Client Side & UI

Internal apps level
- KnockoutJS

Complex internal to large scale
- Durandal

Enterprise level
- AngularJS / EmberJS

UI - easy & free
- Twitter Bootstrap / Foundation Framework

UI - fancy but expensive
- Kendo UI

Data Access

Dev-Minimum / Perf-Slow
- Entity Framework – Easy Commonly Known

Dev-Moderate / Perf-Average
- Nhibernate – Most feature rich ORM now (2014)

Dev-Consuming / Perf-Fast
- ADO.NET with Stored Procedures
- MSSQL 2014 (With in memory tables)
- Memcached for stale data
- Redist for stale data and/or non crucial data

Costs & Economics

Database choice
- MySql (Cheap relational & high performance)
- MsSql (Expensive relational – High performance and integrated security)
- NoSql solutions vary to much to list here but there are free as well as paid solutions to be investigated for their specific advantages over relational DBMS

3Rd party libraries
- StackOferflow's – Service Stack – Expensive but blazing fast ORM, Media Formatters, IOC There is also older open source version of Service Stack with similar value but less up to date
- Expensive UI framework like Kendo vs Free Twitter Bootstrap

Best Practices & Conventions

FxCop for Visual studio
- Integrates with various continuous integration systems build rules
- Ensures following specified coding conventions

Scalability

Tens of hundreds (Requests at peak)
- No technology choices

Tens of thousands
- Horizontal scalability
- Distributed database in case of large user data sets – sharding & archiving should be considered

Hundred thousand+
- Private / Public Cloud
- Hadoop for data processing
- Geo-distributed Content Delivery Networks
- Multiple geographically separated Data Centers for app-servers (Security / Response Time)

Caching

Dev-Minimum / Perf-Fast
- Full view caching with .Net Output Cache for websites with slowly changing content

Dev-Average / Perf-Fast
- Simple data caching with Memcached (Simple in memory key value – Also Distributed Caching) good for service layer calls

Dev-Consuming / Perf-Fast
- Complex structured data caching With Redis (NoSql in Memory DB) good for complex service layer caching with dependencies and other unusuall requirements

Dev-Consuming / Perf-Fast
- Client sIde entity caching with BreezeJs – Additionally provides change tracking & rich server resource querying good for offline mode and easing server load

Solution Architecture

Internal apps level
- 2 Tiers usually Presentation+DAL (With MVC you could use WebAPI as your business layer: MVC provides views > API handles business rules)

Enterprise level
- 3 Tiers is a save bet, do not overcomplicate BLL (Validators, rules etc. All cango into one layer no need to split)

Enterprise (high profile / distributed team)
- Utilize battle tested „Onion Architecture" which puts your BLL in the center and allow for better testability and mocking in case of phase-by-phase development

Security

Internal apps level
- WebForms authentication, build in .Net credentials store

Enterprise level
- SSL on service calls
- SHA512 for password encryption
- BCrypt hashing for key stretching

Enterprise (high profile / multiple solutions)
- Additionally to above enterprise solutions, consider centralizing your security in one authentication application / service;
- Consider single sign on via .net federated authentication mechanizms

Localization & Accessability

Internal apps level
- Resource based (No need for db - usually)

Enterprise level
- Use translation keys with custom helpers throuout your application Eg: @Translation(Dashboard.Title)

Enterprise (fully editable/customizable)
- Use DbResource Provider - build in .net resource provider can be extended to use database as source giving flexibility
- instant refresh and locale integration while not being to hard to implement.

High internalization notes:
- Use UTC Time in db to avoid sync issues as well as .net specific classes helping with currency etc.

Testing

Internal apps level
- Most likely not needed

Internal-Complex
- Test Business Layer and if budget allows other layers including UI (Consider Automated UI Testing)

Enterprise
- Unit test bssines logic; Automate UI testing; Create database logic tests; Write manual regresion test scripts;

Note: Test Coverage (If set in your continuous integration system) should not be to high to avoid Get/Set testing by developers, test logic which can actually fail if missused or not cerfully modified.

Brain Power Focus

High performance
- Employ your star-coders in areas of: DAL / Hardware-Infrastructure

Usability
- Employ your star-coders in areas of: UI / BLL

Usability-HighDemand (Think facebook scale)
- Employ your star-coders in areas of: UI / BLL / DAL / Hardware-Infrastructure; Consider Having separate teams on each layer leaded by senior developers cooperaing with architect/s