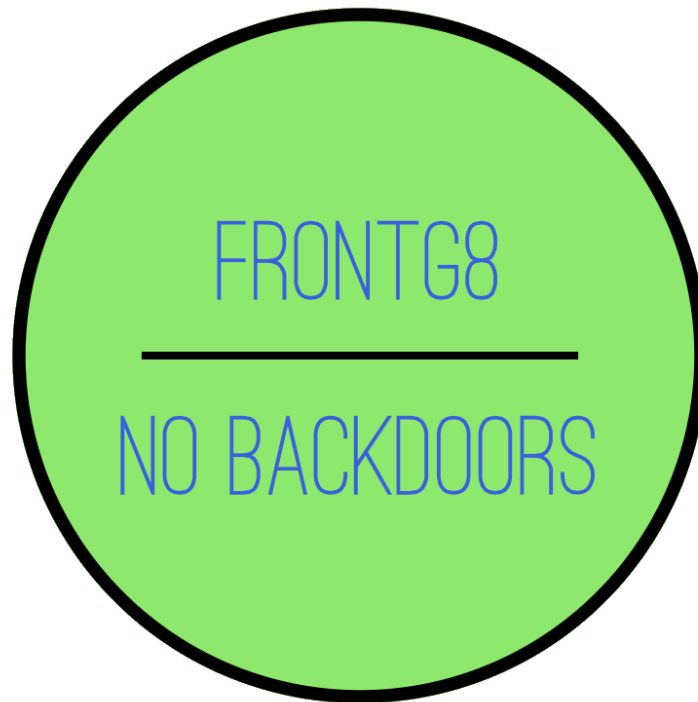




Projekt: frontg8

Projektplan

Version: 1.3

Felix Morgner - fmorgner@hsr.ch
Tobias Stauber - tstauber@hsr.ch
Ueli Bosshard - ubosshar@hsr.ch



Änderungsgeschichte

Datum	Version	Änderung	Autor
27.02.15	1.0		Ueli Bosshard
03.03.15	1.1	Diverse Korrekturen nach Feedback	Ueli Bosshard
02.04.15	1.2	Zeitplanung aktualisiert	Ueli Bosshard
25.05.15	1.3	Kleine Korrekturen	Ueli Bosshard

Inhalt

Änderungsgeschichte	2
Inhalt.....	3
1. Einführung	4
1.1 Zweck	4
1.2 Gültigkeitsbereich	4
1.3 Referenzen	4
2. Projekt Übersicht	5
2.1 Zweck und Ziel.....	5
2.2 Lieferumfang	5
2.3 Annahmen und Einschränkungen	5
3. Projektorganisation	6
3.1 Organisationsstruktur	6
3.2 Externe Schnittstellen	6
4. Management Abläufe	7
4.1 Kostenvoranschlag	7
4.2 Zeitliche Planung	7
4.2.1 Phasen / Iterationen / Arbeitspakete	7
4.2.2 Meilensteine	9
4.3 Besprechungen.....	9
5. Risikomanagement	10
5.1 Risiken	10
5.2 Umgang mit Risiken.....	10
6. Infrastruktur.....	11
7. Qualitätsmassnahmen	12
7.1 Dokumentation	12
7.2 Projektmanagement.....	12
7.2.1 Source	12
7.2.2 Fehlertracking	12
7.2.3 Vorgehen	13
7.2.4 Unit Testing.....	13
7.2.5 Code Reviews	13
7.2.6 Code Style Guidelines	13
7.3 Testen.....	14
7.3.1 Komponententest.....	14
7.3.2 Integrationstest	14
7.3.3 Systemtest	14
8. Verzeichnisse	15



1. Einführung

1.1 Zweck

Dieses Dokument beschreibt die Planung für die Entwicklung des Programms mit dem Namen frontg8 im Rahmen des Moduls „SE2 Projekt“.

1.2 Gültigkeitsbereich

Das Dokument gilt für die Entwicklung der Software frontg8 während des SE2 Projekts im Frühlingssemester 2015.

1.3 Referenzen

Folgende Dokumente werden im vorliegenden Projektplan referenziert:

- Projektantrag.pdf
- TechnischeRisiken.pdf

2. Projekt Übersicht

frontg8 ist ein Text-Chatsystem, vergleichbar mit WhatsApp, mit folgenden Besonderheiten:

1. Sichere Kommunikation, Ende-zu-Ende verschlüsselt und Partner identifizierbar
2. Serverkomponente selber betreibbar
3. Quellcode offen, d.h. er kann eingesehen und überprüft werden.
4. Unterstützung für mehrere Geräte pro Benutzer

2.1 Zweck und Ziel

Unsere Zielpublikum, wozu wir uns selbst zählen, sind geschlossene Benutzergruppen mit sehr hoher Anforderung an die Wahrung der Privatsphäre. Uns reizt es ein Text-Messaging-System zu bauen, über welches eine solche Gruppe vollständige Kontrolle besitzt.

2.2 Lieferumfang

Unsere Lösung besteht aus einer Serverkomponente und einer Desktop-Clientkomponente. Ausgeliefert werden die Quelltexte zu Server und Client, Installations- und Konfigurationsanleitung, sowie eine Architekturbeschreibung und Protokollspezifikation.

Der Server, geschrieben in C++, kümmert ausschliesslich um den Austausch von Text-Messages und wird bewusst sehr schlank gehalten. Dabei ist uns wichtig, dass diese zentrale Stelle möglichst keine Angriffsfläche bietet, d.h. es werden keine Informationen gespeichert, die nicht absolut nötig sind. Die Nachrichten werden in einem flüchtigen Puffer für maximal 3 Tage aufbewahrt.

Der in Python geschriebene Desktop-Client ist eine CLI-Applikation und kümmert sich komplett um die Ende-zu-Ende Verschlüsselung und die Identifizierung des Kommunikationspartners mittels Public-Key-Kryptographie. Er verfügt über ein Adressbuch, damit die „anonymen“ Kommunikationspartner identifiziert und überprüfte Zertifikate zugeordnet werden können.

2.3 Annahmen und Einschränkungen

Die Zielplattform für die Server- und Clientkomponente ist Ubuntu 14.04. Wir gehen von einer starken Portabilität von C++ und Python, bzw. den verwendeten Bibliotheken aus, garantieren die Lauffähigkeit im Rahmen dieses Projekts aber nur die definierte Zielplattform.

Die ursprünglich angedachte Lösung hat auch Applikationen für Android und iOS beinhaltet. In Rücksprache mit Herrn Keller, haben wir uns aufgrund der kurzen Projektzeit entschlossen, die Mobile Applikationen aus dem Projekt zu streichen. In diesem SE2-Projekt geht es dementsprechend nur um die Umsetzung der oben beschriebenen schlanken Server und Client-Komponenten. Wir können uns aber gut vorstellen, dass wir die Handy-Umsetzungen ausserhalb des SE2-Projekts entwickeln.

3. Projektorganisation

An diesem Projekt arbeiten die drei unten aufgeführten Informatik-Studenten. Aufgrund seiner Stärke im Bereich Kommunikation übernimmt Tobias Stauber die Rolle des Projektleiters. Verantwortlich im Bereich C++ ist Felix Morgner. Ueli Bosshard übernimmt die Verantwortung über Netzwerk relevanten Fragen. Trotz der Trennung der Verantwortlichkeiten, soll jedes Teammitglied alles sehen und machen und seine Mitstreiter entsprechend seiner Stärken unterstützen.

3.1 Organisationsstruktur



Felix Morgner

Stärken:

- C, C++
- Netzwerk- / Crypto-programmierung
- MacOS, Objective-C



Tobias Stauber

Stärken:

- Kommunikation
- Linux



Ueli Bosshard

Stärken:

- Systemtechnik (OS, Netzwerk, DB)
- Dokumentation

3.2 Externe Schnittstellen

Das Projekt wird von Herrn Prof. Dr. A. Steffen betreut.

4. Management Abläufe

4.1 Kostenvoranschlag

Das SE2-Projekt findet im Frühlingssemester 2015 statt, dauert also ca. 14 Wochen. Für jede Person wird mit einem Zeitbudget von 120 Stunden gerechnet. Insgesamt können im Projekt 360 Arbeitsstunden verplant werden.

4.2 Zeitliche Planung

Dieser Abschnitt beschreibt die zeitliche Planung, die Einteilung der Iterationen und die Festlegung der Meilensteine. Die detaillierte Zeitplanung erfolgt in Redmine und wird laufend angepasst.

<https://redmine.frontg8.ch/redmine/projects/frontg8/issues/gantt>

Gastzugang (Readonly) ist verfügbar mit Benutzername „gast“ und Passwort „gast“.

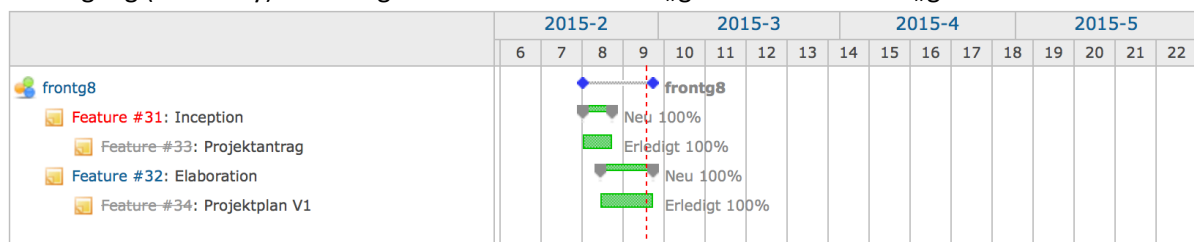


Abbildung 1: Gantt-Diagramm aus Redmine

4.2.1 Phasen / Iterationen / Arbeitspakete

Die folgende Abbildung gibt einen Überblick der Phasen, Iterationen und Meilensteine. Die Meilensteine sind als „x“ markiert und werden nachfolgend beschrieben.

Inception	Elaboration			Construction			Transition
1 Woche Inception	1 Woche 1. Iteration	3 Wochen 2. Iteration	3 Wochen 3. Iteration	2 Wochen 1. Iteration	2 Wochen 2. Iteration	1 Woche 3. Iteration	2 Wochen Transition
		x MS1: Projektplan					
			x MS2: Anforderungen und Analyse				
				x MS3: End of Elaboration			
						x MS4: Architektur & Design	
						x MS5: Feature Freeze	
						MS6: Schlusspräsentation x	
Wo 1	Wo 2	Wo 3-5	Wo 6-8	Wo 9-10	Wo 11-12	Wo 13	Wo 14-15

Abbildung 2: Iterationen und Meilensteine

Iteration	Arbeitspakete	Meilensteine	SW
Inception	- Projektantrag schreiben - Antrag Virtueller Server		1
Elaboration 1	- Projektplan Schreiben - Arbeitspakete bestimmen - Risikoanalyse - Domainanalyse - Usecase Analyse - Setup Projektinfrastruktur	MS1: Abgabe Projektplan	2
Elaboration 2	- Kernusecases fully-dressed formalisiert - Domainmodell (Detaillierte Domainanalyse) - Protokollspezifikation - Crypto-Design - Überarbeitung Projektplan - Requirements Analyse	MS2: Anforderungen und Analyse	3-5
Elaboration 3	- Toolchain Setup - Prototyp Server - Prototyp Client - Prototyp Protokoll - Test-Harness	MS3: EOE (Demonstration)	6-8
Construction 1	- Client TUI - Server Storage - Client Storage		9-10
Construction 2	- Key Exchange - Kontaktverwaltung - Server Parallelität - Client Gruppenchat - Architekturdokumentation - Server Config-File support	MS4 : Architektur Review MS5: Feature Freeze	11-12
Construction 3	- Code cleanup - Bug Fixing Client - Bug Fixing Server		13
Transition	- Schlusspräsentation Schreiben - User-Manual schreiben	MS6: Schlusspräsentation	14-15

Tabelle 1: Iterationen und Arbeitspakete

Die aktualisierten Arbeitspakete können im Redmine eingesehen werden.

4.2.2 Meilensteine

4.2.2.1 MS1 Projektplan

Termin: Woche 3 - Do, 05.03.2015

Review Projektplan mit Zeitplan und aktuellen Iterationsplänen

4.2.2.2 MS2 Anforderungen und Analyse

Termin: Woche 5 - Fr, 20.03.2015

Review der Anforderungsspezifikation und der Domainanalyse

4.2.2.3 MS3 Ende Elaboration

Termin: Woche 8 - Fr, 10.04.2015

Zwischenpräsentation mit Demo eines Architekturprototypen, Review

4.2.2.4 MS4 Architektur/Design

Termin: Woche 9-11 - Fr, 08.05.2015

Review von Architektur/Design und Architekturdokumentation

4.2.2.5 MS5 Feature Freeze

Termin: Woche 12

Ab diesem Zeitpunkt werden keine neuen Features mehr umgesetzt. Codeänderungen sind dann nur noch Bug-Fixes/Cleanups.

4.2.2.6 MS6 Schlusspräsentation/-abgabe

Termin: Woche 15 - 29.05.2015

Präsentation und Demo der Software: Do 28.5. und Fr 29.5. 2015

Spätester Abgabetermin: Freitag, den 29. Mai 2015

4.3 Besprechungen

Die Besprechung in der Projektgruppe findet jeden Freitag um 10:10 Uhr statt. Ziel ist es, den aktuellen Stand und mögliche Probleme zu besprechen und Arbeiten zu verteilen.

5. Risikomanagement

5.1 Risiken

Siehe Dokument TechnischeRisiken.pdf

5.2 Umgang mit Risiken

Durch seriöse Planung glauben wir die Risiken einigermaßen im Griff zu haben. Da in der knappen Zeit keine Reserven vorgesehen sind, bleibt uns nur ein Mehreinsatz oder der Verzicht auf gewisse Features.

Bei kryptographischen Problemen, die wir nicht unter Zuhilfenahme von Dokumenten aus InfSi1/2 oder Fachliteratur lösen können, wenden wir uns an Herrn Prof. Dr. Steffen.

6. Infrastruktur

Jedes Teammitglied arbeitet mit seinem eigenen Laptop, ausgestattet mit folgender Software: Browser für Zugang zu Tools auf dem Linux-Server, Entwicklungsumgebung für C++ und Python inkl. git-Integration.

Zur Koordination verfügen wir über einen virtuellen Linux-Server der HSR, auf dem Redmine und Jenkins eingerichtet sind. Dieser Server steht nur bis Projektende (Ende Mai 2015) zur Verfügung.

Softwareauflistung

Bezeichnung	Version	Beschreibung	Zugang
Redmine	2.4	Ticketsystem, Planung	https://redmine.frontg8.ch
git	1.9	Versionskontrolle	https://source.arknet.ch/groups/frontg8
Jenkins	1.599	Build-Server	http://gerrit.frontg8.ch
vim	7.4	Entwicklungsumgebung	\$ vim
cmake	2.8	Build-Tool	\$ cmake

Tabelle 2: Eingesetzte Software

7. Qualitätsmassnahmen

Massnahme	Zeitraum/Intervall	Ziel
Zeiterfassung	Fortlaufend	Soll-/Ist-Vergleich
Besprechungen/Reviews	Wöchentlich (Freitag)	Fortschrittskontrolle
Seriöse Anforderungserfassung	Elaboration	Probleme frühzeitig erkennen um nicht zielführende Arbeiten zu reduzieren
Detaillierte Analyse	Elaboration	s.o.
Coding Style Guidelines	Elaboration	Einheitlicher Stil vereinfacht Reviews und verhindert semantische Fehler
Umsetzung der Software-Engineering Techniken aus SE1 und SE2	Ganzes Projekt	Typische Fehler vermeiden, Qualitätsniveau erhöhen

Tabelle 3: Qualitätsmassnahmen

7.1 Dokumentation

Da die Dokumentation auf den bereitgestellten Vorlagen beruht setzen wir binäre Dateitypen ein, welche git zu einer schlechten Wahl machen. Deshalb werden wir die Dokumentation auf Redmine hinterlegen.

Folgende Objekte müssen dokumentiert werden:

- Public Klassen und Interfaces
- Jegliche Methoden ausser:
 - Sie implementieren ein Interface oder überschreiben eine Methode einer Basisklasse
 - Sind private
 - Jegliche nicht private Felder
- Bei der Dokumentation von Methoden sollte folgendes Angegeben werden:
 - Alle Parameter
 - Was ist der Return-Wert
 - null erlaubt (Parameter und Return-Wert)
 - Was ist die Funktion der Methode
 - Welche Exceptions werden in welchem Fall geworfen.

7.2 Projektmanagement

Für das Projektmanagement setzen wir Redmine ein. Somit hat auch unser Betreuer einen Einblick in den aktuellen Stand der Entwicklung.

7.2.1 Source

Unsere Sourcefiles liegen auf unserem GitLab. Die Qualitätssicherung wird durch den Einsatz von Gerrit (Code-Review & Erfolgreicher build) gewährleistet. Andere Teile des Programms, wie etwa Bibliotheken werden wir nicht auf GitLab halten.

7.2.2 Fehlertracking

Fehler werden in einem Redmine-Ticket erfasst. Wenn in dieser Iteration ausreichend Zeit vorhanden ist, werden wir den Fehler, falls sinnvoll sofort beheben. Andernfalls wird die Behebung in eine spätere Iteration verlegt.

7.2.3 Vorgehen

7.2.3.1 Regelmässige Treffen

Jeden Freitag investieren wir in die Planung, Problemanalyse und Lösungsfindung der anstehenden Arbeitsschritte.

7.2.3.2 Iterations Retrospektiven

Am Ende einer Iteration werden wir jeweils einen Rückblick durchführen und festhalten. Die Bewertung des Rückblicks sollte mindestens folgende Fragen abdecken:

- Was hat gut funktioniert?
- Was hat weniger oder nicht gut funktioniert?
- Gibt es gewisse Ziele die wir nicht erreicht haben und jetzt neu platziert werden sollen? Falls ja, welche Massnahmen sind für die nächste Iteration zu treffen?

Durch diese Rückblicke wird die Produktivität des Teams optimiert.

7.2.4 Unit Testing

Unittests müssen für folgende Bereiche vorhanden sein:

- Server
- Client

7.2.5 Code Reviews

Die Code Reviews müssen gemacht werden, bevor ein Commit von Gerrit in das GitLab-Repo übernommen werden kann.

7.2.6 Code Style Guidelines

7.2.6.1 C++

Für C++ verwenden wir eine spezialisierte Version des Whitesmith-Stils. Hervorheben möchten wir folgende Anpassungen:

- Eingerückt wird mit 2 Leerzeichen, statt eines Tabulators
- Access specifiers werden eingerückt, zudem werden die Blöcke nochmal eingerückt.
- Die Breite ist nicht limitiert.

7.2.6.2 Python

Für Python verwenden wir PEP8.

7.3 Testen

7.3.1 Komponententest

Zu den meisten Klassen werden Unit-Tests geschrieben. Durch Gerrit kann kein Commit in das Hauptrepository gelangen, welcher nicht alle Tests bestanden hat.

7.3.2 Integrationstest

Während der Umsetzung werden verschiedene Integrationstest durchgeführt. Die Tests werden nach der Umsetzung einer Funktion durchgeführt. Eine Funktionalität gilt erst als implementiert, wenn alle Tests erfolgreich abgeschlossen wurden und die Anforderungen als erfüllt angesehen werden.

7.3.3 Systemtest

Am Ende jeder Construction-Iteration, sowie während der Transition-Iteration, wird die Funktionalität von netzwerkbezogenen Anforderungen manuell überprüft. Dafür werden wir als Team den Server und Client intensiv testen. Fehler können dann gezielt behoben werden.



8. Verzeichnisse

Abbildungen:

Abbildung 1: Gantt-Diagramm aus Redmine	7
Abbildung 2: Iterationen und Meilensteine.....	7

Tabellen:

Tabelle 1: Iterationen und Arbeitspakete	8
Tabelle 2: Eingesetzte Software.....	11
Tabelle 3: Qualitätsmassnahmen	12