U-Mail System Design Specification

Joseph Woo, Chris Hacking, Alex Benson, Elliott Conant, Alex Meng, Michael Ratanapintha

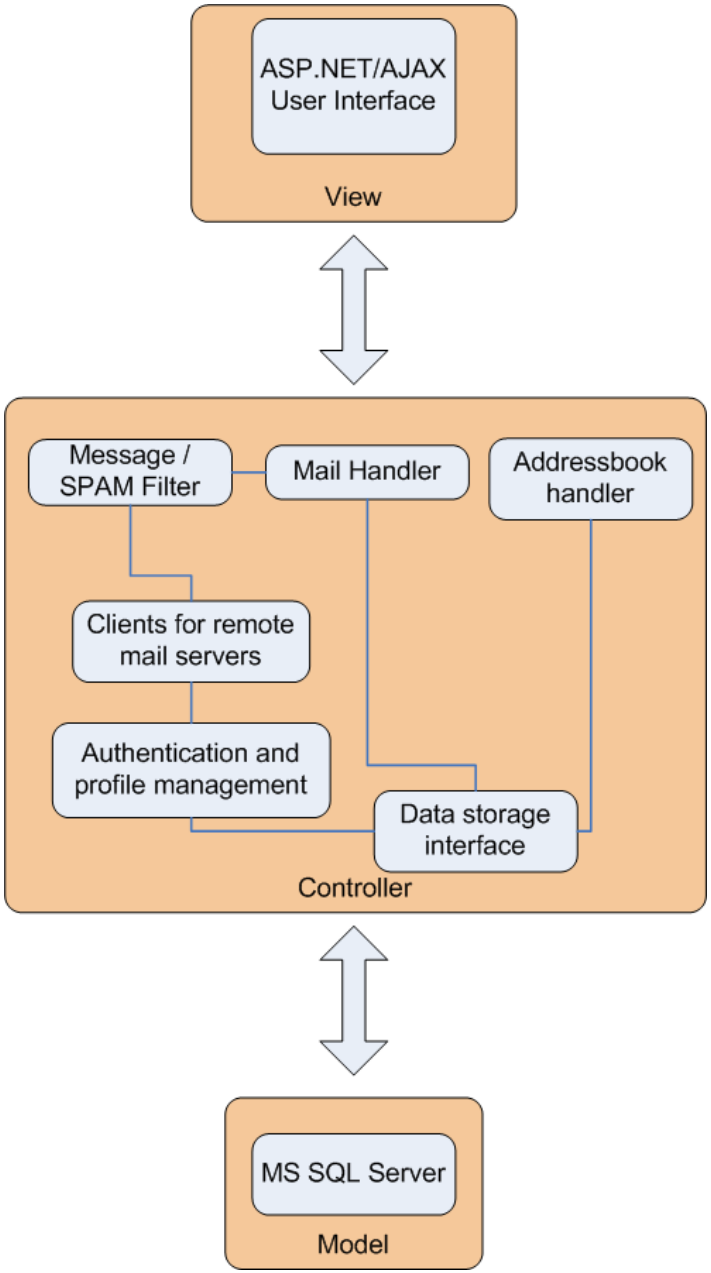
April 28, 2008

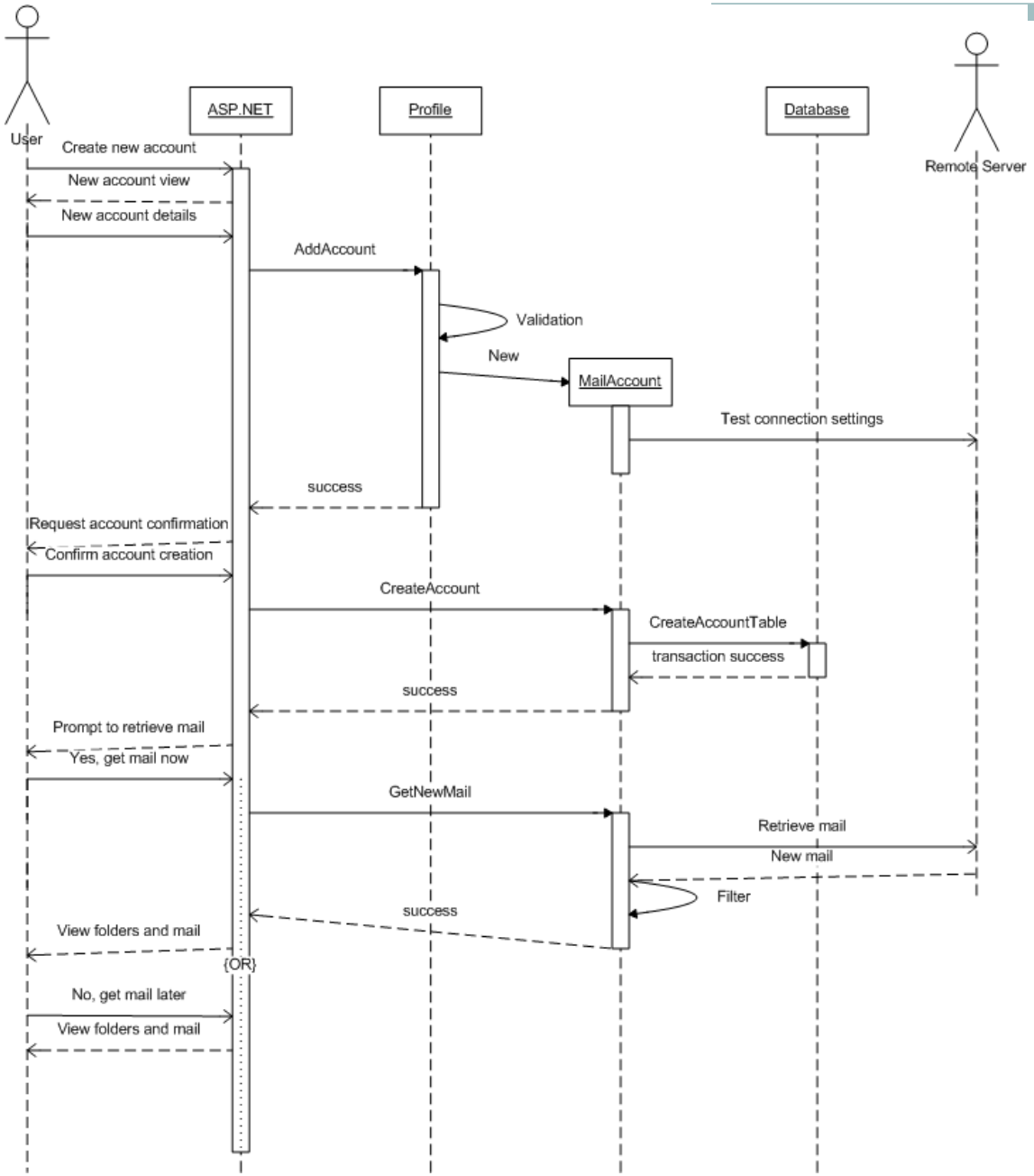
Outline

- ***System architecture***
- Development plan
- Test and documentation plan

U-Mail, the online e-mail client

- Objective: to create a web e-mail client that allows universal access to all of a user's accounts, as conveniently and securely as a desktop mail client.
- Customers: Users with multiple e-mail accounts
- Scope: Access to POP/IMAP accounts, sending e-mail, address book, security, Spam filtering, AJAX UI, and storing encrypted messages.


ASP.NET/AJAX
User Interface
View
Message /
SPAM Filter
Mail Handler
Addressbook
handler
Clients for remote
mail servers
Authentication and
profile management
Data storage
interface
Controller
MS SQL Server
Model


User
ASP.NET
Profile
Database
Remote Server
Create new account
New account view
New account details
AddAccount
Validation
New
MailAccount
Test connection settings
success
Request account confirmation
Confirm account creation
CreateAccount
CreateAccountTable
transaction success
success
Prompt to retrieve mail
Yes, get mail now
GetNewMail
Retrieve mail
New mail
Filter
success
View folders and mail
{OR}
No, get mail later
View folders and mail

Outline

- System architecture
- ***Development plan***
- Test and documentation plan

How to Structure The Team?

- We would like everyone to see all of the software lifecycle, system design
- But if everyone does everything, then we have chaos, slipping deadlines, unhappiness…
- So, we let a few people see “the big picture”
- The rest get assigned a single subsystem to work on

Our Team Structure

- "Big picture" jobs
  - 1 Architect to design at high level
  - 1 Project Manager to keep work on track
- Everyone else is a "developer/tester"
  - Performs all other job functions
  - But each person is responsible for a separate functional area

Roles of Architect (Chris Hacking)

- Designs high-level system architecture
  - How many subsystems, and what do they do?
- Makes subsystems talk to each other
  - Subsystem interfaces
  - Custom low-level APIs
  - Integration tests (supervision only)
- Gives final word on design decisions

Roles of Project Manager (Joseph Woo)

- Manages work schedule: deadlines, internal meetings, etc.
- Submits deliverables to customer
- Talks to customer on team's behalf
- Keeps team records: meeting notes, design documents, customer communications

Roles of Developer/Testers (Everyone else)

- Implements subsystems based on architect's design documentation
- Identifies and fixes bugs
- Gives feedback on design to architect
- Writes unit tests
- Writes test applications

Schedule and Features of Releases

- April 30: Zero-feature release
  - Back end (DB schema, MVC) working, can login
- May 12: Beta 1 release
  - Create accounts, view mailbox, send, receive, and compose mail, use address book
- June 6: Final release
  - Compose with rich text and attachments, import contacts, use public people directories (LDAP)

Is the schedule good enough?

- Tasks for ZFR are specified in high detail.
- But for future releases, we know what to deliver, but not what tasks to do to deliver it.
- We understand your concern about this – we believe it is our most likely risk.

Possible risks (1)

- We can't deliver promised features
  - Very likely to occur
  - As our schedule and architecture evolve, we will see what we can't deliver
  - Then, we prioritize – let some features be late, so others can stay on time
- We're lazy assholes
  - Leads to above
  - Schedule, good management by PM should fix this

Possible risks (2)

- Our 3rd party libraries aren't good enough
  - We have several choices, so can easily switch
- Our database doesn't scale
  - Can limit storage if needed
- You realize you (do not) *really* want X
  - Must make sure our architecture can adapt
- We don't know C# or .NET
  - Online tutorials, talk to our team's experts

Outline

- System architecture
- Development plan
- ***Test and documentation plan***

Our test strategy

- We use 3 types of tests
  - Unit tests of individual classes/modules
  - System/integration tests to ensure everything talks to each other
  - Usability tests so we know if YOU can use it
- We think this is enough; it covers all the building blocks of your user experience
  - But we are willing to add more or different tests if needed

Unit Tests

- Test an individual class for correct operation, *independently* of the rest of the system
- The developer of a class writes its unit tests
- But we will use common standards for unit tests
  - Examples: Test at least 3 border cases, use at least 5 data sets on each method
- Will be run with the nightly build

System/Integration Tests

- Test interaction between classes and methods to ensure interoperation of system components
- Architect will supervise development of integration tests, and may write some himself
- Ideally, will be run with nightly build
  - But may take too long, especially as system nears completion

Usability Tests

- Test the transparency of system functions when presented to an end user
- Will be developed around use cases
  - Use cases taken from SRS or written specially
  - We give tester the goal of the use case
  - Observe tester using system, note actions, responses not covered in use case
- Will be done weekly
  - Ideally, do them with customers in customer meetings

Bug Correction

- “No broken windows”: everyone is responsible for the entire system
  - Though some are more responsible than others
- Bugs will initially be assigned to the writer of the affected class or module
- BUT, if no action is taken within ~3 days, reporter or others will take responsibility

Structure of documentation

- User's guide
  - Accessible through online help
  - Context-sensitive (clicking Help in the composer shouldn't send me to the configuration help)
- Administrator's guide
  - Because we'll forget 10+ years from now
  - Because others have to learn fast

Who writes the documentation?

- Architect outlines both user, admin guides
  - He designs the system at a high level, so he should describe it to the user at a high level
- But each developer writes the part that concerns his own subsystem
  - Architect will read, revise as needed

Conclusion

- We know what we are doing
  - Our feature set is complete (until you change it)
- We know how to do it
  - Our architecture is becoming clear, and remains adaptable
  - Our team roles and schedule are well-defined
  - We test from the tiny to the humongous
- We know we may have problems
  - We know about and can mitigate our top risks