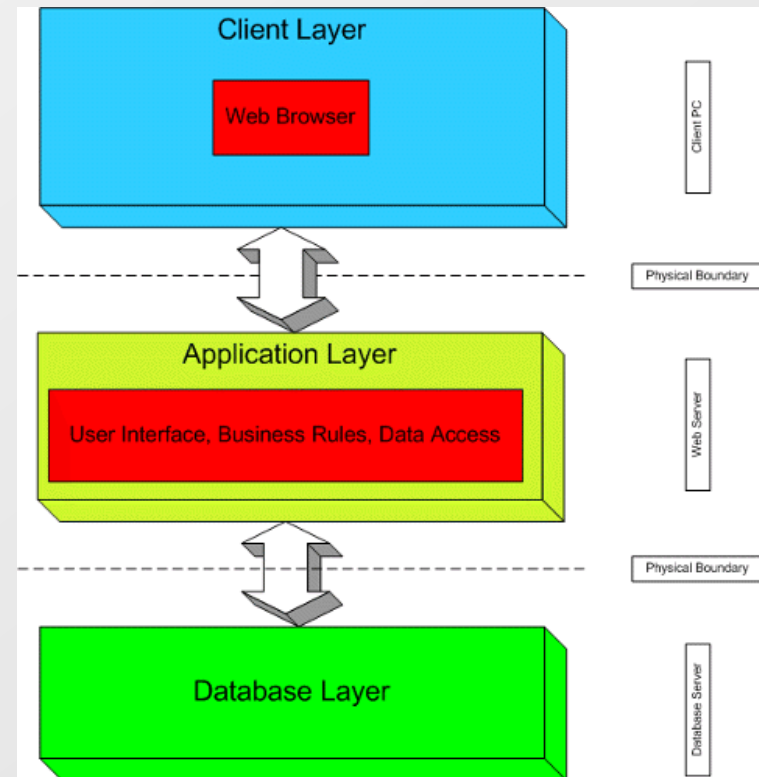
- Separation
 - First tier - User interface
 - Second tier - Application logic
 - Third tier - Persistence
- Separation allows to increase quality of the system
 - E.g. change of the business logic doesn't influence presentation tier (user interface)
- Valid for most of web applications
- Standard in designing systems today

Architecture types - 3-tier (n-tier) II

Physical model



Logical model

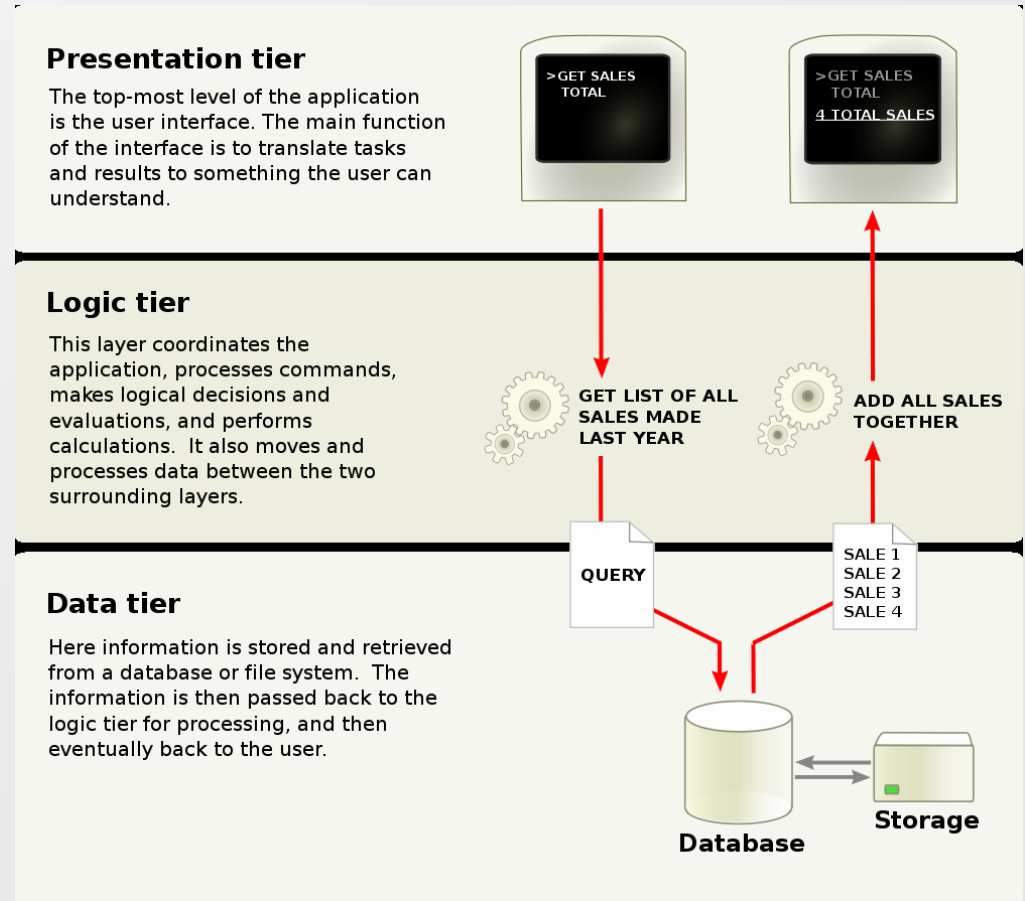


Architecture types - 3-tier (n-tier) III

Tiers represents machine boundaries

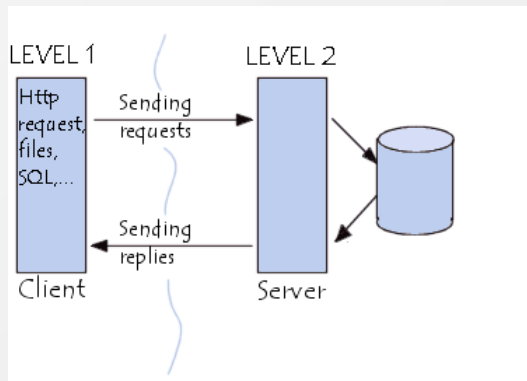
Much easier to scale

Changes are separated

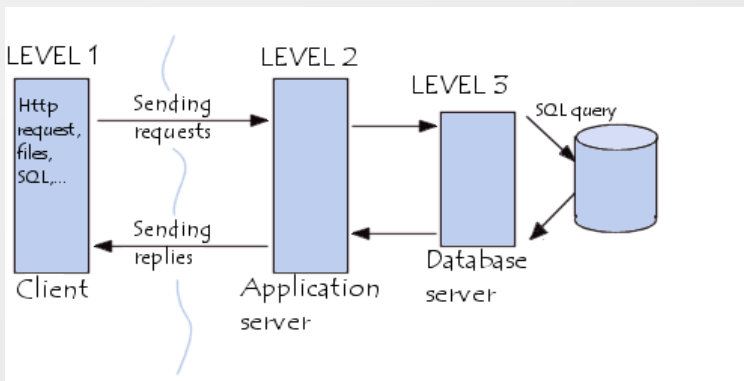


Architecture types - Comparison of 2,3 and n-tier

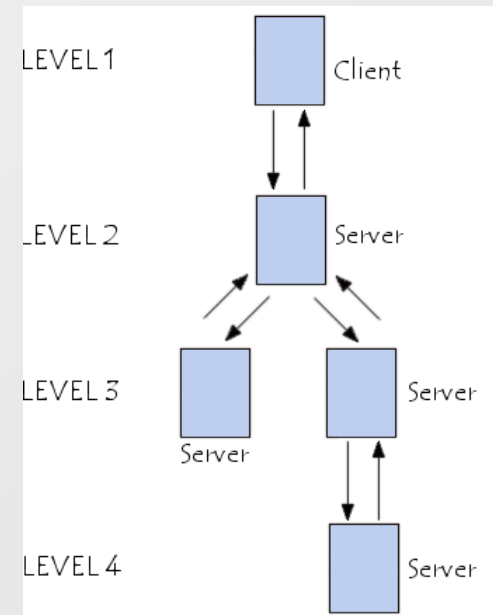
2-tier



3-tier

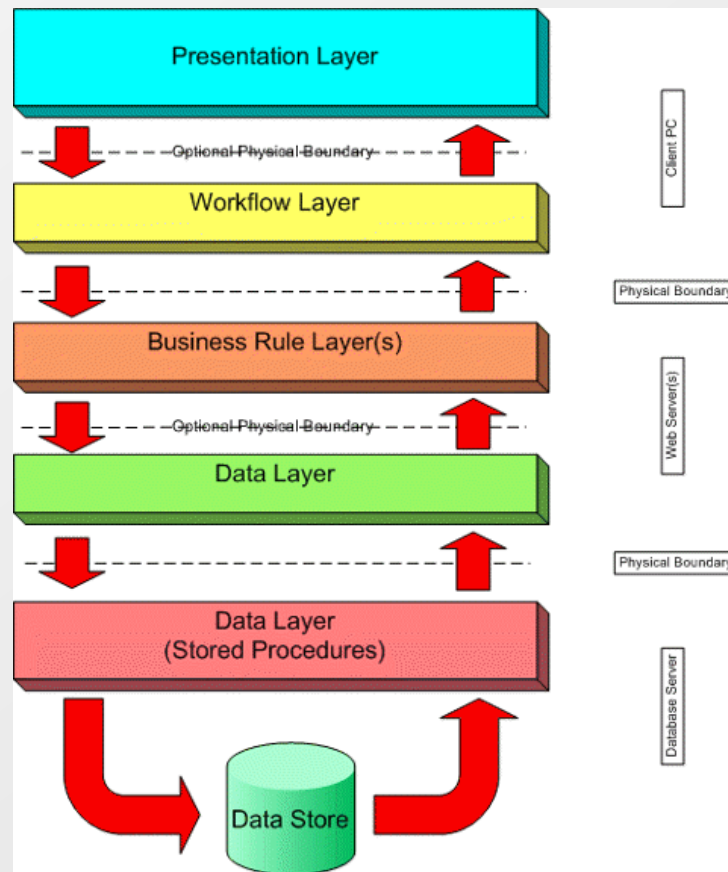


n-tier



Architecture types - Possible structuring in n-tier

Logical model



Thick and thin client

- ❏ **Thick client** provides rich functionality without server
 - ❏ Requires some installation on the client's device
 - ❏ Can run independently without servers
 - ❏ Can use all HW device provides (rich and fast GUI, external HW)
 - ❏ Migration to newer version (update) is harder

- ❏ **Thin client** delegates functionality to other devices (servers)
 - ❏ Don't require installation on the client's device
 - ❏ Cannot run independently without servers
 - ❏ Access to HW is limited
 - ❏ Migration to newer version (update) is easier
 - ❏ Requires some "specialized OS" (web browser, "player", etc.)
 - ❏ Servers are the bottleneck

In reality thick and thin clients overlap (browser plugins, zero installations, ...).

Example - Reservation system (refresh)

Reservation system allows to

- Make reservations for pool equipments
- Cancel reservations
- Show calendar with reservations
- Import definition of equipments from external inventory system
- Send usage statistics to management reporting system

Example - Reservation system - Architectural thoughts I

You have to consider which architecture solves the best your functional and non-functional requirements with additional restrictions (constraints) to

- ❏ Given budget
- ❏ Expected timeframe
- ❏ Skills in your team
- ❏ Expected live time of the product
- ❏ Non-technical constraints (team motivation, overall atmosphere, ...)
- ❏ ...

You can try **negotiate** some (or all) constraints with project (or product) manager (business people, customer, ...).

Live is not ideal but good **communication** with other people can simplify it!

Example - Reservation system - Architectural thoughts II

Possible questions we have to ask

- ❏ How users will use the Reservation system?
 - ❏ Is program installation on user's device required? Thick / thin client? What are prerequisites?
 - ❏ Which operating systems are required to support?
 - ❏ Which types of user's devices are required (e.g. mobile)?
- ❏ How data will be shared between users? Is sharing of data possible?
- ❏ How product will interact with external systems (inventory and reporting system)?
- ❏ How system will be maintained?
- ❏ How system will scale when amount of users grow?
- ❏ How possible new requirements will influence the system?
- ❏ ...

Example - Reservation system - 1-tier architecture?

- ❏ Implies thick client
 - ❏ Installation of client required
 - ❏ Update of client can be complicated (distributed redeployment)
- ❏ Sharing of data complicated
 - ❏ Each user has its own copy of database
 - ❏ Needed to solve synchronization of databases (when number of users grow more data has to be transferred between clients)
- ❏ Some functional requirements are hard to implement - e.g. export to reporting system
 - ❏ Which client will do it? What if he is ill?
 - ❏ Another special application for export? How and who will start it?
- ❏ Result: Inadequate architecture for our purposes

Example - Reservation system - 2-tier architecture I?

- ❏ Desktop application (thick client)
 - ❏ User interface and application logic on the client
 - ❏ Persistence on the database server (data can be easily shared between clients)
 - ❏ Installation of the client required
 - ❏ Update of client can be complicated (distributed redeployment)
 - ❏ Export of data still problematic

- ❏ Result: Inadequate architecture for our purposes

Example - Reservation system - 2-tier architecture II?

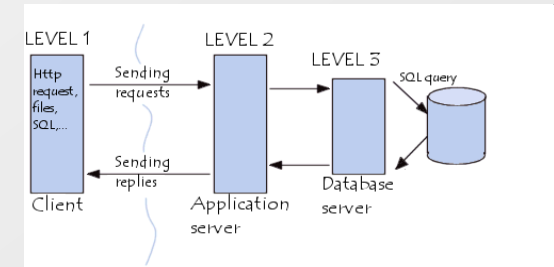
- ❏ Web application (thin client)
 - ❏ User interface on the client (web browser)
 - ❏ No need to do installation
 - ❏ Change in the application means redeployment only on the server
 - ❏ Application logic and persistence on the database server
 - ❏ Application logic e.g. in stored procedures - logic is tightly bind to the database
 - ❏ Implies database which support such configuration (vendor lock-in)
 - ❏ Export of data
 - ❏ Automatically by scheduled stored procedures (scheduled jobs)
 - ❏ Special application on the database server
 - ❏ Manually in the web GUI by special user

- ❏ Result: Better than 2-tier desktop application but still some flaws

Example - Reservation system - 3-tier architecture I?

- ❑ Desktop application (thick client)
 - ❑ User interface on the client
 - ❑ Application logic on the application server
 - ❑ Persistence on the database server
 - ❑ Easier to switch to different database or use simpler database
 - ❑ Installation of the client still required
 - ❑ Performance of user interface can be slower than in 2-tier (application logic executed on the application server - network boundaries)
 - ❑ Better scalable than in 2-tier (application server can also cache common data shared by multiple clients without need to load them from DB)
 - ❑ Still need to solve update of client (user interface)
 - ❑ Export of data solved on the application server
 - ❑ Used in cases when you need advanced and fast GUI

- ❑ Result: 3-tier desktop application doesn't solve deployment issues



Example - Reservation system - 3-tier architecture II?

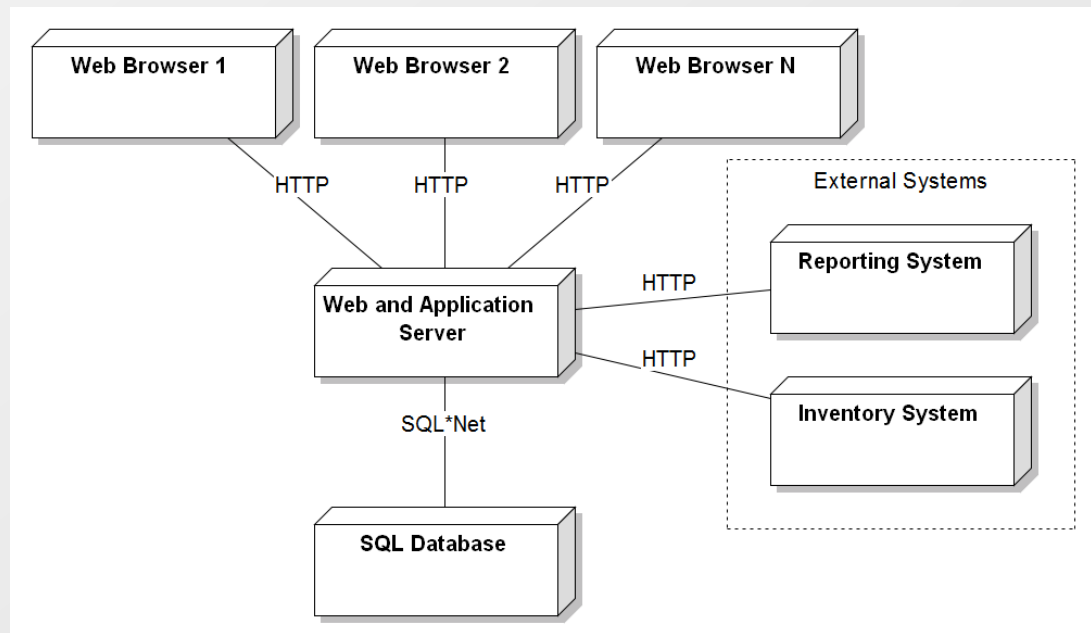
- ❏ Web application (thin client)
 - ❏ User interface on the client (web browser)
 - ❏ No need to do installation
 - ❏ Change in the application means redeployment only on the application (web) server
 - ❏ Application logic on the application (web) server
 - ❏ Persistence on the database server
 - ❏ Export of data
 - ❏ Special application (or just component) on the application (web) server
 - ❏ Manually in the web GUI by special user
 - ❏ Scales quite well
 - ❏ If weakest point is application server add additional server
 - ❏ If weakest point is database use clustered database (there are some limits)
 - ❏ Standard how applications are build today (a lot of “support” on the internet)

- ❏ Result: 3-tier web application is the best for our needs

Example - Reservation system - Deployment diagram I

Possible deployment for intranet

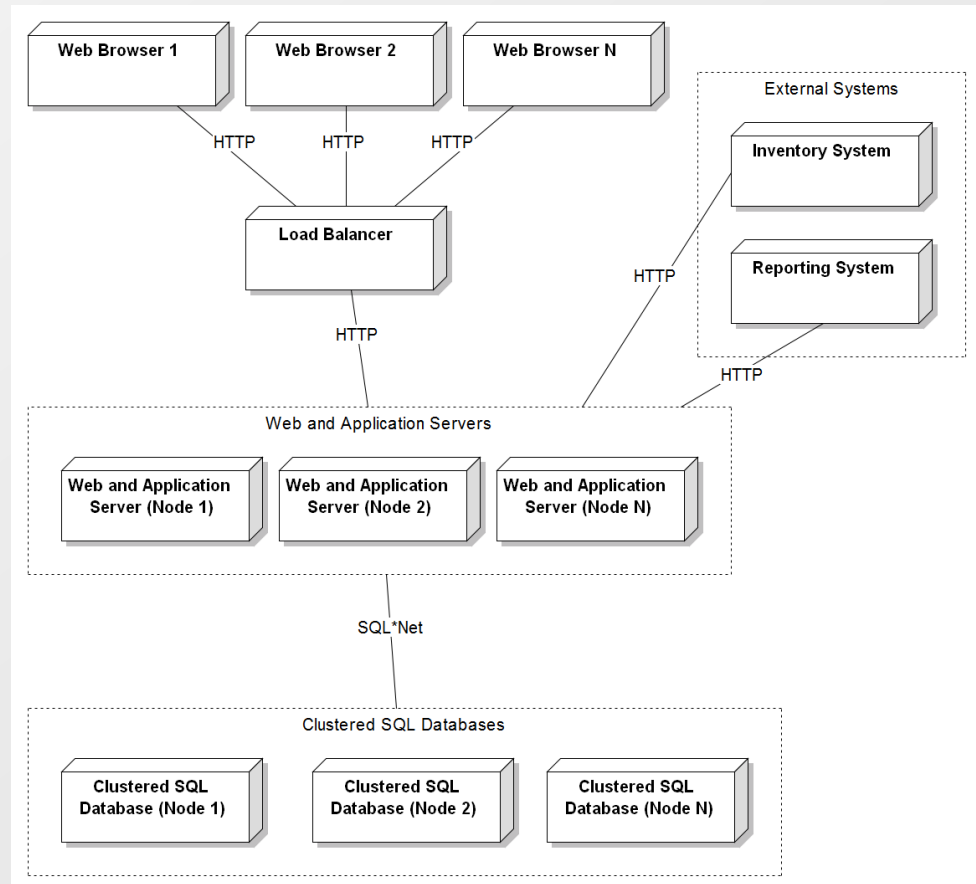
- ❑ User's need just standard web browser (no need for installations)
- ❑ For communication between clients and server just HTTP is used (we are in "secure" area)
- ❑ Web and Application server contains main part of the solution
- ❑ New version of the application means just redeployment on Web and Application (+SQL) server
- ❑ Communication with external systems is via HTTP (HTTPs)
- ❑ For storing data standard SQL database is used



Example - Reservation system - Deployment diagram II

Possible deployment for big amount of clients

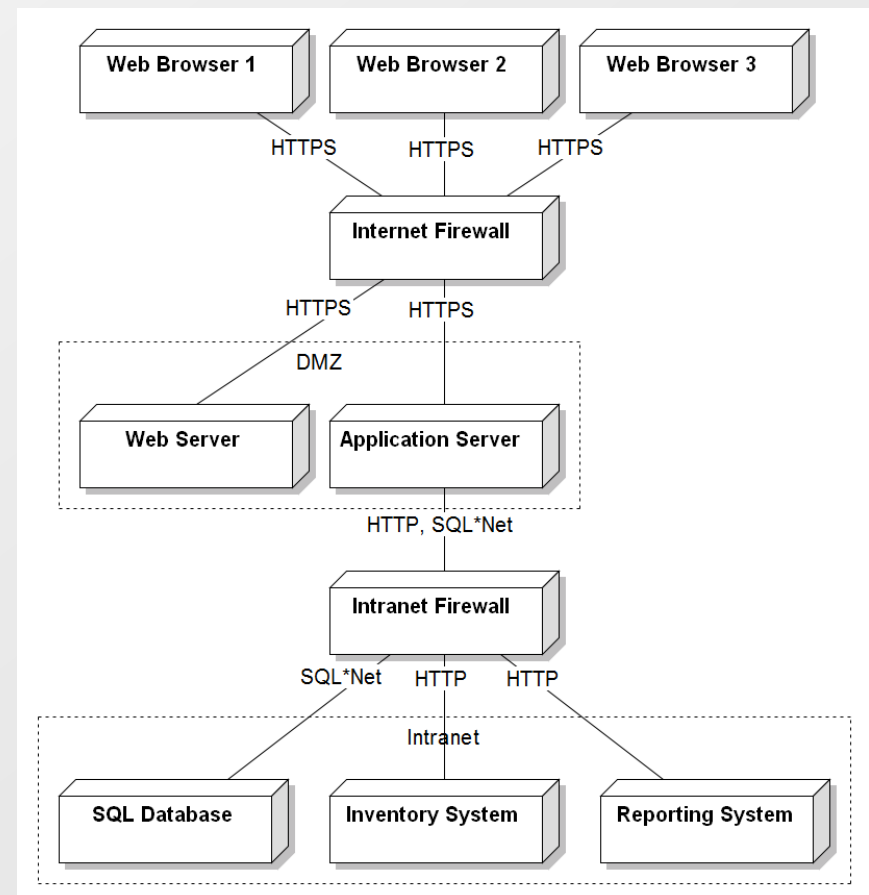
- ❏ Load balancer distributes requests to different nodes
- ❏ There are multiple web and application server nodes
- ❏ SQL database is configured to be clustered database on several nodes



Example - Reservation system - Deployment diagram for internet

Possible deployment for internet environment

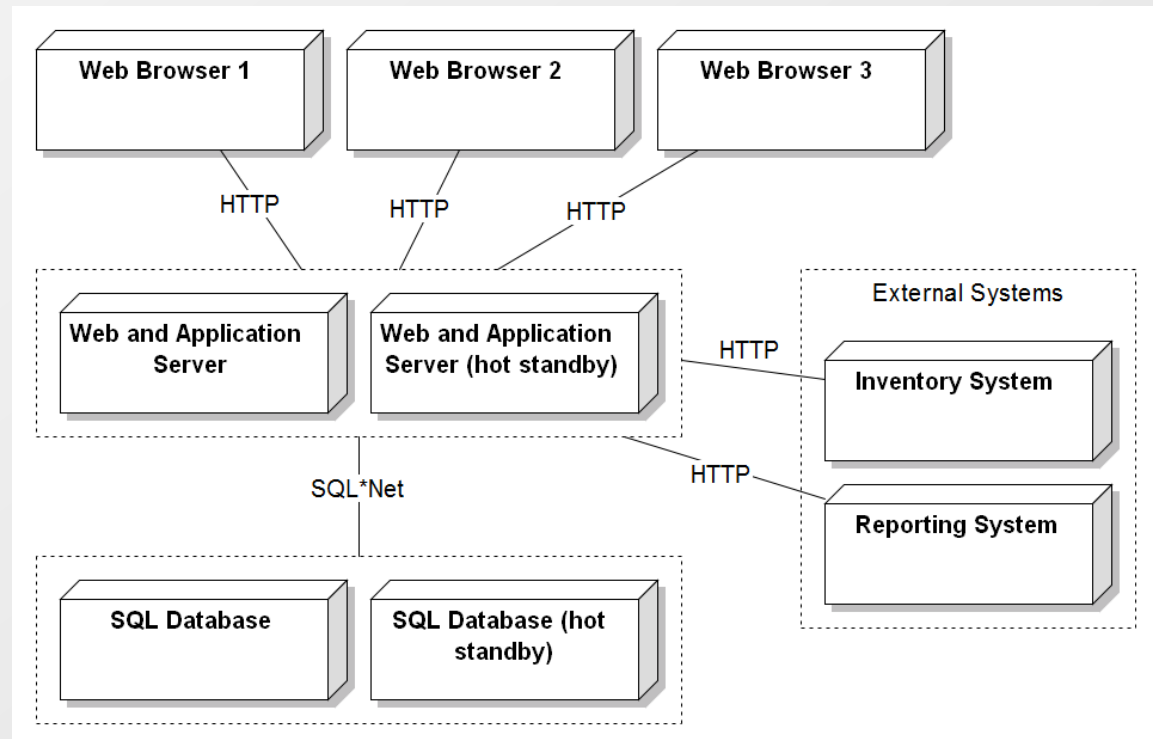
- ❏ Web browsers use HTTPS instead of HTTP
- ❏ Internet firewall restricts protocols just to HTTPS
- ❏ To increase performance static content (pictures, videos, static HTML pages, ...) are handled on special web server node
- ❏ Intranet firewall restricts protocols to HTTP and SQL*Net and requests only from application server
- ❏ Communication in the intranet is just HTTP



Example - Reservation system - Deployment with redundancy

Possible deployment with redundancy

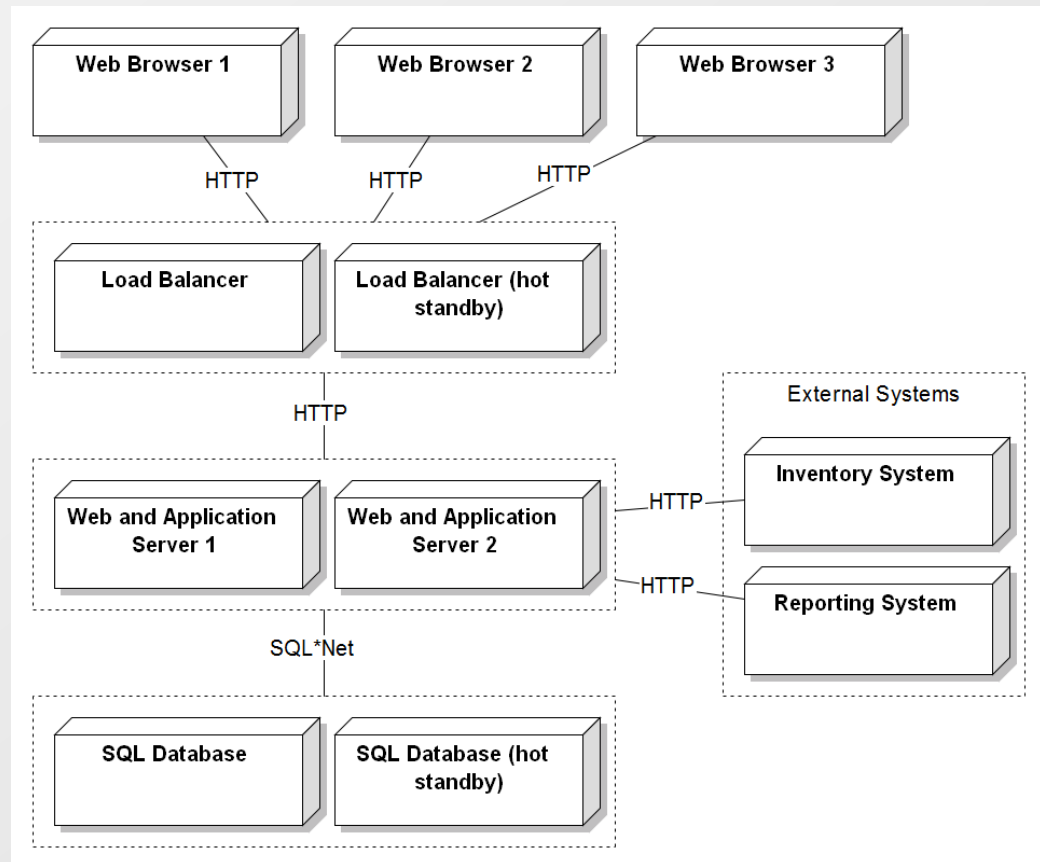
- ❏ Each critical node is backed up by the same node
- ❏ In case of failure (e.g. HW) second (standby) node will handle request
- ❏ Standby nodes add additional costs but do nothing in normal behavior
- ❏ Weak points are external systems which we cannot influence



Example - Reservation system - Deployment with load balancing redundancy

Possible deployment with load balancing redundancy

- ❏ Load balancer distributes requests either to Web and Application server 1 or 2
- ❏ Load balancer is backed up
- ❏ Web and Application Server must be stateless or load balancer must take into account user's session
- ❏ In two node deployment for Web and Application server one node must be able to handle all requests (in some decreased performance)
- ❏ Standard database cannot be easily load balanced. Often they are weak points in scalability



Recommendations and what you have to think about I

- ❏ For majority of new EIS applications use 3-tier (n-tier) web based architecture
- ❏ Backup critical systems (calculate losses in case of not functional system)
- ❏ Know expected number of user's
 - ❏ In the beginning it can be hard to guess especially for applications which will become popular
- ❏ Be prepared for increasing number of users
 - ❏ Cloud computing can help you
- ❏ Do load testing before system goes to productive
 - ❏ Only a few projects do this
- ❏ Be prepared to use your system from internet
 - ❏ You have to take care more about the security on the internet
- ❏ Be prepared for different user devices (standard PC, tablet, mobile)
 - ❏ There is increasing importance of mobile devices

Recommendations and what you have to think about II

- ❏ Prepare the architecture before any development starts
 - ❏ When real development starts it is too late to create the architecture
- ❏ Keep in mind all important non-functional requirements
- ❏ Create documentation and guidelines for developers and share the knowledge
 - ❏ Better is to make an architecture meeting than just telling to developers “here are documents and read them”
 - ❏ Do it continuously
- ❏ Setup tools which help you finding abnormalities in the architecture from code
- ❏ Do the code reviews to find “code smells”
- ❏ Adapt architecture through the lifetime to fit into new requirements
- ❏ Role of architect is not valid only in the beginning but through the whole lifetime of the project
 - ❏ A lot of projects starts with good architecture but degrade through the lifetime as requirements change

Different architectures examples I

Which architecture type (number of tiers, type of the client) do you recommend for following usages?

- ❏ Simple game on mobile device without connectivity to the network
 - ❏ 1-tier, thick client
- ❏ Game on mobile device with possibility to use multiplayer functionality and sharing game scores
 - ❏ 3-tier, thick client
- ❏ Application for managing business trips which must run on mobile devices and standard PCs
 - ❏ 3-tier, thin client (web application)
- ❏ Mobile application for collecting data from house construction in the field and reporting them to the headquarter (must work offline)
 - ❏ 3-tier, thick client

Different architectures examples II

- ❏ Management reporting application with advanced graphical interactive reports which must run on “any” device
 - ❏ 3-tier, thin client (web application with some technology which allows displaying interactive reports e.g. Flash or HTML5)
- ❏ Industry application for reporting weather conditions
 - ❏ 3-tier, thick client (embedded SW)
- ❏ News application aggregating data from different sources
 - ❏ 3-tier, thin client (web application)
- ❏ Project management application
 - ❏ 3-tier, thin client (web application)
- ❏ Computational intensive application (simulation of chemical processes)
 - ❏ 3-tier, thick client (special desktop application), distributed computing

Děkuji za pozornost.

Tento projekt je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky.



INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ