Software Design Description

Created by Auto Mation, last modified by Thijs Baan on Jun 17, 2020

User	Edits	Comments	Last Update
Yuri Ruler	74	9	234 days ago
Jelmer Wijnja	66	3	234 days ago
Tim Hemmes	66	5	240 days ago
Dennis Hakvoort	55	34	234 days ago
Stefan Anbeek	52	13	228 days ago
Thijs Baan	50	4	227 days ago
Mario Blaucik	46	12	229 days ago
Jonathan Vandionant	42	7	235 days ago
Rutger Broekkamp	38	5	229 days ago
Coen Hoogduin	33	9	229 days ago
Vu Le	31	33	228 days ago
Mark Ogink	29	3	234 days ago
Yildirim Sengul	28	1	234 days ago
Roel van de Wiel	26	0	234 days ago
Sven Molhuijsen	26	0	234 days ago
Gino vd Bund	21	25	228 days ago
Liam Rougoor	6	7	234 days ago
Robin van der Vliet	5	0	251 days ago
Auto Mation	1	0	355 days ago
Rody Middelkoop	1	0	237 days ago
Ivan Miladinovic	0	0	
Pepijn Erp	0	0	
Yael Bakker	0	0	

Wijziging 11 juni: hele document verbeterd op Nederlandse taal inclusief spelfouten.

Inhoudsopgave

1. Introduction

1.1. Overall Description

Een gedetailleerde omschrijving van de opdracht wordt gegeven in hoofdstuk 2. Achtergrond van het project van het plan van aanpak. Een toelichting hierop wordt in hoofdstuk 2 van het Software Architecture Document gegeven. De scope van dit project wordt toegelicht in hoofdstuk 4. Projectgrenzen van het plan van aanpak. Om het doel van het project inzichtelijk te krijgen wordt er een toelichting gegeven in hoofdstuk 3. Doelstelling van het plan van aanpak.

1.2. Purpose of this document

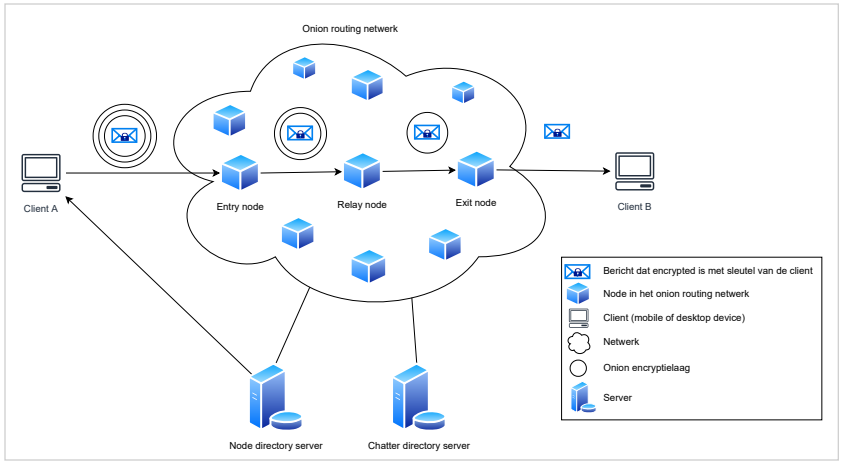
Het doel van dit document is om de technische kant van de Submarine-chatapplicatie te beschrijven. Zo kan tijdens het ontwikkelen van de code rekening gehouden worden met dit document, alsmede het Software Requirements Specification. Dit document licht dan ook de richtlijnen van alle deelsystemen toe. Dit document is voor het huidige en toekomstige development team, zodat de richtlijnen gevolgd kunnen worden in de te bouwen producten. Mochten er in de toekomst ontwikkelteams zijn die verder willen bouwen op onze software, zal dit document voor hun ook een belangrijke bron zijn van informatie.

1.3. Definitions, acronyms, and abbreviations

Term	Description
SRS	Software Requirements Specification
SAD	Software Architecture Document
BLoC	Business Logic Component. Een statemanagement pattern

Tabel 1: Begrippenlijst

2. Architectural Overview

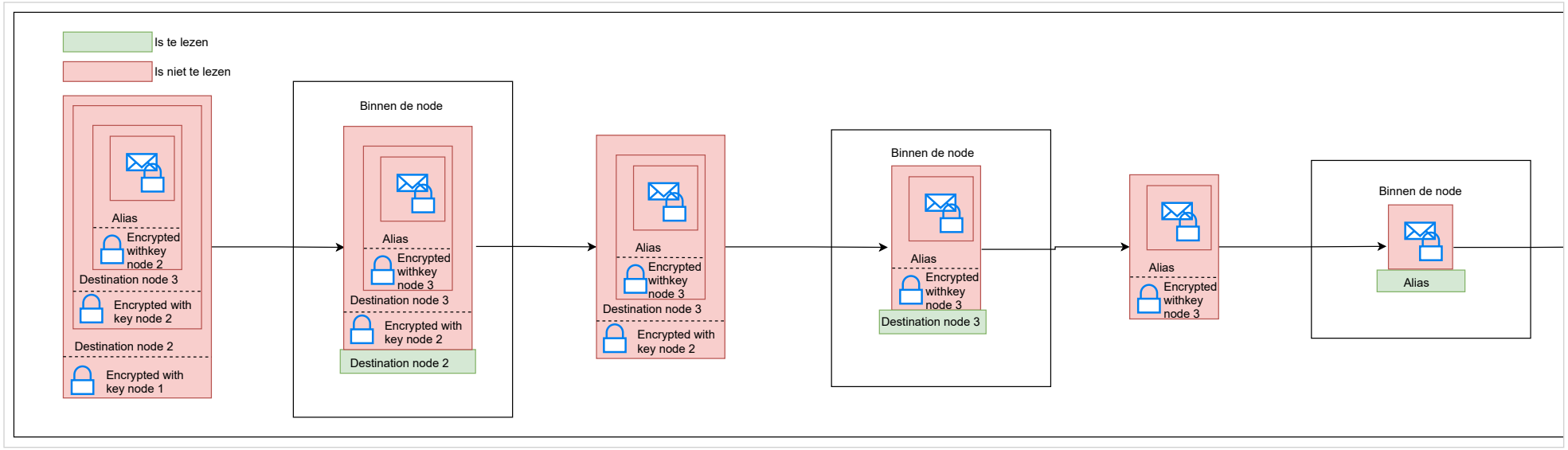


Figuur 1: Architecturaal overview

In het architecturaal overview wordt duidelijk gemaakt dat er servers en clients zijn. Clients communiceren via het netwerk van nodes(het onion netwerk) met elkaar. Dit doen ze door versleutelde berichten te sturen. Om de route voor het bericht te bepalen wordt de server benaderd. De server zal dan de identificatie van een aantal nodes terug sturen naar de client, waarna de client het bericht op de juiste manier kan versleutelen en versturen.

2.1. Toelichting onion

Wijziging 11 juni: Figuur uitgebreid zodat de symmetric key ook wordt weergegeven.



Figuur 2: Versturen bericht door nodes

In Figuur 2 hierboven is te zien hoe een bericht door de nodes heen gaat en welke gegevens het bevat. In het rood is te zien welke gegevens versleuteld zijn en niet zichtbaar zijn. In het groen zijn de gegevens te zien die wel beschikbaar zijn.

Het volgende gebeurt in dit proces:

1. Er komt een versleutelde bericht in binnen node 1.
2. Deze versleutelde bericht wordt ontsleuteld met de key binnen de node.

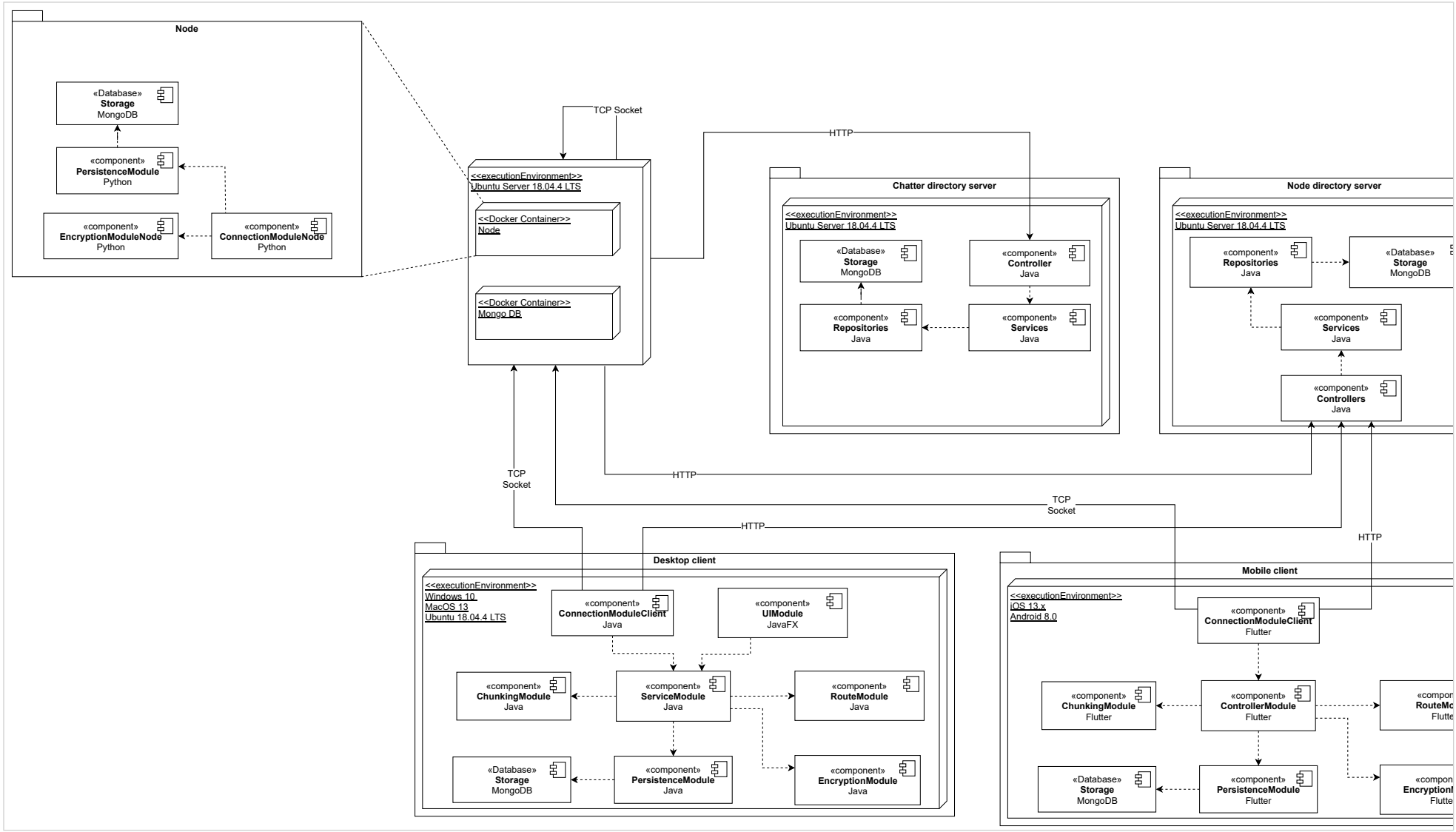
3. De eerste node ziet de destination (node 2) en stuurt deze door.
4. Node 2 ontvangt het versleutelde bericht.
5. Node 2 ontsleuteld het versleutelde bericht met zijn key.
6. De tweede node ziet de destination (node 3) en stuurt deze door.
7. Node 3 ontvangt het versleutelde bericht.
8. Node 3 ontsleuteld het versleutelde bericht met zijn key.
9. Node 3 zoekt de IP-adres van de ontvanger door zijn alias te gebruiken.
10. Node 3 stuurt het versleutelde bericht toe naar de eindgebruiker.

3. Detailed Design Description

In dit hoofdstuk wordt dieper ingegaan op de design van de software componenten. Dit wordt gedaan door middel van een deployment diagram, een toelichting op de datamodellen die door de systemen heen gebruikt worden, de standaard waarden die gebruikt worden voor testen etc. etc.

3.1. Deployment diagram

Wijziging 11 juni: Database in node in aparte docker container weergegeven. Diagram voorzien van uitleggende teksten.



Figuur 3: Deployment Diagram

3.1.1. Ontwerpbeslissingen met betrekking tot deployment

Wijziging 11 juni: toegelicht dat Nodes in docker containers draaien zodat deze eenvoudig vermeervoudigd kunnen worden.

3.2. Datamodellen

In dit hoofdstuk staan alle models die gebruikt worden in de applicatie. Bij elk model staan de attributen met datatype benoemt. Indien een team een model aanpast, dan dient dit eerst overlegd te zijn met alle andere teams zodat er geen verschil in datamodellen ontstaat. Deze models zijn een visuele representatie van de JSON die tussen de verschillende systemen wordt verstuurd.

Wijziging 11 juni: datamodellen gelijk getrokken met de code. Beschrijvingen verduidelijkt indien nodig.

3.2.1. TextMessage

TextMessage

```
{
   "title":"TextMessage",
   "description":"Een message waarin een tekstbericht meegegeven kan worden.",
   "attributes":{
      "sender":{
         "type":"string",
         "description":"De alias van de verzender.",
         "example":"John_Doe"
      },
      "timestamp":{
         "type":"timestamp",
         "description":"De tijd waarop de message is aangemaakt.",
         "example":"01-01-2020"
      },
      "conversationId":{
         "type":"string",
         "description":"Het id van de chat waarin de message is verstuurd. Dit is een UUID.",
         "example":"550e8400-e29b-41d4-a716-446655440000"
      },
      "message":{
         "type":"string",
         "description":"De inhoud van het bericht.",
         "example":"Dit is een berichtje"
      }
   }
}
```

Figuur 4: Datamodel TextMessage

3.2.2. FileMessage

FileMessage

```
{
   "title":"FileMessage",
   "description":"Een message waarin een bestand meegegeven kan worden.",
   "attributes":{
      "sender":{
         "type":"string",
         "description":"De alias van de verzender.",
         "example":"John_Doe"
      },
      "timestamp":{
         "type":"timestamp",
         "description":"De tijd waarop de message is aangemaakt.",
         "example":"01-01-2020"
      },
      "conversationId":{
         "type":"string",
         "description":"Het id van de chat waarin de message is verstuurd. Dit is een UUID.",
```

```
         "example":"550e8400-e29b-41d4-a716-446655440000"
      },
      "fileName":{
         "type":"string",
         "description":"De naam van het bestand.",
         "example":"File_Name"
      },
      "fileType":{
         "type":"string",
         "description":"Het bestandsformaat.",
         "example":".txt"
      },
      "fileSize":{
         "type":"integer",
         "description":"Het bestandsformaat in .. kB(?).",
         "example":1000
      },
      "fileLocation":{
         "type":"Path",
         "description":"De path naar het bestand op de lokale machine.",
         "example":"C://etc etc(?)"
      }
   }
}
```

Figuur 5: Datamodel FileMessage

3.2.3. Conversation

Conversation

```
{
   "title":"Conversation",
   "description":"De chat waarin een of meerdere gebruikers met elkaar communiceren.",
   "attributes":{
      "id":{
         "type":"string",
         "description":"De id van de conversation. Dit is een UUID.",
         "example":"550e8400-e29b-41d4-a716-446655440000"
      },
      "participants":{
         "type":"Contact array",
         "description":"De deelnemers van een conversation.",
         "example":"["Contact object", "Contact object"]"
      },
      "messages":{
         "type":"message array",
         "description":"De messages die zich in een conversation bevinden.",
         "example":"["Message_Object", "Message_Object"]"
      }
   }
}
```

Figuur 6: Datamodel Conversation

3.2.4. Onion

Onion voor message naar Client

De JSON-opbouw van Figuur 7 is voor een Message die verzonden wordt over het Node-netwerk.

Let op de nummers helemaal vooraan de string. Dit is de lengte van de content die volgt. Zonder deze nummers wordt het bericht als foutief gemarkeerd. Dit moeten de clients toevoegen alvorens het versturen naar de entry node.

Onion

```
123{
   "title":"Onion",
   "description":"Een onion die verstuurd wordt vanaf een client en doorgestuurd wordt in het node netwerk.",
   "attributes":{
      "destination":{
         "type":"Destination",
         "description":"De bestemming waar de onion naartoe moet. Dit is of een DestinationClient_Object of een DestinationRelay_Object is.",
         "example": "DestinationClient_Object" || "DestinationRelay_Object"
      },
      "command":{
         "type":"string",
         "description":"De command van de onion. Dit is altijd RELAY,
         "example":"RELAY"
      },
      "data":{
         "type":"string",
         "description":"De data die meegegeven wordt in een onion. Het data attribuut is een onion-object behalve als het ontsleuteld wordt bij de exit-node dan is het een message-ob
         "example":"ubt4mCbaeUIi6KlDsyQwU/BRTvr/J9JHLY229hRxGG6OcLrCg6oL3ELMaFvXm944T97T3MooTLm+Lle0msuC0mTs7PBtdDSkYYU96QUdaanjgbuBc4bWi2dxr+ZiMW1LbB7KCpwJzj//Q8XfrrremRj+p7MuLwjRgM
                   Bg/njhNUy63fkRj20cOoRlwOejTkHxKtyooxsv4N/WJXiFo8F74CtgLB5qbI2a5PGFmVWEI476QTlBWU0Gja/zYxneFfOBj1bSS98KTTbG8sCg\u003d\u003d"
      }
   }
}
```

Figuur 7: Datamodel Onion

Onion voor berichten naar ChatterDirectoryServer

Voor de berichten naar de ChatterDirectoryServer wordt gebruikt gemaakt van de opbouw zoals te zien in Figuur 8. De body is afhankelijk van de request die nodig is, de endpoints zijn hier terug te vinden. De JSON is uitgewerkt voor een login-request.

Let op de nummers helemaal vooraan de string. Dit is de lengte van de content die volgt. Zonder deze nummers wordt het bericht als foutief gemarkeerd. Dit moeten de clients toevoegen alvorens het versturen naar de entry node.

Onion

```

123{
   "title":"Onion",
   "description":"Een onion die verstuurd wordt vanaf een client en doorgestuurd wordt naar de ChatterDirectoryServer",
   "attributes":{
      "destination":{
         "type":"Destination",
         "description":"De bestemming waar de onion naartoe moet. Dit is of een volgende relay node of null",
         "example": "DestinationRelay_Object" || null
      },
      "command":{
         "type":"string",
         "description":"De command van de onion. Bij de exit node is dit HTTP_REQUEST, bij de andere nodes is dit HTTP_RELAY",
         "example":"HTTP_RELAY"
      },
```

```
      "data":{
                "endpoint":{
                        "type":"string",
                        "description":"De endpoint waar de request naar toe moet. Dit bevat de server IP en de endpoint.",
                        "example":"http://94.124.143.192/chatter/login"

                },
                "request_type":{
                        "type":"string",
                        "description":"De soort endpoint dat bereikt wordt. Dit is PUT, DELETE, POST of GET.",
                        "example":"PUT"
                }
                "body":{
                    "username": {
                        "type":"string",
                        "description":"Het username van de chatter. Dit is plain text.",
                        "example":"hackerman123"

                    },
                    "password": {
                        "type":"string",
                        "description":"Het wachtwoord van de chatter. Dit is plain text.",
                        "example":"htmlIsProgramming"
                    },
                    "ipAddress":{
                        "type":"string",
                        "description":"De IP-address van de chatter.",
                        "example":"123.123.123"
                    }
            }
      }
   }
}
```

Figuur 8: Datamodel Onion

3.2.5. Node

Node

```
{
   "title":"Node",
   "description":"Een node dat deel uitmaakt van het node netwerk.",
   "attributes":{
      "destination":{
         "type":"Destination",
         "description":"De destination met daarin de adresgegevens van de node.",
         "example":"DestinationRelay_Object"
      },
      "key":{
         "type":"string",
         "description":"De symmetrische sleutel van de node.",
         "example":"X/GAXebJQR56ncdUsj3hTqKIryYYbywxyjtTxlGodDBLX7uj8jEwN/BletIKDdteZ6+IldKphzdAbaSQ+Edanax7xokp3vuWrb/9hR9Z2CfxumRDL2h+k5+2o12wVXoOqz
                   PeqGqEECX8v8tyGwSvBsQ4BBSXzVjdk/n0hzIZnGQ1831x4TVWLv6E4srSbbcTxK/QhuS+6xAYvqxY3oIcAb3hBQj/HgMfJ6kzLCG5mYhwqhsWDnQi/2X03OepcBWW3s+j
                   yHki1ufdZCgYMh4JXKXZjFN6tJ6SDXVBcWI02dCyuZAccl3rMMOYr2qQ7nBeNqpuc9LCn+uP+nPGqwxrew=="
```

```
        }
    }
}
```

Figuur 9: Datamodel Node

3.2.6. DestinationRelay

DestinationRelay

```
{
   "title":"DestinationRelay",
   "description":"De DestinationRelay wordt gebruikt voor alle destinations die van toepassing zijn voor de nodes. Hier in zit de hostname en de port van de node.",
   "attributes":{
      "hostname":{
         "type":"string",
         "description":"Het ip adres van de node.",
         "example":"193.134.4.5"
      },
      "port":{
         "type":"integer",
         "description":"De poort van de node.",
         "example": 25010
      }
   }
}
```

Figuur 10: Datamodel DestinationRelay

3.2.7. DestinationClient

DestinationClient

```
{
   "title":"DestinationClient ",
   "description":"De DestinationClient wordt gebruikt voor de destination die van toepassing zijn voor de exit node naar de client. Hier zit alleen de alias van de ontvanger in.",
   "attributes":{
      "alias":{
         "type":"string",
         "description": "De alias van de ontvanger.",
         "example":"John_Doe"
      },
      "encryptedSymmetricKey":{
         "type":"string",
         "description":"De symmetrische sleutel waarmee de message is versleuteld.",
         "example": "X/GAXebJQR56ncdUsj3hTqKIryYYbywxyjtTxlGodDBLX7uj8jEwN/BletIKDdteZ6+IldKphzdAbaSQ+Edanax7xokp3vuWrb/9hR9Z2CfxumRDL2h+k5+2o12wVXoOqzPeqGqEECX8v8tyGwSvBsQ4BBSXzVj
                    dk/n0hzIZnGQ1831x4TVWLv6E4srSbbcTxK/QhuS+6xAYvqxY3oIcAb3hBQj/HgMfJ6kzLCG5mYhwqhsWDnQi/2X03OepcBWW3s+jyHki1ufdZCgYMh4JXKXZjFN6tJ6SDXVBcWI02dCyuZAccl3rMMOYr2qQ7nB
                    eNqpuc9LCn+uP+nPGqwxrew=="
      }
   }
}
```

Figuur 11: Datamodel DestinationClient

3.2.8. Path

Path

```
{
   "title":"Path",
   "description":"Het pad wat een onion af moet leggen door het node netwerk om bij de ontvanger te komen.",
   "attributes":{
      "nodes":{
         "type":"Node array",
         "description": "De nodes van het pad.",
         "example":"[NODE_OBJECT, NODE_OBJECT, NODE_OBJECT]"
      }
   }
}
```

Figuur 12: Datamodel Path

3.2.9. ChatterRegistrationDTO

De public key is een Base64-encoded string welke een representatie is van de bytes van de public key. Deze string moet op een ontvangende locatie worden omgezet naar de juiste implementatie van een PublicKey.

ChatterRegistrationDTO

```
{
   "title":"ChatterRegistrationDTO",
   "description":"Een ChatterRegistrationDTO zoals deze wordt meegegeven met aanroepen aan de Chatter Directory Server.",
   "attributes":{
      "alias":{
         "type":"string",
         "description": "De alias van de contact.",
         "example":"John_Doe"
      },
      "publicKey": {
         "type": "string",
         "description": "De public key van de contact om berichten mee te versleutelen.",
         "example": "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsvX9P58JFxEs5C+L+H7WduFSWL5EPzber7C2m94klrSV6q0bAcrYQnGwFOlveThsY200hRbadKaKjHD7qI
                    KHDEe0IY2PSRht33Jye52AwhkRw+M3xuQH/7R8LydnsNFk2KHpr5X2SBv42e37LjkEslKSaMRgJW+v0KZ30piY8QsdFRKKaVg5/Ajt1YToM1YVsdHXJ3vmXFMtypLd
                    xwUDdIaLEX6pFUkU75KSuEQ/E2luT61Q3ta9kOWm9+0zvi7OMcbdekJT7mzcVnh93R1c13ZhQCLbh9A7si8jKFtaMWevjayrvqQABEcTN9N4Hoxcyg6l4neZtRDk75O
                    MYcqmDQIDAQAB"
      },
      "username": {
         "type": "string",
         "description": "De gebruikersnaam van de gebruiker.",
         "example": "John_Doe"
      },
      "password": {
         "type": "string",
         "description": "Het wachtwoord van de gebruiker (niet gehashet)",
         "example": "ThisIsNotMyPassword"
      },
      "ipAddress": {
         "type": "string",
         "description": "Het IP-adres van de gebruiker waarmee hij met het node netwerk verbind..",
```

```
         "example": "127.0.0.1"
      }
   }
}
```

Tabel 2: ChatterRegistrationDTO Model

3.2.10. ChatterLoginDTO

ChatterLoginDTO

```
{
   "title":"ChatterLoginDTO",
   "description":"Een ChatterLoginDTO zoals deze wordt meegegeven met aanroepen aan de Chatter Directory Server.",
   "attributes":{
      "username": {
         "type": "string",
         "description": "De gebruikersnaam van de gebruiker.",
         "example": "John_Doe"
      },
      "password": {
         "type": "string",
         "description": "Het wachtwoord van de gebruiker (niet gehashet)",
         "example": "ThisIsNotMyPassword"
      },
      "ipAddress": {
         "type": "string",
         "description": "Het IP-adres van de gebruiker waarmee hij met het node netwerk verbind..",
         "example": "127.0.0.1"
      }
   }
}
```

Figuur 13: ChatterLoginDTO

3.2.11. ChatterAddressDTO

ChatterAddressDTO

```
{
   "title":"ChatterAddressDTO",
   "description":"Een ChatterAddressADTO wordt gemaakt in de Chatter Directory Server als er van uit
                  een Node een getIP-request komt. De IPaddress is het IP address die hoort bij de alias",
   "attributes":{
        "ipAddress": {
            "type": "string",
            "description": "Het IP-adres van de chatter",
            "example": "127.0.0.1"
     }
}
```

Figuur 14: ChatterAddressDTO

3.2.12. Contact

Contact

```
{
   "title":"Contact",
   "description":"Een contact die een gebruiker heeft om mee te communiceren in de applicatie.",
   "attributes":{
      "alias":{
         "type":"string",
         "description": "De alias van de contact.",
         "example":"John_Doe"
      },
      "publicKey": {
         "type": "string",
         "description": "De public key van de contact om berichten mee te versleutelen.",
         "example": "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsvX9P58JFxEs5C+L+H7WduFSWL5EPzber7C2m94klrSV6q0bAcrYQnGwFOlveThsY200hRbadKaKjHD7qIKHDEe
                    0IY2PSRht33Jye52AwhkRw+M3xuQH/7R8LydnsNFk2KHpr5X2SBv42e37LjkEslKSaMRgJW+v0KZ30piY8QsdFRKKaVg5/Ajt1YToM1YVsdHXJ3vmXFMtypLdxwUDdIaLEX6
                    pFUkU75KSuEQ/E2luT61Q3ta9kOWm9+0zvi7OMcbdekJT7mzcVnh93R1c13ZhQCLbh9A7si8jKFtaMWevjayrvqQABEcTN9N4Hoxcyg6l4neZtRDk75OMYcqmDQIDAQAB"
      }
   }
}
```

Figuur 15: Datamodel Contact

3.2.13. UserData

UserData

```
{
   "title":"UserData",
   "description":"Dit is data van de current user, hiermee kan onderscheid gemaakt worden tussen contacten en de gebruiker.",
   "attributes":{
      "alias":{
         "type":"string",
         "description":"De alias van een gebruiker.",
         "example":"John_Doe"
      },
      "publicKey":{
         "type":"string",
         "description":"De public key is een Base64 encoded string welke een representatie is van de bytes van de public key. Deze string moet
                        op een ontvangende locatie worden omgezet naar de juiste implementatie van een PublicKey.",
     "example":"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsvX9P58JFxEs5C+L+H7WduFSWL5EPzber7C2m94klrSV6q0bAcrYQnGwFOlveThsY200hRbadKaKjHD7qIKHDE
        e0IY2PSRht33Jye52AwhkRw+M3xuQH/7R8LydnsNFk2KHpr5X2SBv42e37LjkEslKSaMRgJW+v0KZ30piY8QsdFRKKaVg5/Ajt1YToM1YVsdHXJ3vmXFMtypLdxwUDdIaLEX6pFUkU
        75KSuEQ/E2luT61Q3ta9kOWm9+0zvi7OMcbdekJT7mzcVnh93R1c13ZhQCLbh9A7si8jKFtaMWevjayrvqQABEcTN9N4Hoxcyg6l4neZtRDk75OMYcqmDQIDAQAB"
      },
      "privateKey":{
         "type":"string",
         "description":"De private key is een Base64 encoded string welke een representatie is van de bytes van de private key. Deze string moet op een ontvangende locatie worden omg
         "example":"MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDFwRRX5n+/9YMYbod8kHJ/M3Y36kEpRtxyAW/+AhUpiC6hWzBHRyFv9DQGy3Z+DzFX9udYohYDH4r97ZWsjr0rBMgrW5vg//k4h/QBs9kVqJgXbg
      }
   }
}
```

Figuur 16: Datamodel UserData

3.2.14. Ontwerpbeslissingen met betrekking op datamodellen

3.3. Algemene ontwerpbeslissingen

In dit hoofdstuk staan de ontwerpbeslissingen welke invloed hebben op alle deelsystemen. Enkele van de beslissingen zijn gemaakt om tijdelijk een probleem op te lossen. Andere beslissingen hebben een groter effect op de werking van de verschillende deelsystemen.

3.3.1. Symmetrische encryptiesleutel

Beslissing	Beschrijving
Probleem/Issue	In een meest ideale situatie zou je voor iedere node een eigen unieke encryptiesleutel willen genereren. Echter is er nog geen manier om de keys te delen met clients.

Beslissing	Er is gekozen om hard-coded een sleutel in de code te zetten die door iedere node gebruikt wordt.
	Deze sleutel is als volgt:
	`Secret Key: 9H+5ObOd2IO/KQcGTTnKR5h3F1DrWFjXt0oud+zlL0Q=`
	`IV: 0705090625458532`

Alternatieven	In de code of file system een unieke key zetten en die vervolgens distribueren naar de code van de clients.
Argumenten	Het scheelt voor nu een hoop distributieproblemen, doordat we niet voor iedere unieke node in de client moeten definiëren welke key daartoe behoort. Ook is het niet handig om in de code van de nodes hard-coded unieke keys te definiëren, omdat dan de code van iedere node verschilt.
Gemaakt op	25 May 2020
Gemaakt door	@Stefan Anbeek

Tabel 3: Beslissing: symmetrische encryptiesleutel

3.3.2. Verbinding node naar client

Beslissing	Beschrijving
Probleem/Issue	Er moet een manier zijn voor de Node om een verbinding te krijgen met de Client waarover een bericht kan worden verzonnen
Beslissing	De client opent een poort voor het publiek, en de node zal hiermee een connectie openen en deze sluiten nadat het bericht is verstuurd.
Alternatieven	De Client opent een connectie met alle nodes, en luisterd hierop. Dit voorkomt dat de client zijn poort open hoeft te zetten.
Argumenten	De port forwarding optie vereist minder implementatiewerk.
Gemaakt op	?
Gemaakt door	@Dennis Hakvoort

Tabel 4: Beslissing: Verbinding node naar client

3.3.3. Identificatie einde bericht

Beslissing	Beschrijving
Probleem/Issue	De nodes weten niet hoe lang het bericht is dat zij gaan ontvangen, dus er moet een manier zijn om het einde van het bericht te vinden.
Beslissing	De eerste paar karakters van het bericht bevatten een nummer met hierin de lengte van het bericht.
Alternatieven	Bij het ontvangen van een bericht wacht je een paar seconden om er zeker van te zijn dat alles binnen is.
Argumenten	Als er meerdere berichten binnen enkele seconden binnen komen kunnen ze in elkaar verzeild raken en komt geen van allen aan.
Gemaakt op	?
Gemaakt door	@Rutger Broekkamp , @Dennis Hakvoort .

Tabel 5: Beslissing: identificatie einde bericht

3.3.4. Symmetrische encryptie bericht

Beslissing	Beschrijving
Probleem/Issue	Een object wordt symmetrisch geëncrypt nadat dit is is opgedeeld in bytearray. Hier kan de Node niet goed mee werken omdat Python geen negatieve waarden in een byte kent. Java wel.
Beslissing	Het object moet omgezet worden naar een JSONstring, deze string wordt vervolgens geëncrypt en via Base64 omgezet naar een String.
Alternatieven	• De object naar bytearray serializen en dit in de node deserlializen. Dit soort echter voor andere complicaties in de Node.
Argumenten	• Huidige oplossing is generiek voor alle objecten • Dit heeft de minste impact op de applicatie
Gemaakt op	?
Gemaakt door	Probleem aangekaart door team Nodes, oplossing bedacht in een discussie met @Thijs Baan , @Yildirim Sengul , @Stefan Anbeek , @Tim Hemmes Tabel 6.

Tabel 6: Beslissing: symmetrische encryptie bericht

3.3.5. Gebruik van account op verschillende clients

Beslissing	Beschrijving
Probleem/Issue	Voor het gebruik van een account is de private key hiervan nodig. Dit is op dit moment niet mogelijk in ons systeem.
Beslissing	We hebben ervoor besloten dat een gebruiker alleen accounts kan gebruiken die hij op die ene cliënt gemaakt heeft.
Alternatieven	• De private key kan op een database beveiligd opgeslagen worden. • De gebruiker kan zijn account door middel van een QR-code met zijn eigen cliënts verbinden.
Argumenten	• Er is weinig tijd voor een nieuw architectuur. • Dit is de makkelijkste oplossing om op dit moment uit te voeren.
Gemaakt op	08 Jun 2020
Gemaakt door	Probleem aangekaart door @Yildirim Sengul en @Thijs Baan .

Beslissing 1: Gebruik van account op verschillende clients

3.3.6. Symmetrische Encryptie algorithme specificatie

Wijziging 11 juni: beslissingen toegelicht

Beslissing	Beschrijving
Probleem/Issue	Voor het gebruik van een symmetrische encryptie moet een algorithme geselecteerd worden
Beslissing	AES/CBC/PKCS7Padding
Alternatieven	?
Argumenten	?
Gemaakt op	?
Gemaakt door	?

Beslissing 2: Symmetrische encryptie algorithme specificatie

Beslissing	Beschrijving
Probleem/Issue	Voor het gebruik van een asymmetrische encryptie moet een algorithme geselecteerd worden
Beslissing	RSA/ECB/OAEPWithSHA-256AndMGF1Padding

Beslissing	Beschrijving
Alternatieven	?
Argumenten	?
Gemaakt op	?
Gemaakt door	?

Beslissing 3: Asymmetrische encryptie algorithme specificatie

3.4. Algemene hardcoded data afspraken

In dit hoofdstuk staan de afspraken over de algemene hardcoded afspraken. Deze zijn vooral voor gebruik bij testen van de applicatie.

3.4.1. Login credentials

De login credentials kunnen tijdens de ontwikkeling worden gebruikt om in te kunnen loggen op de server

Login Credentials desktop

```
Username: chatter1
Password: welkom01


Username: keeshond
Password: catlover123


Username: hackerman
Password: htmlIsProgramming
```

Code block 1: Login Credentials

3.5. Known bugs/ issues

Wijziging 11 jun: Een known issue omtrent het gebruikt van een vaste initialization vector voor symmetrische encryptie

Tijdens de ontwikkeling van de applicatie kunnen er bugs en issues gevonden worden die buiten de taak liggen waar je nu aan werkt. Het kan zijn dat er in de huidige iteratie geen tijd is om aan deze bugs/ issues te werken. Vandaar dat er dit hoofdstuk is opgesteld waar known bugs/ issues gedocumenteerd kunnen worden. Deze kunnen in de toekomst opgepakt worden.

3.5.1. Beveiliging Chatter Directory Server

Bug/issue	Beschrijving
Toelichting	De Chatter Directory Server is publiekelijk bereikbaar. Dit is een probleem voor de veiligheid.
Gemeld op	26 May 2020
Gemeld door	@Yuri Ruler

Tabel 7: Known bug/issue: Beveiling van Chatter Directory Server

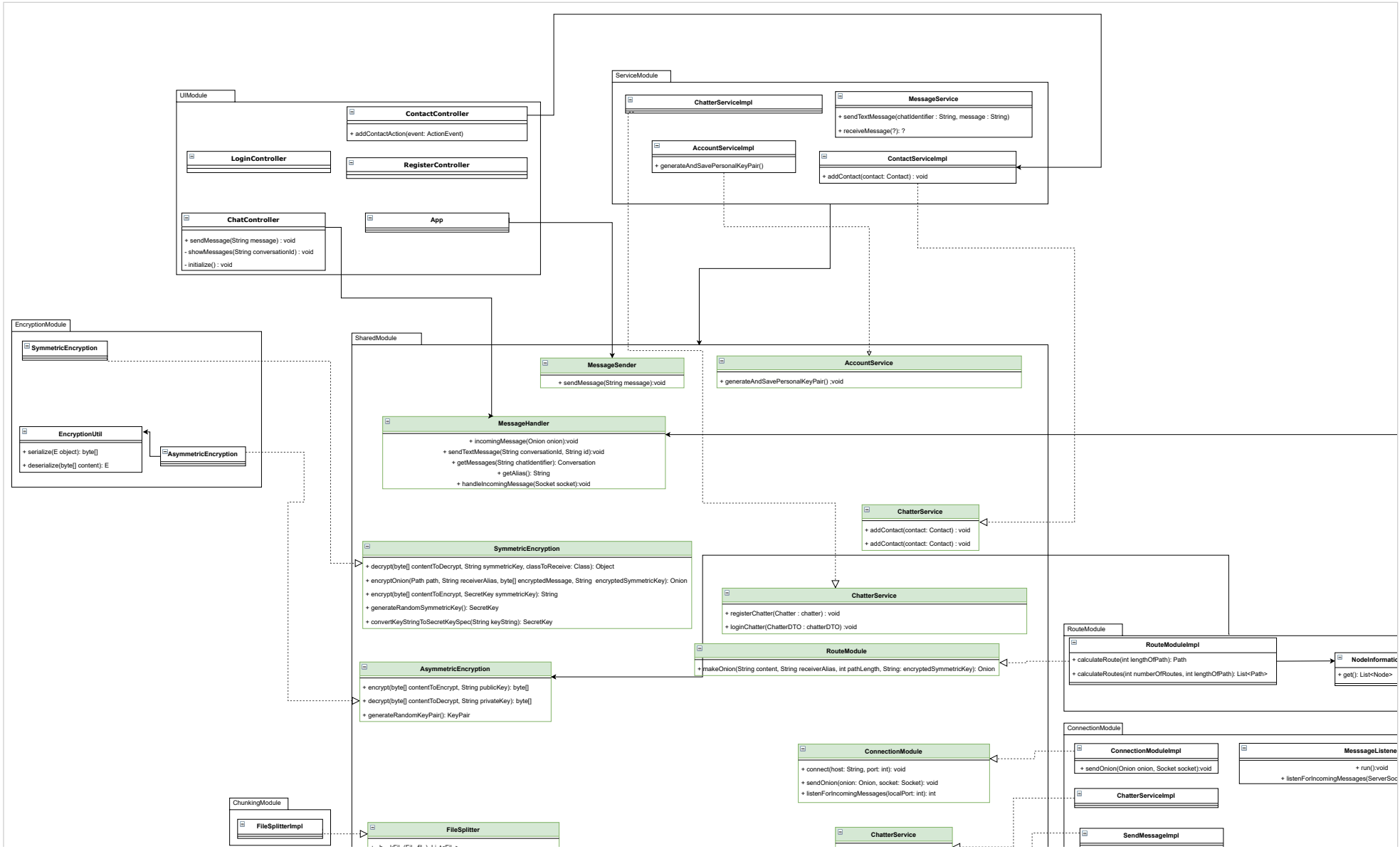
4. Design Desktop Client

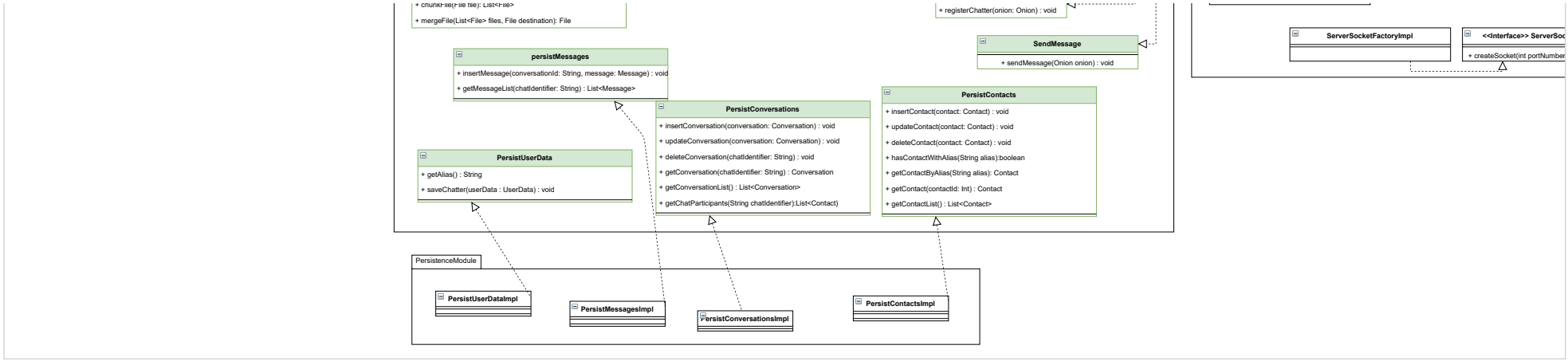
Dit subsysteem houdt de applicatie in die beschikbaar wordt gesteld op de desktop devices, lees macOS, Ubuntu en Windows. Het Software Architecture Document schrijft het gebruik van de programmeertaal Java in combinatie met het JavaFX framework voor.

4.1. Design Class Diagram

Onderstaand Design class diagram geeft een overzicht van alle Modules, interfaces & klassen. Alleen de publieke methodes en variabelen of methodes die veel logica bevatten zijn hierin opgenomen. dependencies tussen klassen

Wijziging 11 Juni: Class diagram consistent gemaakt met de code





Figuur 17: Design class diagram Ontwerp

- Controllermodule is veranderd in ServiceModule, en bevat een klein beetje logica. De services hierin zijn specialisten op een bepaald onderdeel.

4.2. Sequence Diagrams

In dit hoofdstuk wordt aangegeven hoe verschillende functionaliteit, die voortkomt uit de use cases precies in elkaar steekt. Er wordt weergegeven hoe verschillende functies, klassen en modules onderling met elkaar communiceren zodat er een duidelijk beeld ontstaat van de werking van complexe delen in de applicatie.

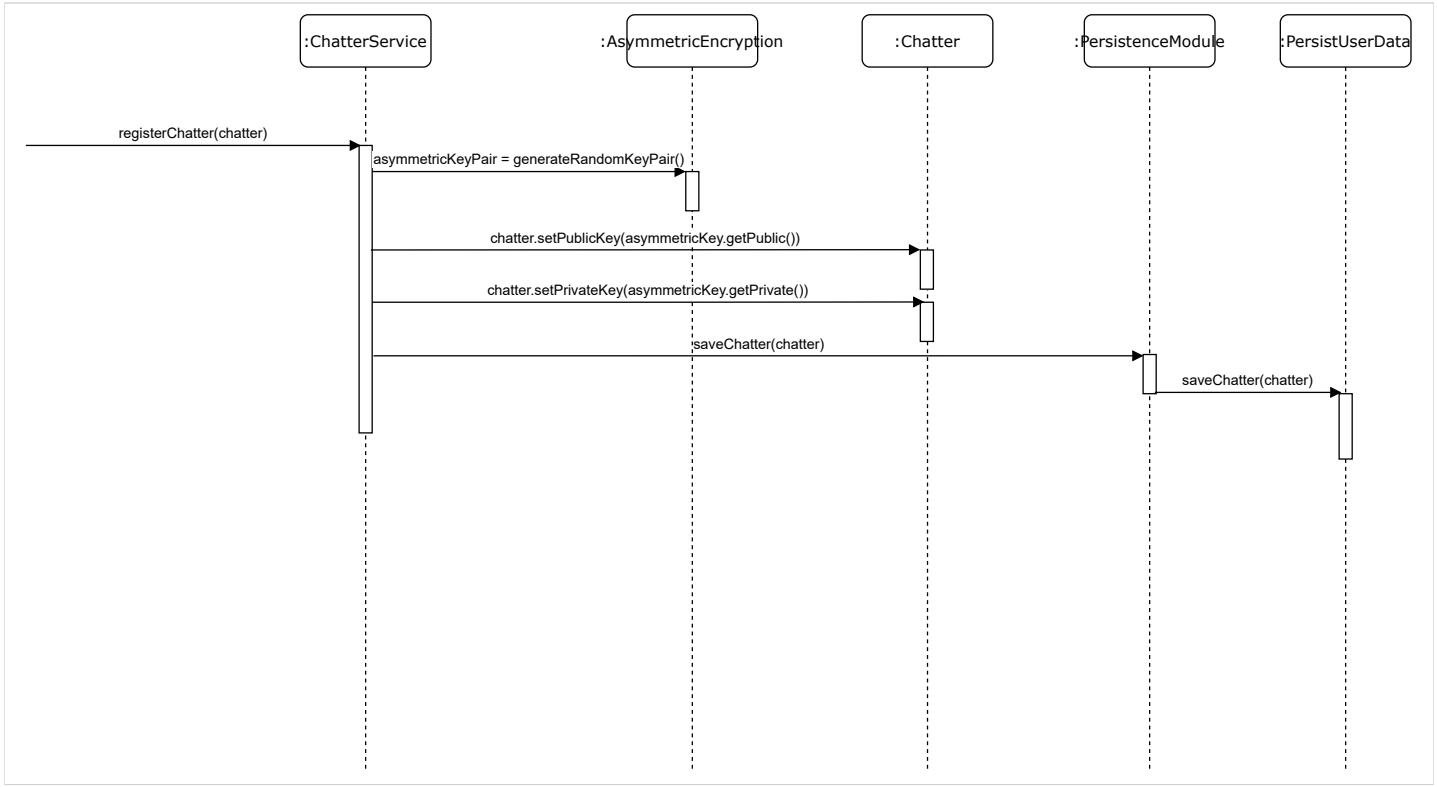
4.2.1. UC 1 Manage account

De globale use case Manageaccount bestaat uit functionaliteiten zoals het create account, view account details en update account. Onderstaande sequence diagrammen gaan in op de precieze werking van deze onderdelen.

Registreren gebruiker

Wijziging 11 juni: deze diagram up-to-date gemaakt met registeren van de code

Een gebruiker maakt een account aan. Na het opgeven van benodigde informatie in een chatter object worden de gegevens gevalideerd en opgeslagen.

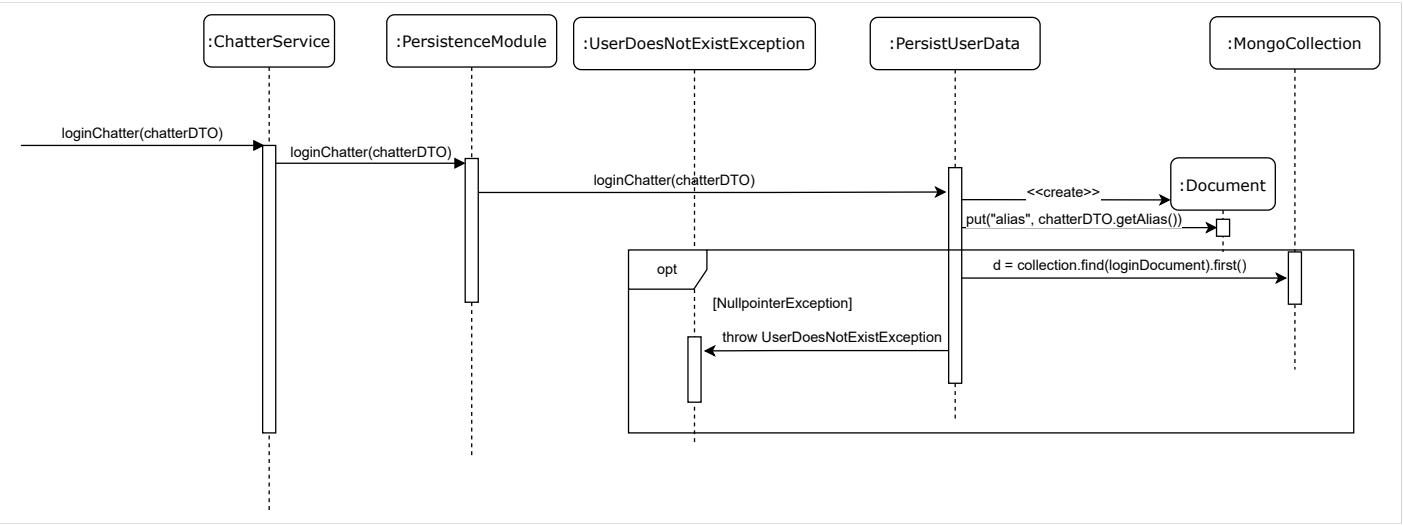


Figuur 18: Registreren Gebruiker

Inloggen gebruiker

Wijziging 11 juni: deze diagram up-to-date gemaakt met inloggen van de code

Onderstaande sequence diagram laat het proces zien wanneer een gebruiker zich aanmeld in het systeem.



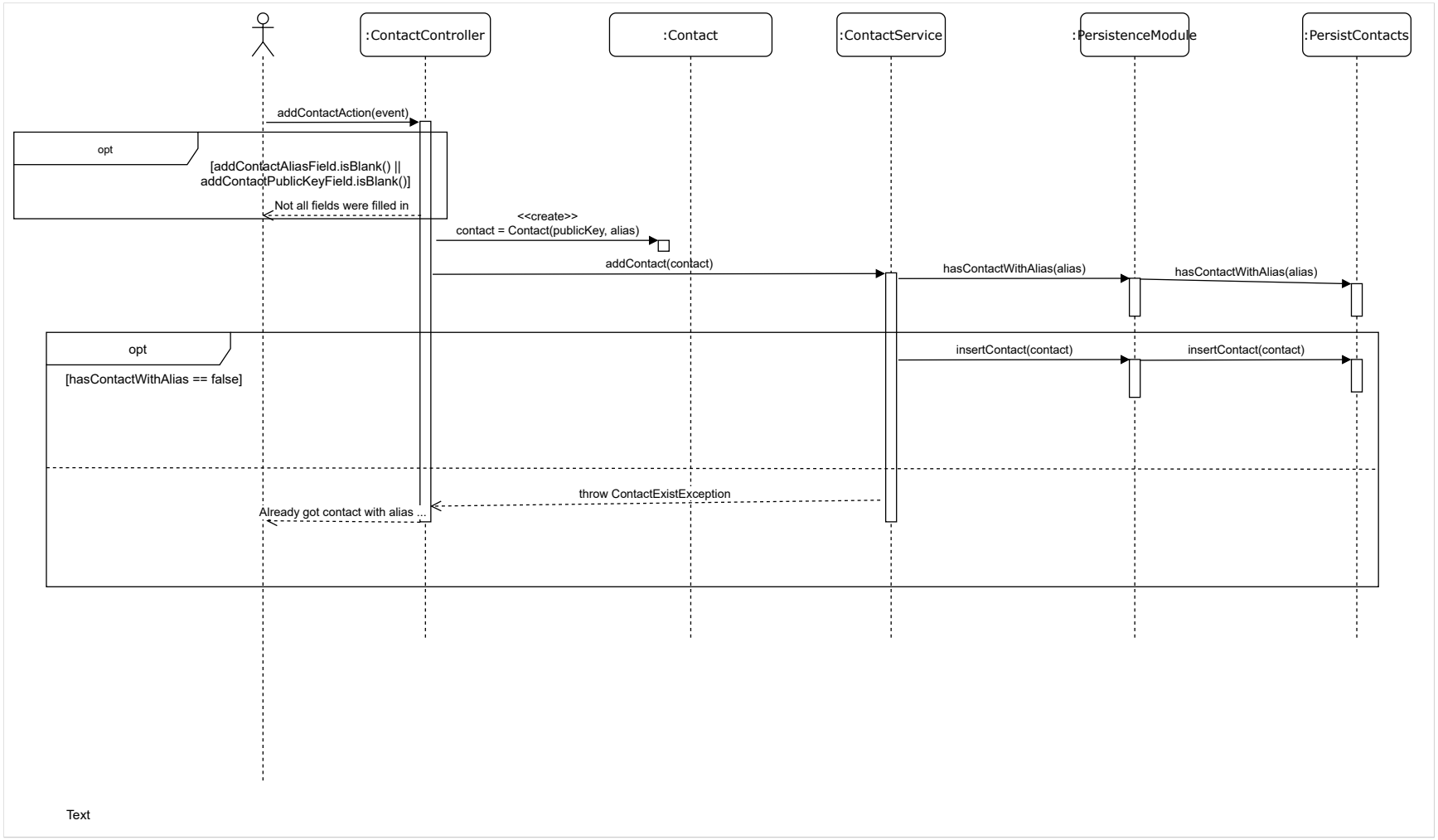
Sequence Diagram 1: Inloggen gebruiker

4.2.2. UC 2 Manage contact

De Use case manage contact bestaat uit de volgende functionaliteiten: add contact, view contact en delete contact. Bij add contact is het resultaat gerealiseerd en daar is dan ook een Sequence diagram voor gemaakt

Add contact

Een chatter voegt een andere chatter toe aan zijn contacten. Deze creeert een contact in de contactcontroller en insert deze in de database, wanneer het contact bestaat zal er een foutmelding optreden.



Figuur 19: Sequence diagram Contact toevoegen

4.2.3. UC 4 Exchange Message

De use case exchange message betsaat uit de volgende onderdelen: Send message (en ontvangen) en Receive offline message. Send message is veruit de grootste UC en bestaat uit een grote hoeveelheid complexe code zoals het maken van een message, creeeren van de onion en het

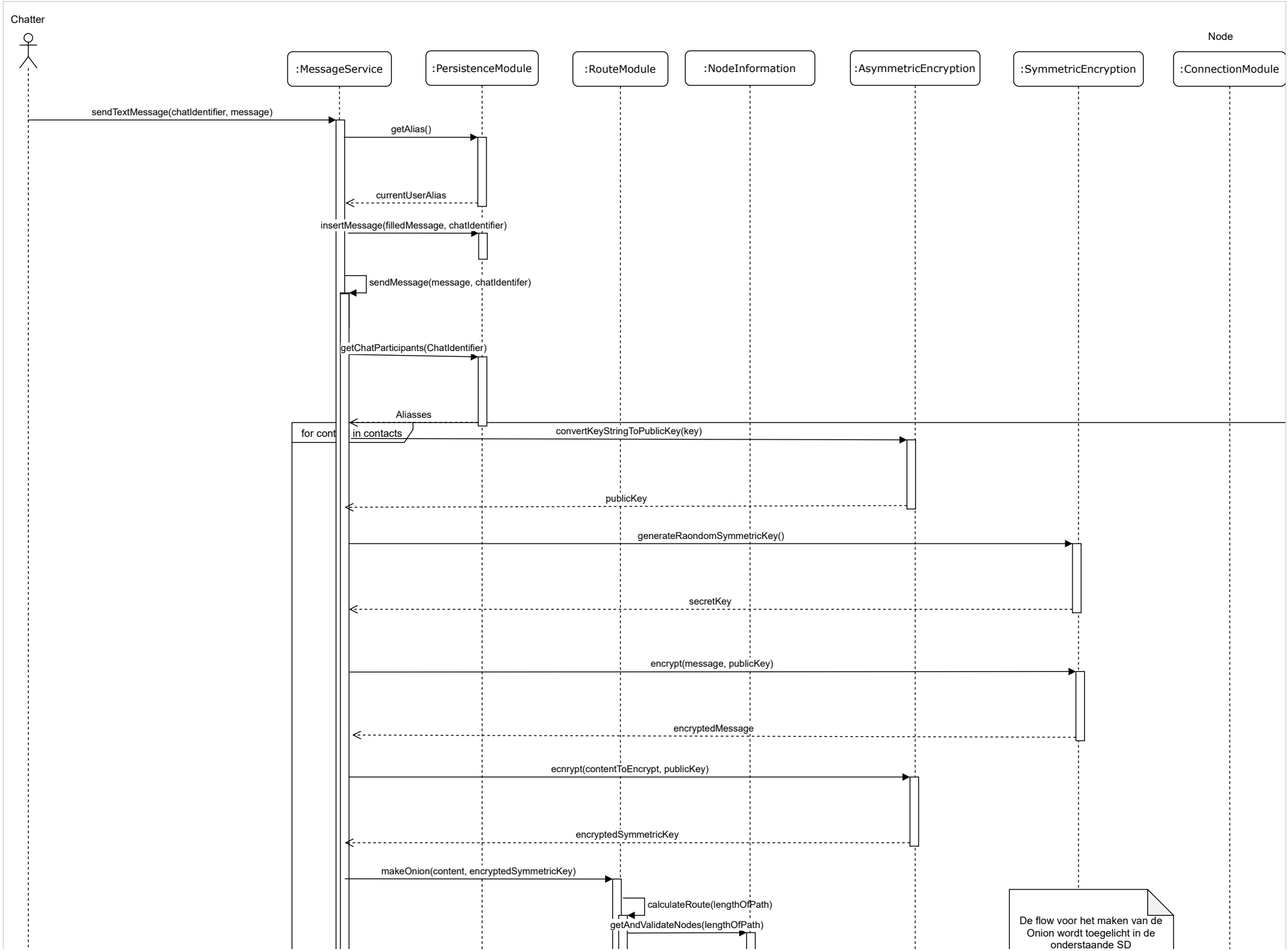
Bericht versturen

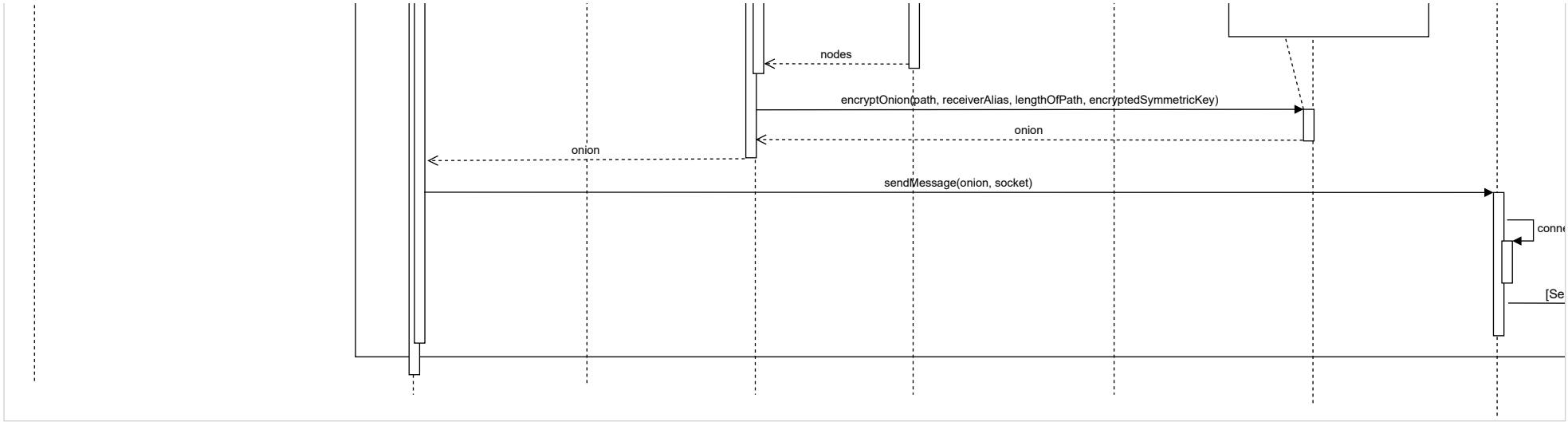
Onderstaande sequence diagrammen geven een inzicht in alle stappen die komen kijken bij het versturen van een bericht, Zo wordt er ingegaan op het een globaal overzicht en het toevoegen van de onion lagen

Globaal

Wijziging 11 juni: rechtgetrokken met de code

Figuur 20 illustreert welke stappen er worden gezet om een bericht te versturen. De actor "chatter" is hier zowel de persoon als de UI. De UI zal kennis hebben van bijvoorbeeld de conversationId. De chatter verstuurt het bericht. Deze wordt door de ControllerModule opgeslagen en versleuteld. Vervolgens wordt het bericht door RouteModule ingepakt in een onion om verstuurd te kunnen worden door de ConnectionModule.

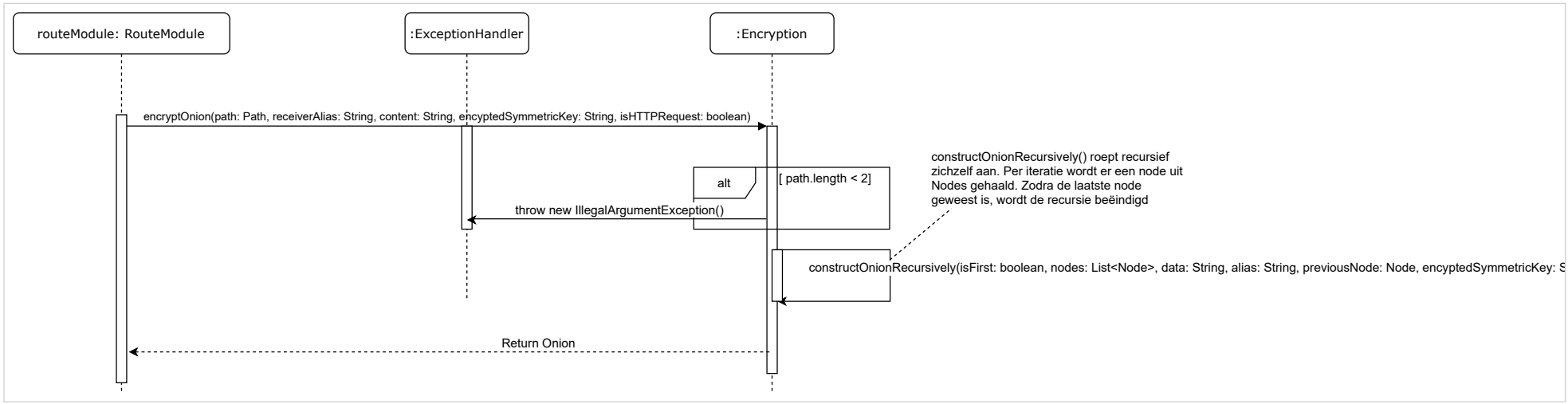

Chatter
:MessageService
:PersistenceModule
:RouteModule
:NodeInformation
:AsymmetricEncryption
:SymmetricEncryption
Node
:ConnectionModule
sendTextMessage(chatIdentifier, message)
getAlias()
currentUserAlias
insertMessage(filledMessage, chatIdentifier)
sendMessage(message, chatIdentifer)
getChatParticipants(ChatIdentifier)
Aliasses
for cont
in contacts
convertKeyStringToPublicKey(key)
publicKey
generateRaondomSymmetricKey()
secretKey
encrypt(message, publicKey)
encryptedMessage
ecnrypt(contentToEncrypt, publicKey)
encryptedSymmetricKey
makeOnion(content, encryptedSymmetricKey)
calculateRoute(lengthOfPath)
getAndValidateNodes(lengthOfPath)
De flow voor het maken van de Onion wordt toegelicht in de onderstaande SD



Figuur 20: SD: Berichten versturen

Onion laag toevoegen

De onion wordt recursief gemaakt. Aan de hand van de nodes uit de path wordt de onion geconstructueerd en zal de inhoudelijke data versleuteld zijn. De flow is te zien in Figuur 21.



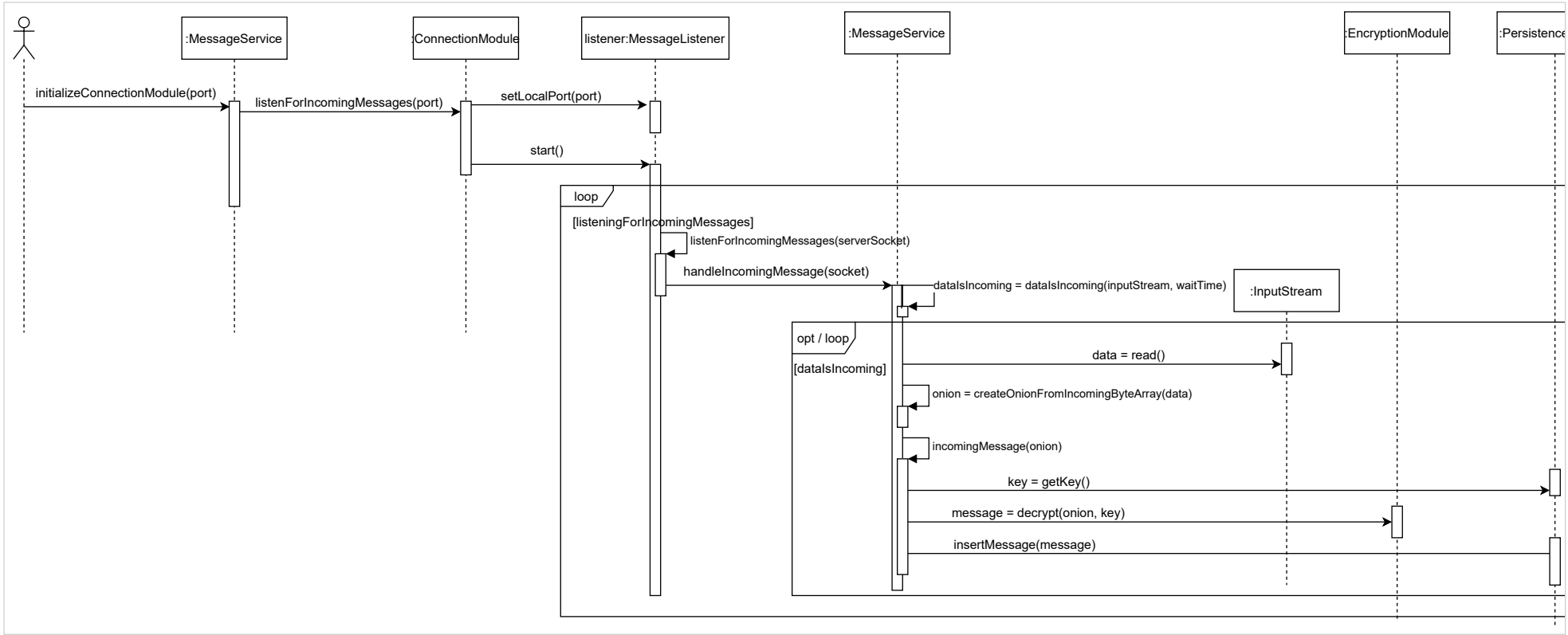
Figuur 21: SD: Onion lagen toevoegen

Ontvangen bericht

Wijziging 11 juni: rechtgetrokken met de code

Het luisteren naar binnenkomende berichten wordt uitgevoerd op een aparte thread. Hierdoor hoeft de rest van het programma niet te wachten op binnenkomende berichten. Een instantie van een MessageListener heeft een eigen instantie van een MessageService, waardoor er geen conflicten ontstaan met andere interacties met MessageServices.

Binnenkomende connecties worden gemaakt met sockets. Over de socket wordt gelezen of er data in de input stream zit. Zo ja, dan wordt die data omgezet naar een Onion object. De data wordt gedecrypt en omgezet naar een Message object. Deze kan vervolgens opgeslagen worden in de database, zodat die in de UI uiteindelijk uitgelezen kan worden.

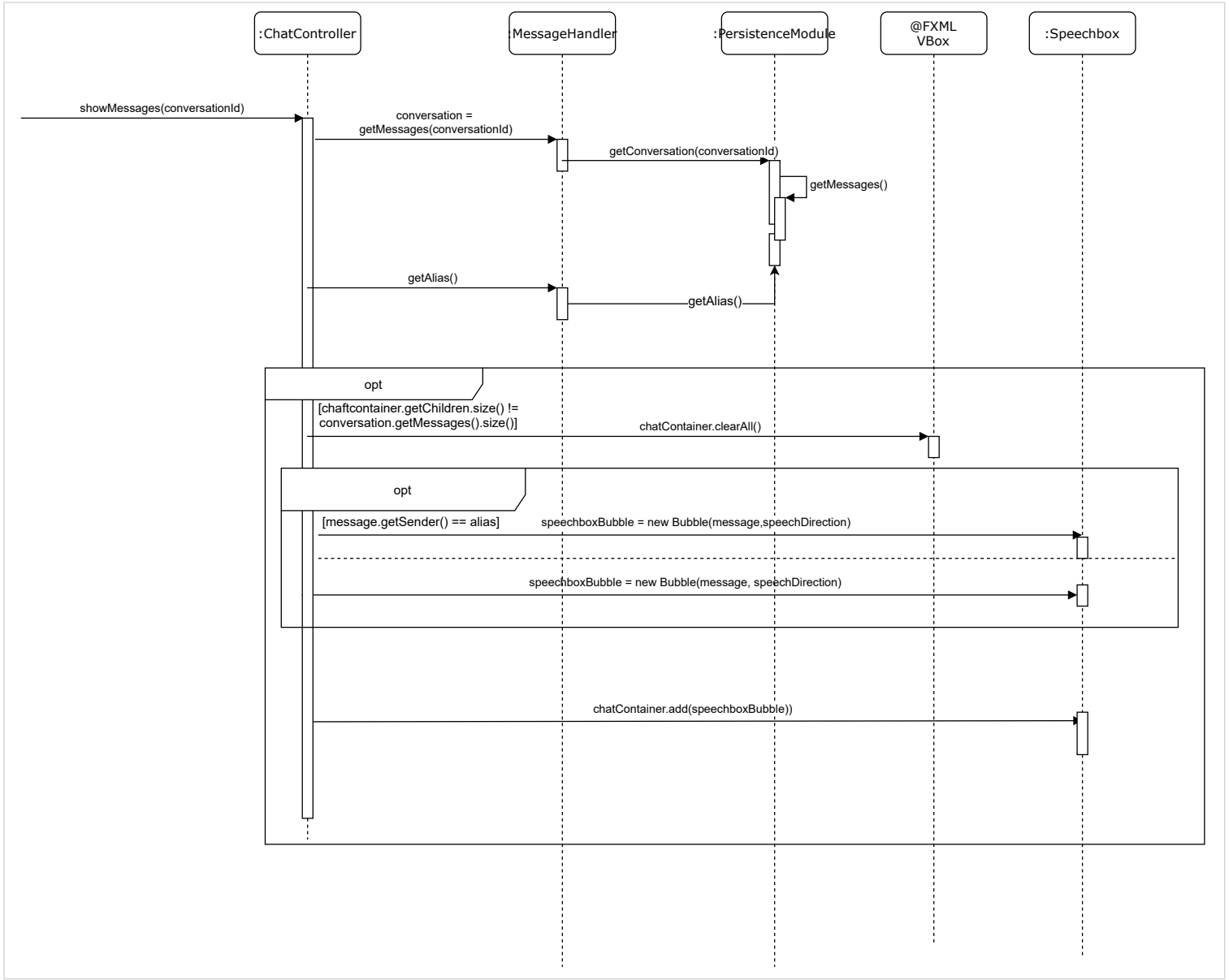


Figuur 22: SD Bericht ontvangen

Bericht weergeven

Wijziging 11 juni: actueel getrokken met de code

Onderstaand sequence diagram laat zien hoe een bericht wordt weergegeven, dit gebeurt momenteel door iedere seconde een functie aan te roepen die de UI doet updaten. In het eindproduct is het echter goed om een observer pattern toe te passen, zodat binnenkomende berichten automatisch getoond worden.



Sequence Diagram 2: Bericht weergeven

4.3. Databasemodel

Er word gebruik gemaakt van een NoSQL database, dit betekent dat de database niet een strakke structuur volgt. Om ervoor te zorgen dat toch voor iedereen duidelijk is wat waar is opgeslagen is hier een datamodel voor aangemaakt. Een dataobject bestaat uit de naam van de collectie en de opmaak van de JSON.

4.3.1. conversation

Onderstaande Json bevat de deelnemers van een gesprek en de messages die erin zitten. De messages kunnen van verschillende types zijn, zoals een textmessage, of een recipemessage

conversation

```
{
  "_id": "CMtrK9nlKTrE1SrxhxyIUKBOD", //Het ID is hetzelfde als de conversationId in de berichten.
  "participants": [
    {
      "alias": "Anja",
      "publicKey": "PublicKey"
    },
    {
      "alias": "Geert",
      "publicKey": "PublicKey"
    }
  ],
  "messages": [
    {
      "messageType": "TEXT",
      "alias": "Anja",
      "conversationId": "CMtrK9nlKTrE1SrxhxyIUKBOD",
      "timeStamp": {
        "$date": "2020-06-05T09:53:24.491Z"
      },
      "message": "Hallo Geert"
    },
    {
      "messageType": "TEXT",
      "alias": "Geert",
      "conversationId": "CMtrK9nlKTrE1SrxhxyIUKBOD",
      "timeStamp": {
        "$date": "2020-06-05T09:53:24.491Z"
      },
      "message": "Goedemiddag Anja"
    }
  ]
}
```

Code 1: Voorbeeld voor een object in 'conversation'

4.3.2. Contacts

Onderstaande Json is bedoeld om de verschillende contacten op te slaan en bestaat uit een lijst met de alias(nickname) en de public key per contact.

```
"contacts": [
    {
      "alias": "TEST1",
      "publicKey": "base64 encoded string"
    },
    {
      "alias": "TEST2",
      "publicKey": "base64 encoded string"
    },
    {
      "alias": "TEST3",
```

```
    "publicKey": "base64 encoded string"
  },
  {
    "alias": "TEST4",
    "publicKey": "base64 encoded string"
  }
]
```

Code 2: Voorbeeld van de inhoud van de contactscollectie

4.4. Ontwerpbeslissingen voor het deelsysteem

Wijziging 11 juni: beslissingen toegevoegd. Meer toelichting gegeven over gebruikte patterns

Tijdens het ontwerpen van de applicatie zijn er een aantal beslissingen genomen, sommige met een grotere invloed dan andere. Sommige wijken bijvoorbeeld af van de gewenste manier om iets te programmeren. Of zijn niet meteen herleidbaar naar de documentatie, onderstaande kopjes geven een goed overzicht van de gemaakte beslissingen, waar ze invloed op hebben.

4.4.1. Encrypten van een onion

Beslissing	Beschrijving
Probleem/Issue	Een bericht moet even vaak worden encrypt als het aantal nodes dat er zijn. Dit is lastig te implementeren omdat de lengte van de nodes niet vastgesteld is.
Beslissing	Er is ervoor gekozen om de methode encryptOnion een ander methode te laten oproepen, namelijk de methode constructOnionRecursively. Deze methode is een recursieve methode die een bericht met alle lagen van de nodes encrypt.
Alternatieven	Een alternatief zou het maken van een for loop geweest zijn.
Argumenten	Recursie is een goed algoritme om dit soort problemen op te lossen. Een andere reden om deze techniek te gebruiken is zodat Yuri competentie ASD P-5 kon aantonen.
Gemaakt op	27 May 2020
Gemaakt door	@ Yuri Ruler

Tabel 8: Beslissing: 'Symmetrische encryptie' Onion Lagen

4.4.2. Encryptie input en output

Beslissing	Beschrijving
Probleem/Issue	De 'encrypt' functie vraagt om een byte[] als input, dit betekent dat iedere functie die de 'encrypt' aanroept het te encrypten object eerst om moet zetten naar een byte[]. De 'decrypt' functie geeft juist een byte[] terug, wat er dus voor zorgt dat de ontvanger het zelf om moet zetten naar een gewenst formaat.
Beslissing	De 'encrypt' functies vragen nu om een generiek 'Object' als input, en zetten het vervolgens om tot een byte[], de 'decrypt' maakt gebruik van een 'Generic' die wordt aangegeven bij het aanmaken/importeren van de interface. Dit bepaald in welk formaat de 'decrypt' zijn resultaten teruggeeft.
Alternatieven	Het laten zoals het was, met een byte[] als input voor 'encrypt' en als output voor 'decrypt'
Argumenten	Het is goed om de cohesie omlaag te brengen. De verantwoordelijkheid voor het omzetten naar een byte[] ligt nu bij de klasse die het ook daadwerkelijk gaat gebruiken. Daarnaast is het een stuk makkelijker voor de programmeur om het te implemetneren op deze manier.
Gemaakt op	?
Gemaakt door	@ Dennis Hakvoort

Tabel 9: Beslissing: 'encryptie input en output'

4.4.3. Format meegeven aan sendmessage

Designkeuze	Beschrijving
Probleem/Issue	De 'sendMessage' functie in de messageService heeft een aantal parameters nodig als input. Waaronder het te versturen bericht.

Designkeuze	Beschrijving
Beslissing	Hij krijgt als input een implementatie van de klasse 'Message'. Voor de verschillende message implementaties worden aparte functies gemaakt om de verschillende soorten berichten af te handelen.
Alternatieven	Een aantal nodige variabelen meegeven voor de verschillende soorten bericht dat verstuurd word.
Argumenten	Er zijn twee soorten berichten die verstuurd kunnen worden in het systeem: • een tekstbericht • een bestand Beide vereisen een andere input, en ook iets andere logica. In eerste instantie was dit allemaal in dezelfde functie te vinden, dit is nu verspreid over meerdere functies.
Gemaakt op	?
Gemaakt door	@Dennis Hakvoort

Tabel 10: Beslissing: Format meegeven aan 'sendmessage'

4.4.4. Opsplitsing van symmetrische en asymmetrische encryptie

Designkeuze	Beschrijving
Probleem/Issue	De encryptie, decryptie, keygeneration en encryptionmodule classen hebben methoden met dezelfde naam en met andere parameters. Bijvoorbeeld in de encryption classe de methoden encrypt(data, key) en encrypt(data, publicKey). Deze methoden zijn bedoeld voor andere doeleinden en moeten niet met elkaar verward worden.
Beslissing	De encryptie, decryptie en keygeneration klassen worden samengevoegd en op een andere manier verdeeld. Nu met een klasse voor asymmetrische encryptie en een klasse voor symmetrische encryptie. Deze klassen hebben beide methoden voor encrypt, decrypt en key generation. De encryption module wordt verwijderd.
Alternatieven	Deze klassen laten staan zoals ze eerder waren opgedeeld.
Argumenten	• De bijbehorende methoden zijn nu bij elkaar gebundeld. De RSA-keypair heb je nodig voor RSA-encryptie, voor decryptie moet je de bijbehorende RSA-private key gebruiken, en ook decrypten via RSA. • Asymmetrische encryptie gebruik je voor andere doeleinden dan symmetrisch, met ook verschillende voor- en nadelen. Deze duidelijk opsplitsen maakt het duidelijker waar je mee bezig bent en welke methode je nodig hebt.
Gemaakt op	19 May 2020
Gemaakt door	@Gino vd Bund

Tabel 11: Beslissing: Opsplitsen symmetrische en assymetrische encryptie

4.4.5. Encryptie bericht

Beslissing	Beschrijving
Probleem/Issue	Het bericht versleutelen op een asymmetrische manier staat maar toe dat er maximaal 190 bytes encrypt kunnen worden met een 2048 byte lange asymmetrische sleutel.
Beslissing	De klasse 'ClientDestination' wordt uitgebreid met een extra veld waar een een asymmetrisch encrypted, symmetrische sleutel in word opgeslagen. De inhoud van het bericht wordt hierdoor symmetrisch encrypt, maar om de sleutel te krijgen is wel de asymetrische (privé) sleutel nodig.
Alternatieven	• De asymmetrische sleutel kan langer worden gemaakt. Hoe langer de sleutel, hoe meer content er versleuteld kan worden. Voor iedere byte die je extra wil versleutelen, moet de sleutel 8 bytes langer worden. • Het bericht wordt versleuteld op een symmetrische manier, maar de symmetrische sleutel wordt op een asymmetrische manier gedeeld met de ontvanger voordat het bericht wordt verstuurd.
Argumenten	• Bestanden kunnen niet worden versleuteld op een asymmetrische manier. • Er is niet veel tijd meer, en we willen zo weinig mogelijk invloed hebben op andere componenten. Door de symmetrische sleutel door te sturen als onderdeel van de destination wordt hij pas zichtbaar in de onderste laag van de onion, dus bij de exit node en de client. Daarnaast hoeft er op deze manier niet heel veel te veranderen in de implementatie van de drie componenten.
Gemaakt op	?
Gemaakt door	Probleem aangekaart door @Dennis Hakvoort , oplossing bedacht in een discussie met @Tim Hemmes , @Gino vd Bund en @Dennis Hakvoort Tabel 12.

Tabel 12: Beslissing: encryptie bericht

4.4.6. Gebruik database bij uitvoering testen

Beslissing	Beschrijving
Probleem/Issue	De database query's moeten getest worden met een database. De vraag is met welke database er getest wordt.
Beslissing	Er is gekozen om met een replica van de productiedatabase te testen. Dit houdt in dat de database properties hetzelfde zijn op de database naam na. Het overschrijven van de properties is gedaan door middel van een annotatie in de tests.
Alternatieven	• Het testen op een in memory database • Het testen op de productiedatabase
Argumenten	• Het is niet verstandig om op de productiedatabase te testen. • De in memory database verschilt sterk van de Mongo database. Er kan dus beter met een Mongo database getest worden.
Gemaakt op	26 May 2020 .
Gemaakt door	@ Yuri Ruler (voor desktop @ Jelmer Wijnja)

Tabel 13: Beslissing: Gebruik database bij uitvoering testen

4.4.7. Test falen als een resource niet gesloten is

Beslissing	Beschrijving
Probleem/Issue	De tests voor de ChunkingModule lieten een hoop bestanden achter na het uitvoeren van de tests, welke niet door het testframework verwijderd konden worden. Dit bleek te komen doordat er gebruik werdt gemaakt van de RandomAccessFile klasse. Dit is een resource stream, wat betekend dat de stream na het gebruik gesloten moest worden. Omdat dit niet werd gedaan, bleef de file open totdat Java klaar was met ALLE tests. De code welke de bestanden moest verwijderen was al lang uitgevoerd(zonder succes).
Beslissing	Er is gekozen om alle tests van de ChunkingModule te laten falen, middels een fail in de tearDown functie. Hierdoor wordt duidelijk dat er nog ergens resources open staan
Alternatieven	• Niks doen en het risico lopen dat het nog eens gebeurt
Argumenten	• Na het testen wordt er netjes opgeruimd • Slordigheid met resources wordt afgestraft
Gemaakt op	05 Jun 2020 .
Gemaakt door	@ Jelmer Wijnja

Tabel 14: Beslissing: Test laten falen als een resource niet gesloten is

4.4.8. Ophalen berichten

Beslissing	Beschrijving
Probleem/Issue	Het ophalen van berichten verloopt vanuit de UIModule naar de ServiceModule, bij het binnenkomen van een bericht zou de servicemodule het liefst de UIModule updaten, daardoor ontstaat echter een circulaire dependency.
Beslissing	Er is voor gekozen om iedere seconde de functie getMessages aan te roepen
Alternatieven	Observer Pattern toepassen
Argumenten	Een Observer Pattern zou een mooiere oplossing zijn. Deze wordt op dit moment echter niet toegepast omwille van tijd.
Gemaakt op	08 Jun 2020
Gemaakt door	@ Sven Molhuijsen

Tabel 15: Beslissing: Test laten falen als een resource niet gesloten is

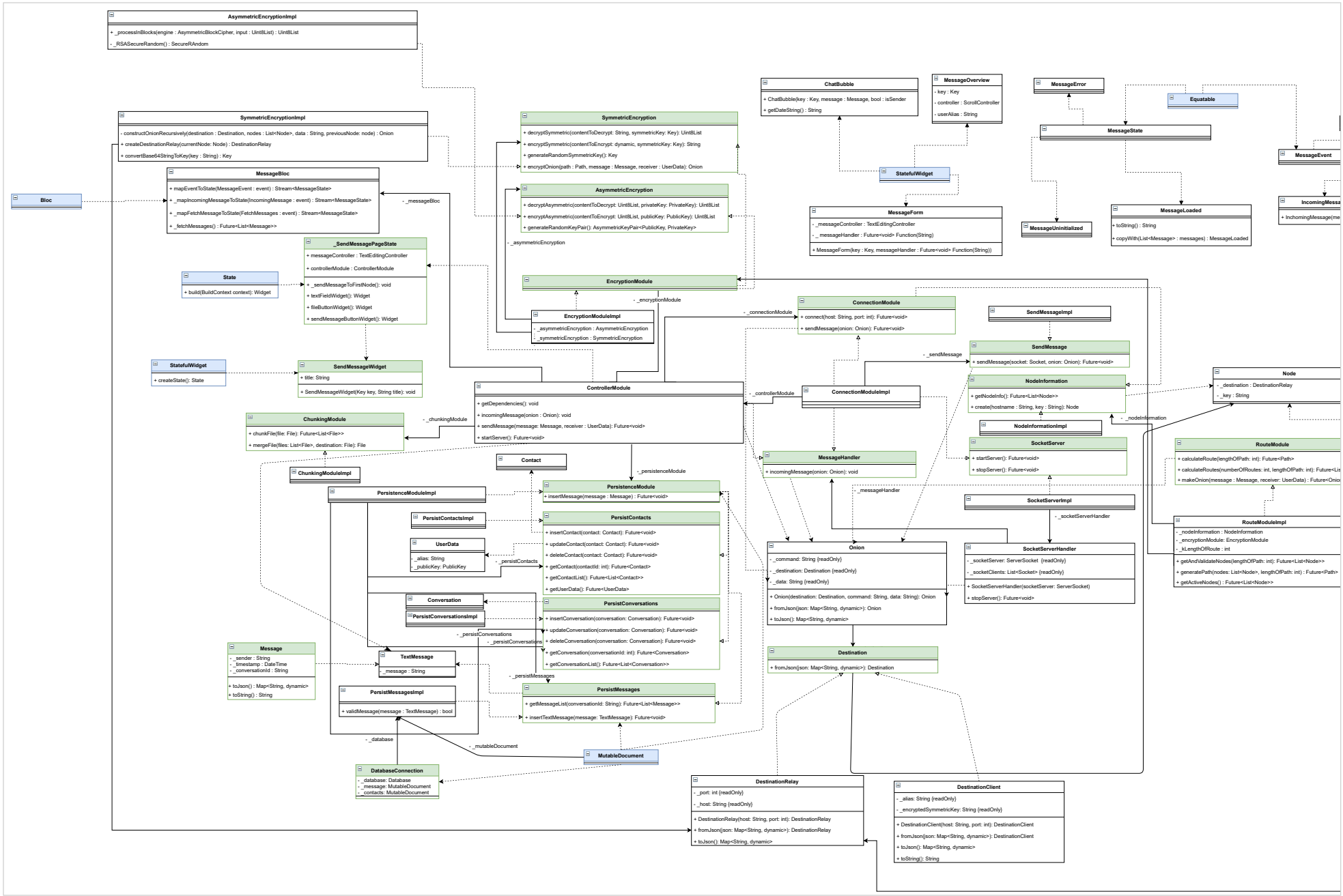
5. Design Mobile Client

Dit subsysteem houdt de applicatie in die beschikbaar wordt gesteld op de mobile devices, lees Android en iOS. Het Software Architecture Document schrijft het gebruik van de programmeertaal Dart in combinatie met het Flutter framework voor.

5.1. Design Class Diagram

De functionaliteiten en implementatie van de mobile client komen vrijwel overeen met die van de Desktop Client. De verschillen worden gedocumenteerd.

Wijziging 11 juni: Gehele Design Class Diagram bijgewerkt aan de laatste code in de Mobile Client en gewijzigd zodat het voldoet aan de Documentatie Styleguide SRS/SDD, door @Roel van de Wiel . Verder ook package diagram toegevoegd ter verduidelijking en ook toelichting naar 2/3 patterns.



Figuur 23: Design Class Diagram

Er is besloten om de Injectorclass niet op te nemen in het Design Class Diagram. Dit om de leesbaarheid van het diagram te verhogen.

5.2. Sequence Diagrams

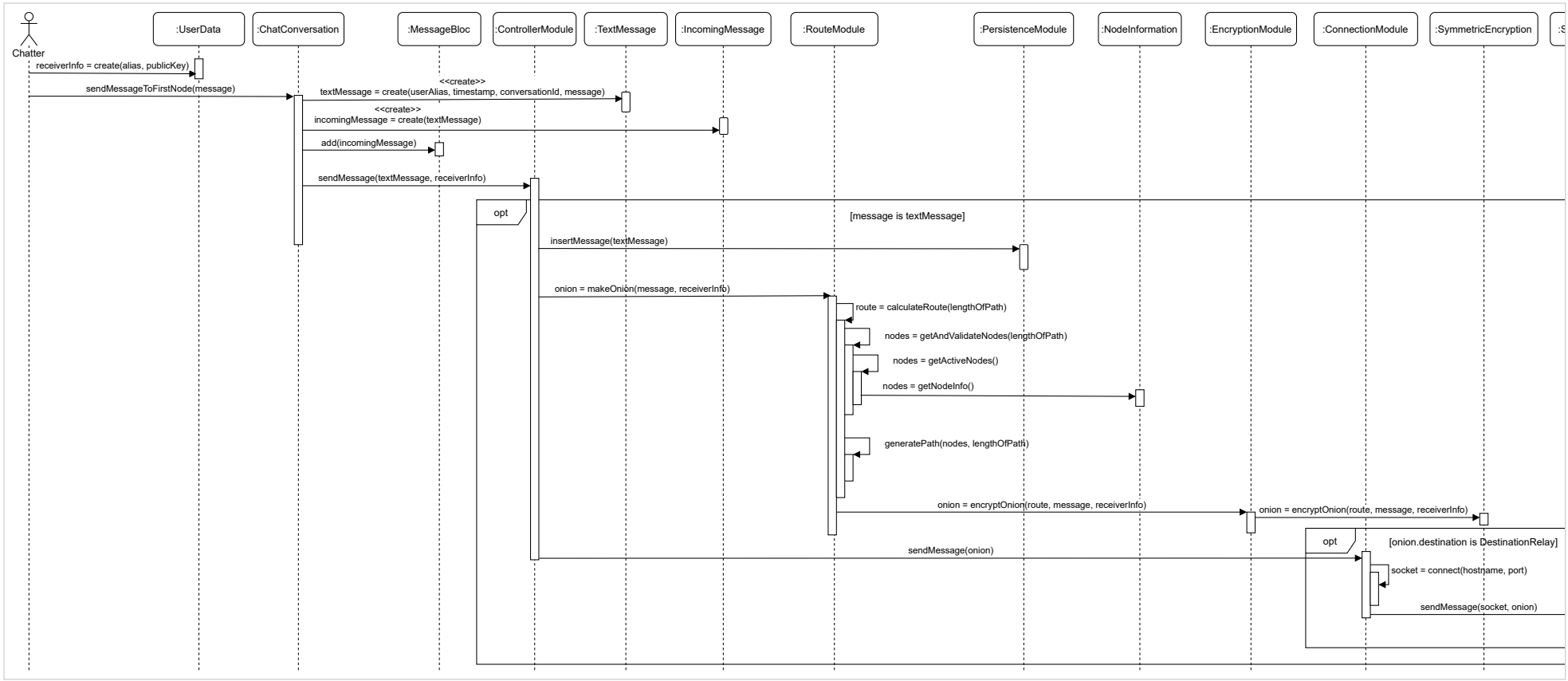
Wijziging 11 juni: Alle sequence diagrammen bijgewerkt aan de laatste code in de Mobile Client en gewijzigd zodat het voldoet aan de Documentatie Styleguide SRS/SDD, door @Vu Le .

5.2.1. UC Exchange Message

Bericht verzenden

<Inleiding, voor inspiratie → zie desktop client>

Globaal



Figuur 24: SD: Globaal - bericht verzenden

Encryptie

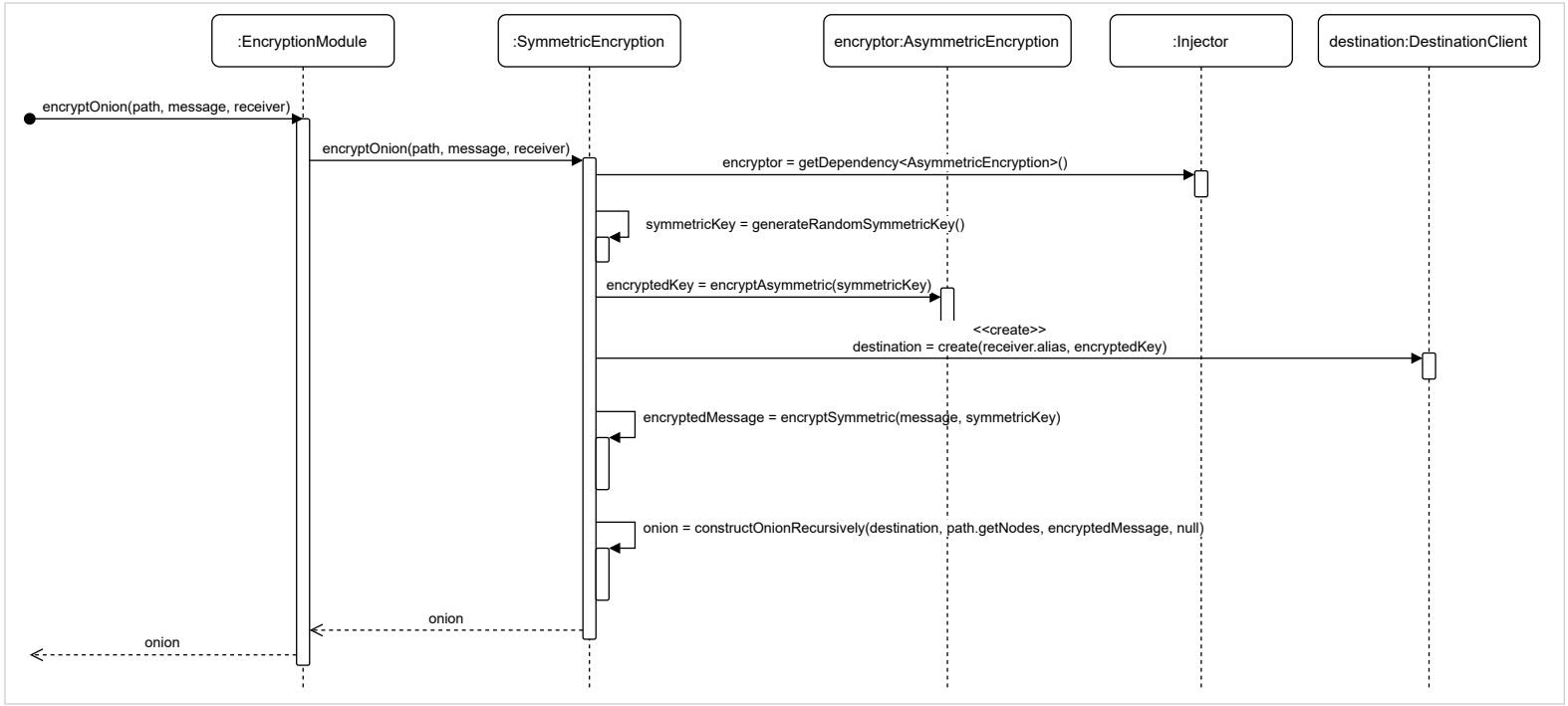
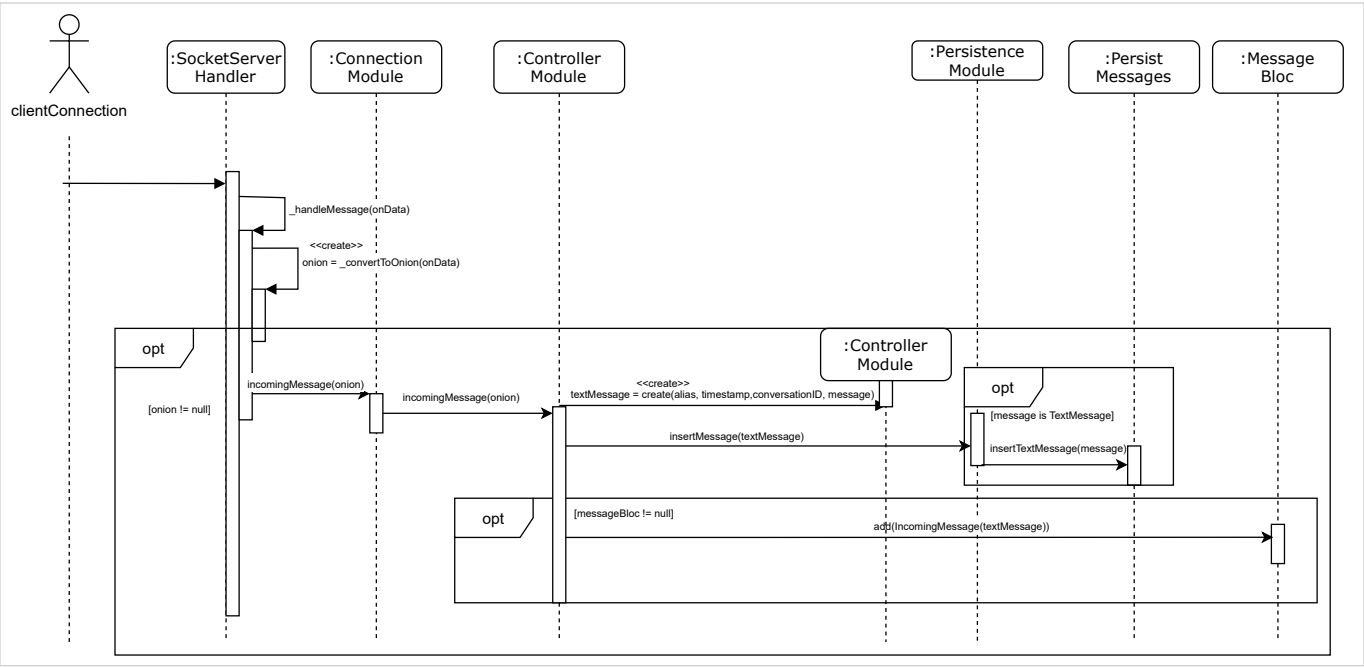


Figure 1: AD: Encryptie

Bericht ontvangen

Globaal



Figuur 25: SD: Bericht ontvangen

5.3. Ontwerpbeslissingen voor het deelsysteem

Wijziging 11 juni: twee ontwerpbeslissingen toegevoegd: abstracte klasses als interface (door @Tim Hemmes , @Roel van de Wiel en @Vu Le) en asymmetrische encryptie (@Mark Ogink)

5.3.1. Widget State Management

Beslissing	Beschrijving
Probleem/Issue	De state moet bijgehouden kunnen worden in de Flutter Widgets. Het moet ook mogelijk zijn om de state te updaten.
Beslissing	Er is gekozen om gebruik te maken van het BLoC (Business Logic Components) Pattern. Hiervoor wordt gebruik gemaakt van de 'flutter_bloc' dependency.
Alternatieven	• Data (state) direct in de widget opslaan en updaten • Redux
Argumenten	• Het Bloc pattern wordt aangeraden door de Flutter developers • State direct in de widget opslaan en updaten werkt enkel met een simpele state, die enkel binnen die widget wordt gebruikt. • Redux is te uitgebreid met veel boilerplate code en heeft een hoge leercurve.
Gemaakt op	19 May 2020
Gemaakt door	@Tim Hemmes

Tabel 16: Beslissing: State Management Mobile Applicatie

5.3.2. Eenmalige getUserData call

Beslissing	Beschrijving
Probleem/Issue	Bij het verzenden van een bericht moet de userdata (=alias) meegestuurd worden. Om te voorkomen dat dit bij elke verzending opnieuw opgehaald moet worden, moet hier een oplossing voor verzonnen worden.

Beslissing	Beschrijving
Beslissing	Er is gekozen voor een 'promise'-call, tijdens de initialisatie van de chat widget.
Alternatieven	• In plaats van een promise call een simpele **await** call.
Argumenten	• Om een await call te kunnen doen, moet de functie asynchrone aanroepen ondersteunen. De initialisatie-functie van de widget ondersteund dit echter niet.
Gemaakt op	02 Jun 2020
Gemaakt door	@ Tim Hemmes , @ Jonathan Vandionant

Tabel 17: Beslssing: Eenmalige getUserData call

6. Design Nodes

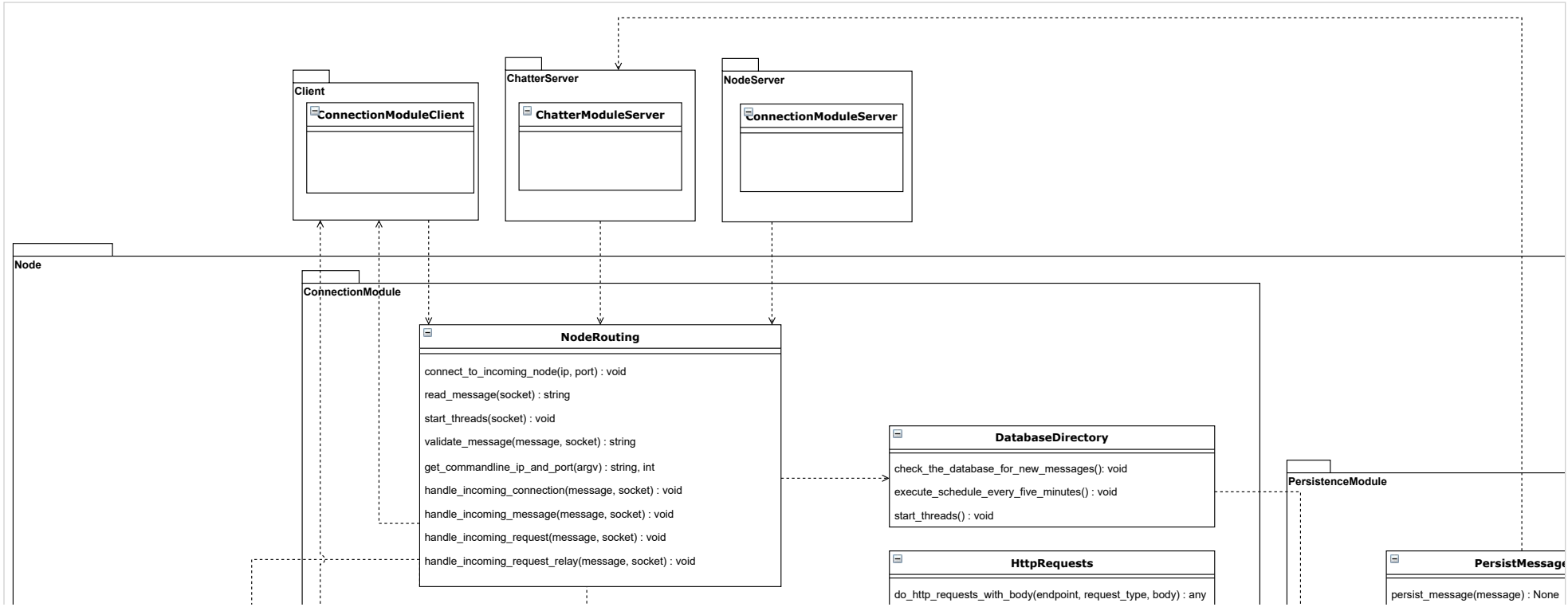
Een node is een systeem dat verantwoordelijk is voor het doorspelen van berichten. Elke node haalt een encryptielaag van het binnengekomen bericht af, kijkt waar het bericht vervolgens naartoe moet en stuurt het vervolgens daarheen. Het Software Architecture Document schrijft het gebruik van de taal Python voor met gebruik van functioneel programmeren.

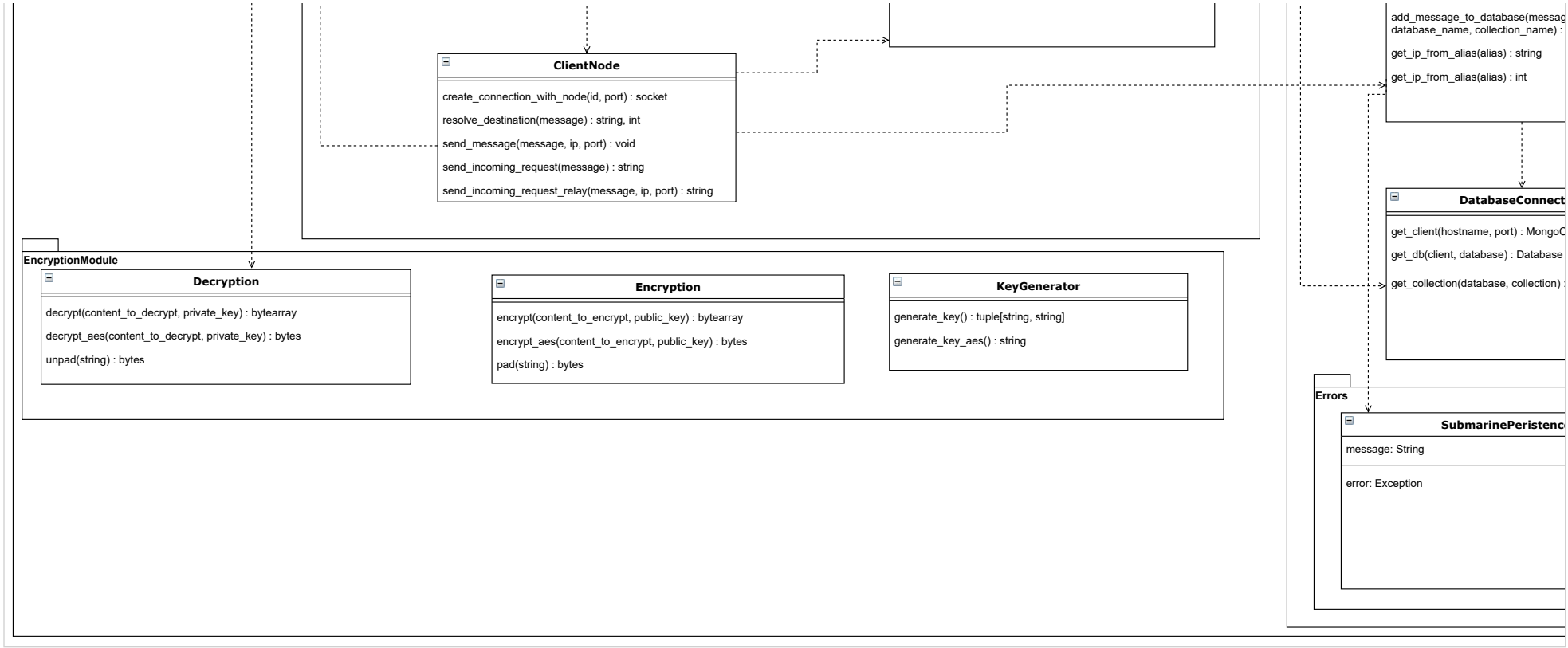
6.1. Design Class Diagram

In dit hoofdstuk worden de modules, Python-bestanden en functies beschreven in de vorm van een class diagram. Hierbij wordt de nadruk gelegd op de modules en de functies, aangezien deze subsectie niet een objectgeoriënteerd systeem beschrijft. Wanneer hier wel variabelen aangegeven zijn, betekent dit dat dit vaste waarden zijn die de functies gebruiken, zoals de host en poort die vaststaan voor een bepaalde instantie van een node.

Wijziging 11 jun: Toelichting aangepast om keuze over design patterns uit te leggen i.v.m functioneel paradigma. Daarnaast is het diagram aangepast om consistent te zijn met code en wat netter gemaakt door lijntjes aan te passen.

6.1.1. Class diagram





Figuur 26: Design class diagram

6.2. Sequence Diagrams

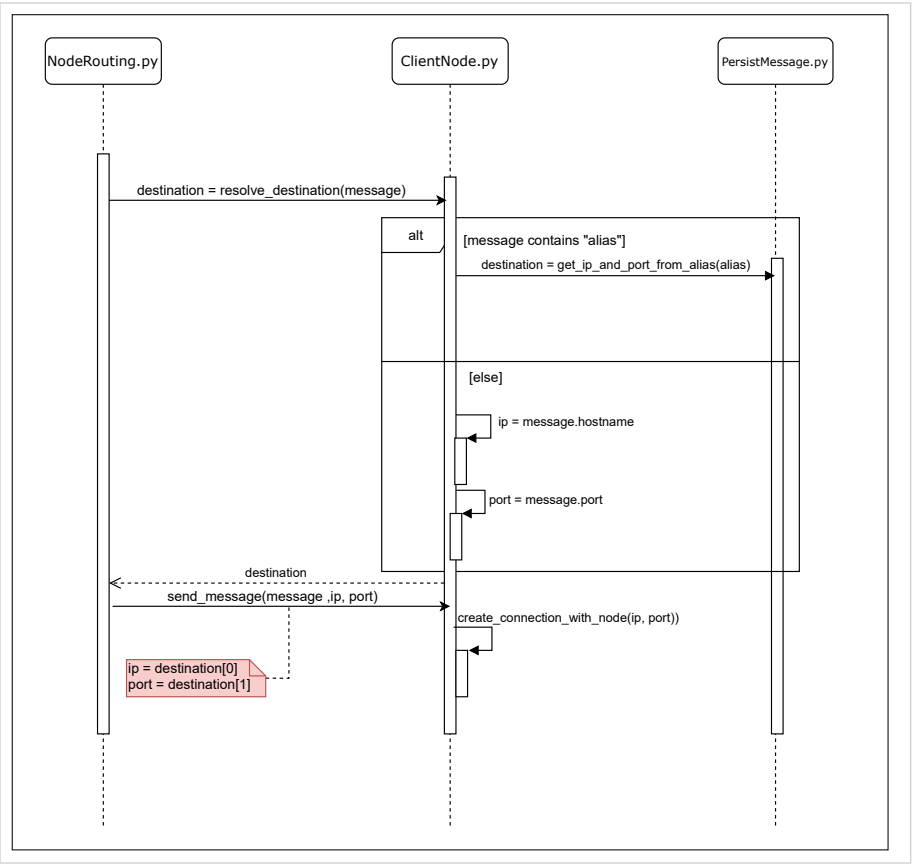
In dit hoofdstuk staan sequence diagrammen beschreven op een functionele manier. Dit houdt in dat objecten in dit geval niet aanwezig zijn, maar er gekeken wordt naar de relaties tussen de python files. Bij het aanroepen van een functie kan dus verwezen worden naar de verschillende files.

6.2.1. UC Exchange Message

Van node naar node EN van node naar client (zelfde flow)

In Figuur 27 wordt de volgorde van functie aanroepen beschreven in de stroom van bericht versturen vanaf een Node naar een client of een andere node. Deze stroom begint bij send message in ClientNode.py. De message die meegegeven wordt aan de send message en resolve destination functies is in JSON-formaat zoals in 3.2.4. Onion beschreven staat. Hier kan niet van worden afgeweken, aangezien vooraf deze sequentie een validatie plaats vindt die ervoor zorgt dat als de binnengekomen message hier niet aan voldoet, dat send_message niet wordt aangeroepen.

Wijziging 11 jun: Flow is gewijzigd, en op basis daarvan is de diagram verbeterd

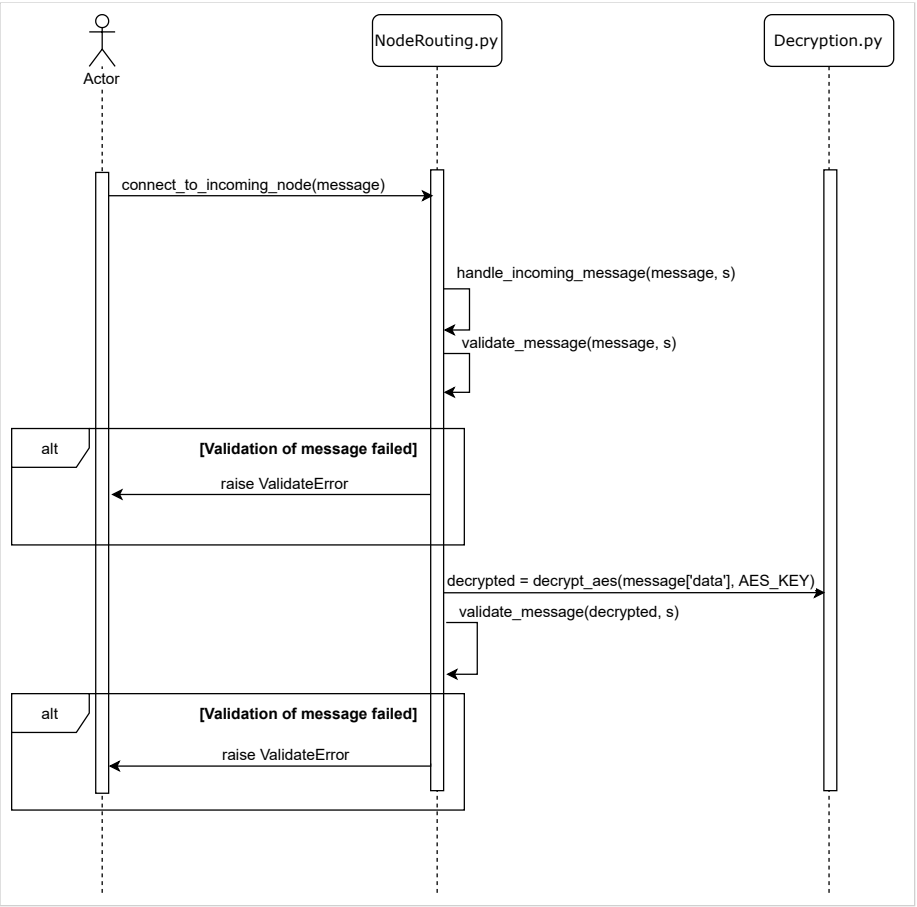


Figuur 27: Sequence diagram Node naar Client

Van client naar node

In Figuur 28 staat omschreven hoe de flow gaat van een binnengekomen bericht op een node. De actor stuurt een message naar de openstaande socket van de node. Deze message moet voldoen aan de opbouw die gespecificeerd is in 3.2.4. Onion. Dit bericht wordt vervolgens gevalideerd of het de data bevat die wordt verwacht door een node. Als er een positief resultaat is gekomen uit de validatie dan gaat de node de data uit de JSON decrypten. Deze ontsleutelde data wordt vervolgens ook weer gevalideerd of het de verwachte structuur heeft alvorens er vervolgacties uitgevoerd worden.

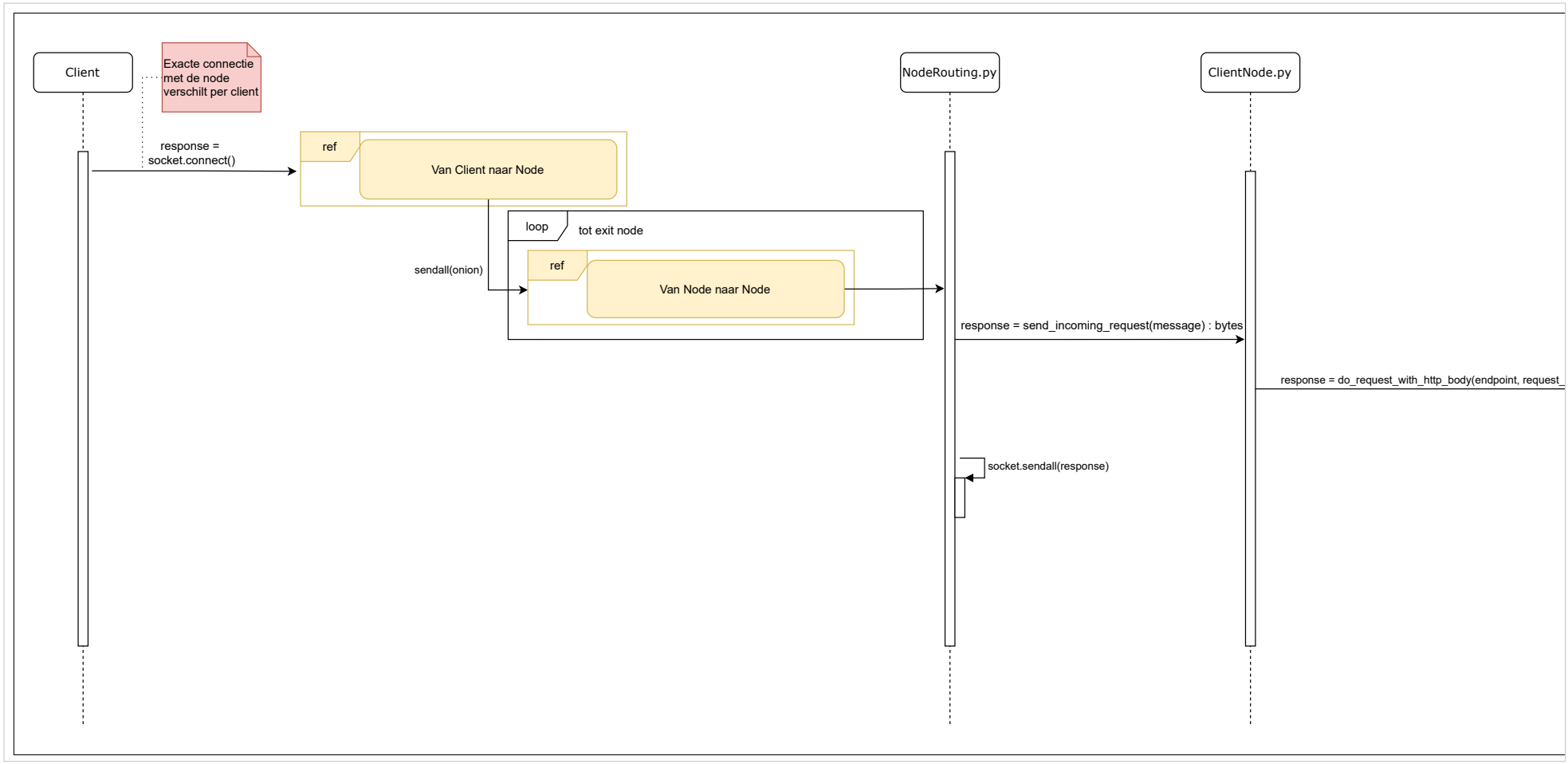
Wijziging 11 jun: Flow is gewijzigd, er is een link gelegd naar vervolgstappen van deze flow en de functies zijn niet meer zoals omschreven in deze diagram



Figuur 28: Sequence diagram client naar node met encryptie

client http-request via de nodes

In de onderstaande Sequence Diagram staat beschreven hoe de flow eruit ziet wanneer de client een http-request wil doen door het node netwerk. In principe lijkt dit heel erg op de flow van een normaal bericht versturen. Het grootste verschil is hier alleen dat de client een response verwacht op basis van de http-request die de node uitvoert. De laatste node zal hier een http-request doen, die hij over alle nodes weer terugstuurd naar de client. Dit betekent ook dat de clients en de nodes met elkaar een connectie openhouden tot de response naar de vorige node/client verstuurd is.

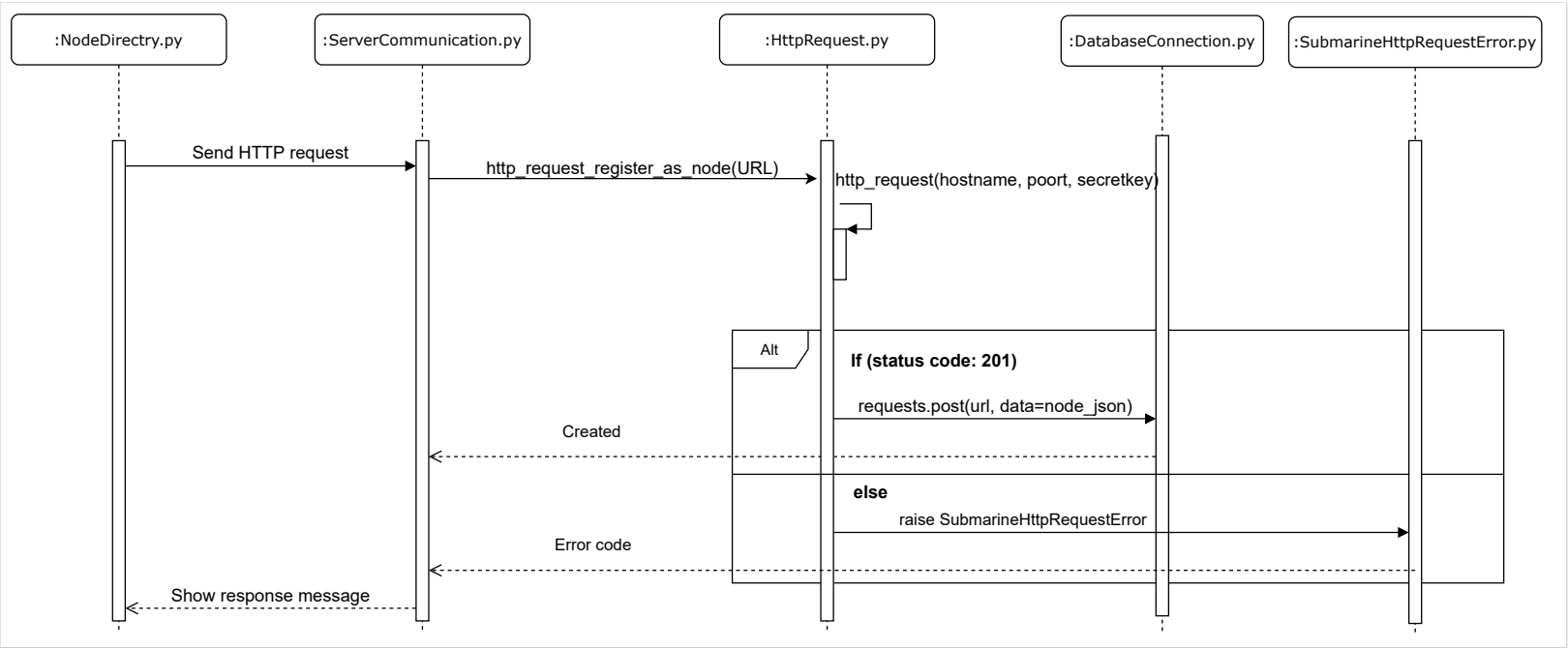


Figuur 29: Client http request via de nodes

Node aanmelden bij centrale server

In de onderstaande Sequence Diagram staat beschreven hoe de flow eruit ziet wanneer een node zich wilt aanmelden bij de centrale server. Een node verstuurt een HTTP request met daarin de Destination die bestaat uit hostnamen, poort en secret key. Indien de HTTP request postitief blijkt te zijn worden de gegeven opgeslagen in de database van de server en krijgt de gebruiker een bericht dat het gelukt is. Indien het niet lukt om HTTP-verbinding te maken of onstaat er andere error dan worden de gegevens niet opgeslagen en krijgt de gebruiker een foutmelding te zien.

Wijziging 11 jun: Van deze diagram klopt vrij weinig, zowel syntactisch als de gehele flow. @Stefan Anbeek heeft deze diagram volledig opnieuw geschreven

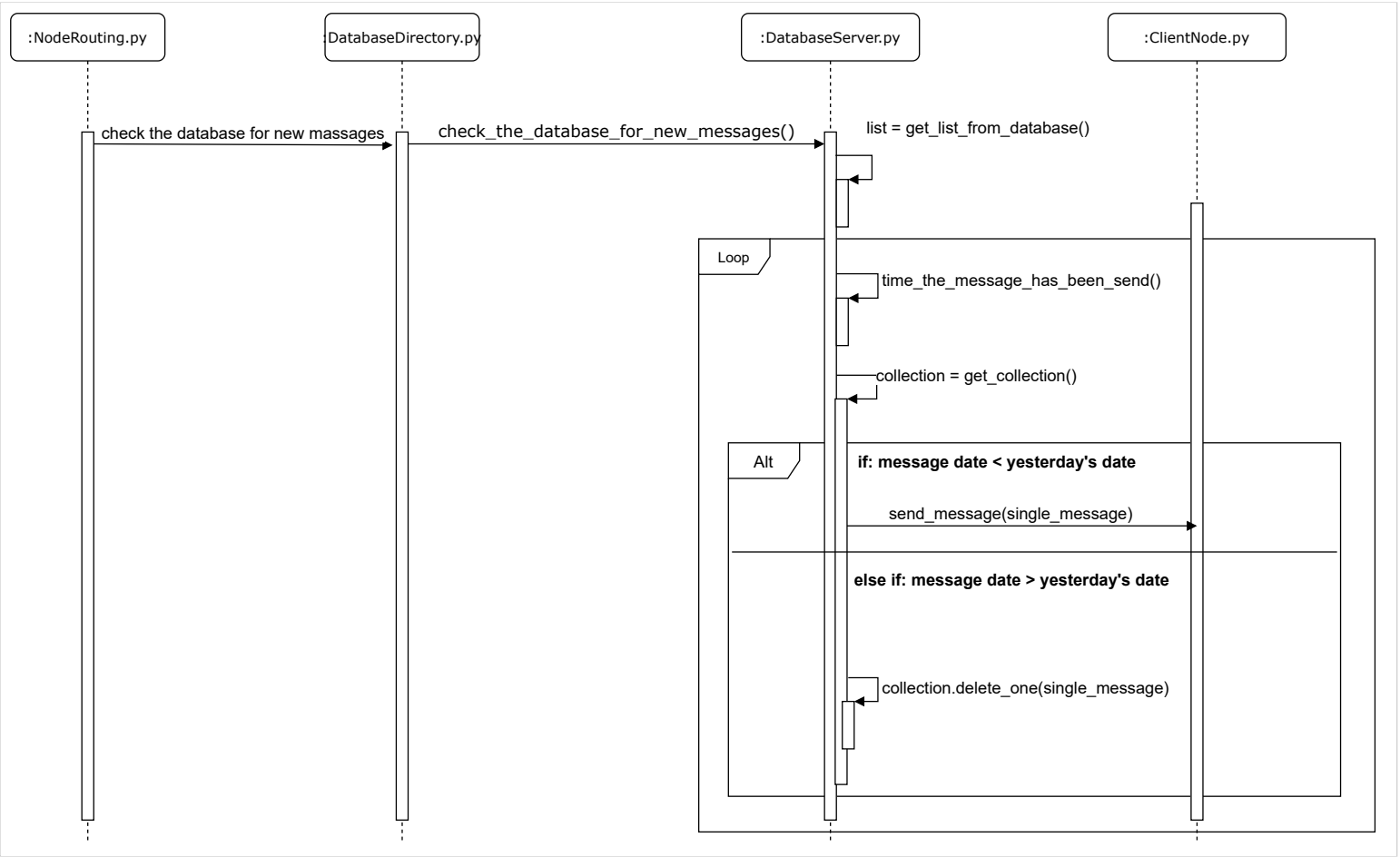


Figuur 30: Node aanmelden bij centrale server

Node automatiseren voor het opnieuw versturen van berichten vanuit de database

In de onderstaande Sequence Diagram staat beschreven hoe de flow eruit ziet van het geautomatiseerd aanbieden van berichten. Wanneer een gebruiker offline blijkt te zijn wordt het bericht opgeslagen in de database van de laatste node. Om ervoor te zorgen at de berichten daar niet altijd blijven hangen. Is het gelukt om automatisch berichten te versturen vanuit de node. Een node maakt verbinding met de Mongo database en vraagt alle berichten op. Alle opgevraagde berichten worden in een lijst gezet en vervolgens wordt over de lijst geïtereerd. Als een bericht niet ouder is als 1 dag wordt die opgestuurd naar de gebruiker. Wanneer een bericht te oud blijkt te zijn, wordt die verwijderd.

Wijziging 11 jun: Van deze diagram klopt vrij weinig. @Stefan Anbeek heeft deze diagram volledig opnieuw geschreven, zodat het klopt met de code.



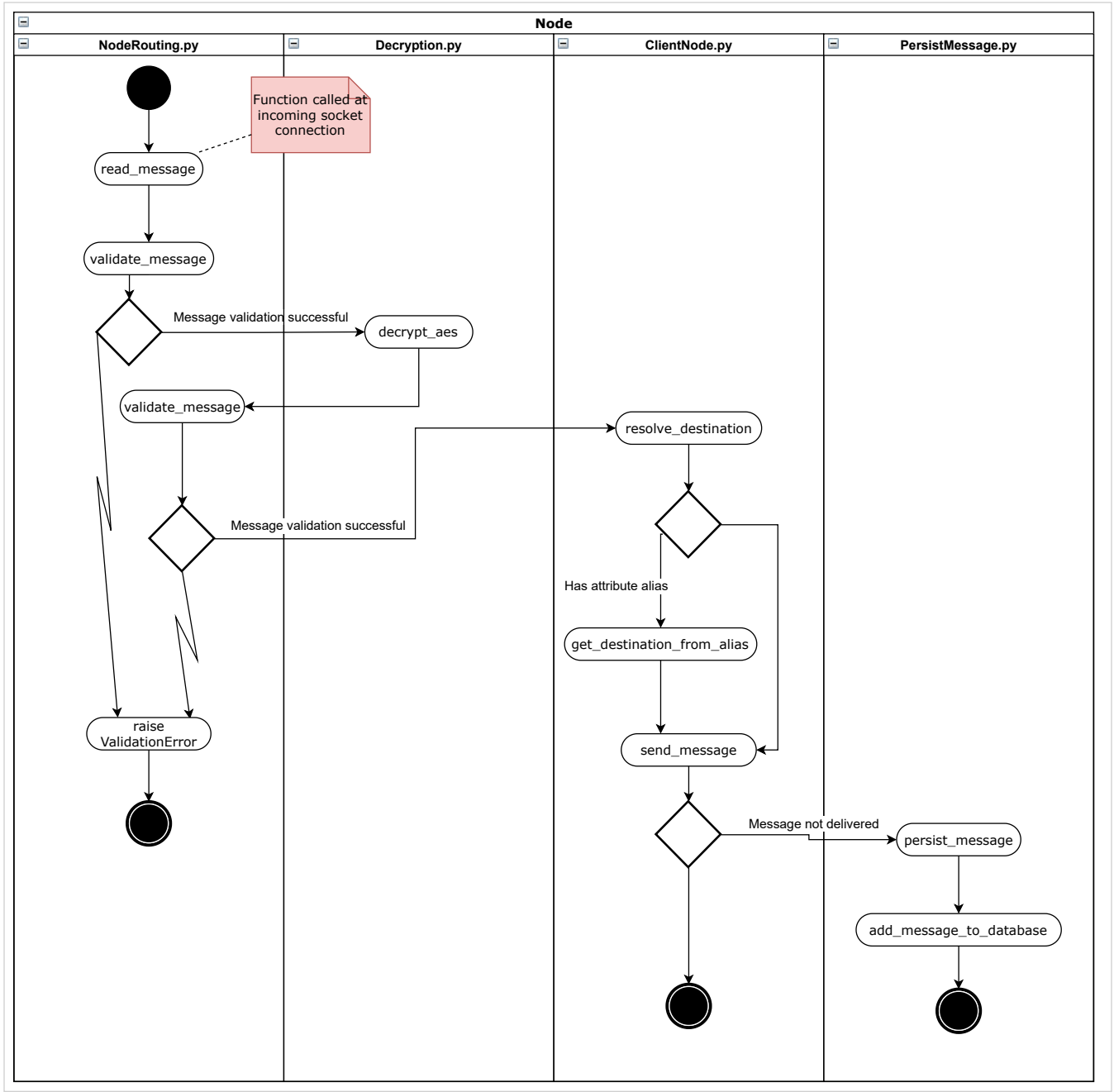
Figuur 31: Node automatiseren voor het opnieuw versturen van berichten vanuit de database

6.3. Activity and State Diagrams

6.3.1. UC Exchange Message

Activity diagram node relay flow

Omdat de flow niet voor zichzelf spreekt en vrij lastig duidelijk gemaakt kan worden met een sequence diagram, hebben wij gekozen om hier een activity diagram voor te maken.



Figuur 32: Activity diagram node relay flow

Activiteit	Toelichting
read_message	Functie die luistert naar binnenkomende berichten over een socket.

Activiteit	Toelichting
validate_message	Functie die een bericht valideert aan een vastgesteld JSON formaat. Deze functie wordt twee keer aangeroepen, omdat de binnengekomen data gecontroleerd moet worden op valide input. Ook wordt er na het decrypten van de laag gecontroleerd of er een valide bericht over blijft.
decrypt_aes	Decrypt een message door een laag van de ui af te halen. Met een 32-bytes symmetrische sleutel en het versleutelde bericht, wordt een nieuwe laag van de ui blootgesteld. Het versleutelde bericht moet een block length hebben van 16 bytes.
resolve_destination	Functie die kijkt of de bestemming van het bericht naar een client of naar een node moet. Dit wordt gecontroleerd door te checken of er data aanwezig is voor het "alias" attribuut in het JSON bericht.
get_destination_from_alias	Functie die voor de meegegeven alias een ip en port zoekt in de database om het bericht naar de client te kunnen sturen.
send_message	Aan de hand van de JSON die wordt meegegeven aan de functie, bepaalt deze functie waar het bericht naartoe moet worden gestuurd. Over een socket-verbinding wordt het JSON bericht verstuurd.
persist_message	Functie om de ConnectionModule met de PersistenceModule te koppelen om een message op te slaan in de database.
add_message_to_database	Functie om een bericht op te slaan in de database.

Tabel 18: Toelichting functies node relay flow

6.4. Ontwerpbeslissingen voor het deelsysteem

Wijziging 11 jun: Er waren meer ontwerpbeslissingen genoemd. Onder andere de keuze voor de bibliotheken die wij hebben gebruikt in Python zoals Request. En nog overige beslissingen die van invloed waren op de Python applicatie.

6.4.1. Gebruik van commands om flow te bepalen

Beslissing	Beschrijving
Probleem/Issue	Er komen verschillende relay flows binnen de nodes en er moet een manier komen om binnenkomende berichten te onderscheiden, om zo te bepalen welke flow het bericht moet bewandelen.
Beslissing	Er is gekozen voor het gebruik van de volgende commands: • RELAY; Dit command wordt gebruikt bij het versturen van berichten tussen clients. Hierbij wordt geen response verwacht. • HTTP_RELAY; Dit command wordt gebruikt bij het doen van HTTP requests over de nodes. Hier wordt een socket verbinding open gehouden tot er een response wordt teruggestuurd van het HTTP request. • HTTP_REQUEST; Dit command wordt gebruikt als er daadwerkelijk een HTTP request moet worden gedaan. Het resultaat wordt dan teruggegeven aan de openstaande socket. Hierna wordt de socket geclosed.
Alternatieven	Er was geen goed alternatief voor het probleem, naast het uitlezen van meegestuurde data en een patroon hierin herkennen.
Argumenten	Door gebruik te maken van commands kunnen we een duidelijk onderscheid maken in flows. Ook is het bepalen van de flow nu simpel aan te geven vanuit de clients.
Gemaakt op	04 Jun 2020
Gemaakt door	@Coen Hoogduin , @Thijs Baan , @Stefan Anbeek

Tabel 19: Ontwerpbeslissing commands

6.4.2. Berichten opnieuw versturen naar gebruikers vanuit de database

Beslissing	Beschrijving
Probleem/Issue	Wanneer een gebruiker offline gaat en niet alle berichten zijn afgeleverd worden die berichten opgeslagen in de database. Om ervoor te zorgen dat de database niet overvol raakt worden berichten herzonden naar de ontvanger.
Beslissing	Berichten die opgeslagen zijn in de database worden elke vijf minuten, een dag lang opnieuw herzonden naar de ontvanger. Wanneer de berichten niet afgeleverd worden na een dag, worden ze verwijderd.
Alternatieven	Als alternatief kunnen berichten voor altijd worden opgeslagen en niet verwijderd worden. Dit heeft erg grootte impact op de database, omdat er mogelijk duizenden of miljoenen berichten opgeslagen moeten worden voor altijd.
Argumenten	Omdat security een belangrijke rol speelt kunnen berichten niet voor eeuwigheid worden opgeslagen. Het is niet veilig om dat te doen. Daarom worden ze nu een dag lang aangeboden om nog eens verstuurd te worden. Wanneer dat niet lukt worden ze verwijderd.
Gemaakt op	08 Jun 2020
Gemaakt door	@Mario Blaucik , @Stefan Anbeek

Tabel 20: Beslissing: Berichten versturen vanuit de database

6.4.3. Gebruik hostname mapping

Beslissing	Beschrijving
Probleem/Issue	Omdat de nodes die gehost worden door de hosting partij in hetzelfde LAN zitten als de Directory Node en de Chatter Directory kunnen deze elkaar niet benaderen via het WAN IP-adres.

Beslissing	In de /etc/hosts file van de nodes hebben we de volgende regels toegevoegd om het netwerkverkeer naar de hostnames goed te laten verlopen naar de LAN IP adressen:
	`# Server mappings`
	`10.20.3.122 asd-p1-server1.asd.icaprojecten.nl`
	`10.20.3.132 asd-p1-server2.asd.icaprojecten.nl`
	`10.20.3.139 asd-p1-server3.asd.icaprojecten.nl`
	`10.20.3.165 nod-server.sanstech.net`
	`10.20.3.166 acc-server.sanstech.net`

Alternatieven	Er had een hard-coded mapping in de software gedefinieerd kunnen worden
Argumenten	Met deze aanpassing hoeft de software niet aangepast te worden. Zo blijft de software dus hetzelfde als de hosting van de nodes wordt veranderd.
Gemaakt op	08 Jun 2020
Gemaakt door	@ Stefan Anbeek

Tabel 21: Beslissing: Gebruik hostname mapping

6.4.4. Gebruik van externe bibliotheek voor taakplanning

Beslissing	Beschrijving
Probleem/Issue	Wanneer een gebruiker offline gaat kunnen niet alle berichten naar hem worden verzonnen. In dat geval worden de berichten opgeslagen in de database. Het probleem is dat ze daar altijd blijven liggen en niet opnieuw worden opgestuurd.
Beslissing	Er is gekozen om gebruik te maken van een schedule library die het mogelijk maakt om een taak om een bepaalde tijd uit te voeren.
Alternatieven	Als alternatief kon Cron job gebruikt worden. De Cron job zou dan op de server gerund kunnen worden en vervolgens een script runnen die de berichten uit de database ophaalt en nog een keer verstuurd.
Argumenten	Er is gekozen voor Schedule omdat het: • Een eenvoudig te gebruiken API voor het plannen van taken. • Zeer lichtgewicht is en geen externe afhankelijkheden. • Cron jobs heeft externe configuratie nodig buiten de code om.
Gemaakt op	01 Jun 2020
Gemaakt door	@ Mario Blaucik , @ Stefan Anbeek

Tabel 22: Beslissing: Gebruik van externe bibliotheek voor taakplanning

6.4.5. Cryptography library voor implementatie AES

Beslissing	Beschrijving
Probleem/Issue	De implementatie met de Crypto library geeft veel problemen bij windows gebruikers om de software te runnen. Ook faalde de buildstraat met die library.
Beslissing	Er is gekozen om gebruik te maken van de mindergebruikte library cryptography.
Alternatieven	• Crypto library gebruiken en de ontstane bugs oplossen. • Zelf AES algoritme uitwerken.
Argumenten	Omdat de Crypto library zo veel problemen gaf, en er ook google ook veel bugs waren gemeld met hetzelfde probleem vond ik de library niet betrouwbaar. Zelf AES uitwerken is veel te veel werk voor dit project.

Beslissing	Beschrijving
Gemaakt op	27 May 2020
Gemaakt door	@ Stefan Anbeek

Tabel 23:

Beslissing	Beschrijving
Probleem/Issue	
Beslissing	
Alternatieven	
Argumenten	
Gemaakt op	
Gemaakt door	

Tabel 24:

Beslissing	Beschrijving
Probleem/Issue	
Beslissing	
Alternatieven	
Argumenten	
Gemaakt op	
Gemaakt door	

Tabel 25:

<Decisions toevoegen, kan niet dat er geen design decisions voor zijn gekomen.>

Mogelijke decision: Lengte van message prependen

7. Design Node Directory Server

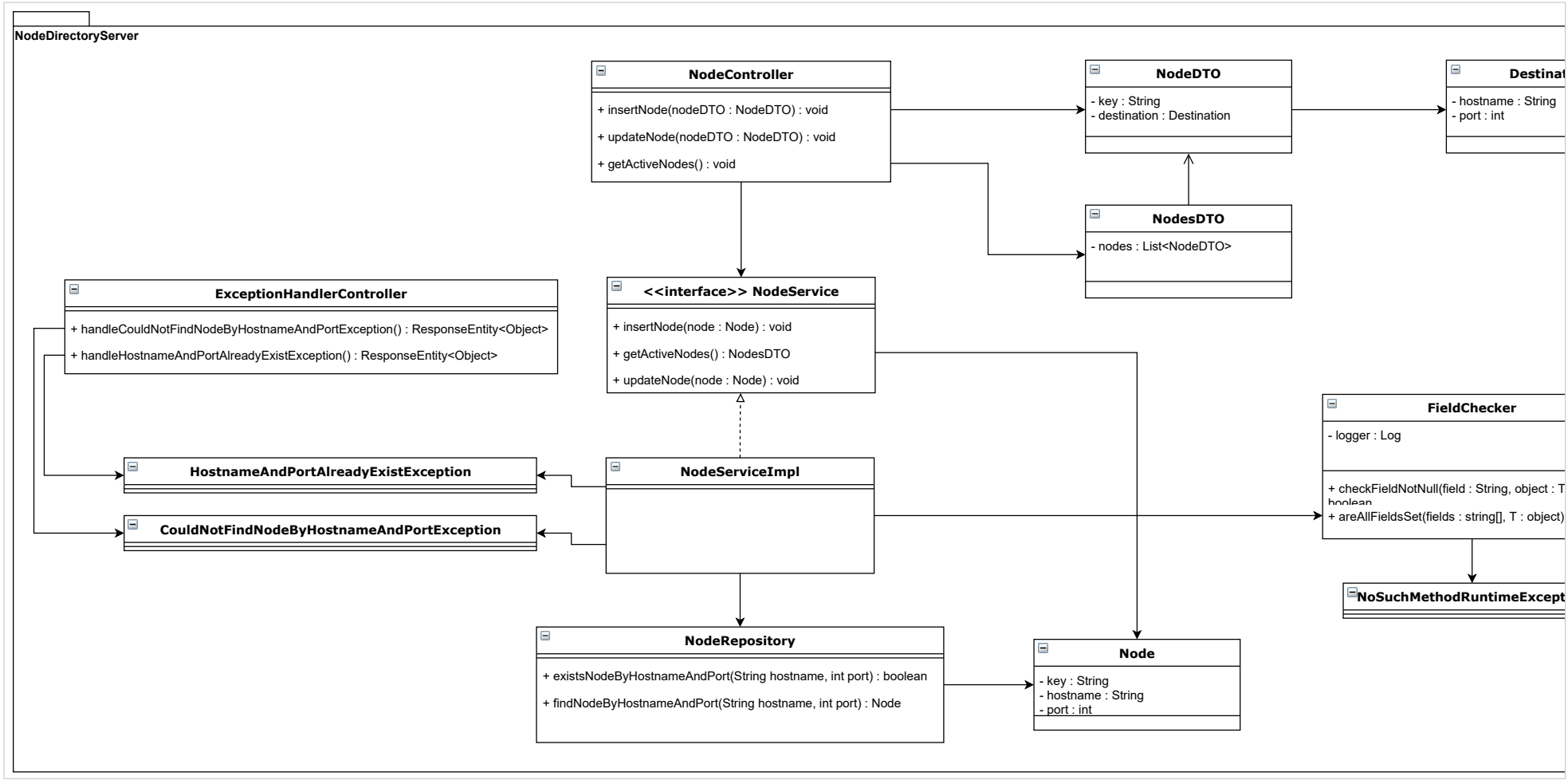
Wijziging 11 jun: Uitgebreidere inleiding

De node directory server heeft als doel om gegevens met betrekking tot de nodes op te slaan. Clients halen door middel van HTTP-requests gegevens van nodes op die zij nodig hebben bij de opbouw van een onion en het versturen ervan. Nodes melden zich aan bij deze server, om hun destination en key te delen. De node directory server biedt meerdere REST-endpoints aan om deze acties te kunnen doen.

7.1. Design Class Diagram

Voor het ontwerp van het systeem is er rekening gehouden met het layer pattern, zoals we die aangeleerd is in de DEA course. Hierbij is de NodeRepository de persitencelaag, NodeService de servicelaag en de NodeController de presentationlaag. Hiermee wordt een lage koppeling en een hoge cohesie gewaarborgd. Het gehele class diagram staat in Figuur 33.

Wijziging 11 jun: Consistent maken van de design class diagram met de daadwerkelijke code. De exception NoSuchRuntimeException was hier bijvoorbeeld verwijderd, waarvoor FieldCheckerException in de plaats kwam. Ook was er een uitgebreidere uitleg voor de diagram



Figuur 33: Design Class Diagram

7.2. Endpoint definities

Wijziging 11 jun: Uitgebreidere inleiding

Voor het systeem Submarine zijn er API calls nodig naar de REST-endpoints van de node directory server. In dit hoofdstuk wordt de specificatie van deze endpoints verduidelijkt.

7.2.1. UC Exchange message

Wijziging 11 jun: Uitgebreidere uitleg

Voor de use case 'Exchange message' zijn calls benodigd naar de node directory om zo een node te kunnen aanmelden en een lijst met nodes te kunnen opvragen.

Node aanmelden

Deze endpoint is bestemd voor een node die zich voor het eerst aanmeldt bij de directory server. De meegegeven destination is de unieke waarde in het systeem en kan niet meer dan een keer voorkomen in de database.

```
Create node
```

```
url:            /node/create
method:         POST
status code:    201 (Created)

Request body need to be conform the NodeDTO model. Example in JSON:
{
    "destination": {
        "hostname": "xxx.xxx.com"
        "port": 25010},
    "key": "9H+5ObOd2IO/KQcGTTnKR5h3F1DrWFjXt0oud+zlL0Q="
}

Of note here is that the key should be a base64 encoded string representing the bytes of the secret key
Common exceptions:
status 400:     Bad Request
message:        "Fill in all fields"
```

Figuur 34: Endpoint definitie Node aanmelden

Update key

Bij deze endpoint kan een node de secret key updaten. Hier wordt de destination gebruikt als unieke waarde. Als deze bestaat in de database wordt de key in de database aangepast naar de meegegeven key.

```
Update node
url:            /node/update
method:         POST
status code:    200 (OK)

Request body need to be conform the NodeDTO model. Example in JSON:
{
    "destination": {
        "hostname": "xxx.xxx.com"
        "port": 25010},
    "key": "9H+5ObOd2IO/KQcGTTnKR5h3F1DrWFjXt0oud+zlL0Q="
}

Of note here is that the key should be a base64 encoded string representing the bytes of the secret key
Common exceptions:
status 400:     Bad Request
message:        "Fill in all fields"
```

Figuur 35: Endpoint deifnitie Update key

Opvragen actieve nodes

Deze endpoint geeft een lijst terug van alle beschikbare nodes. Deze wordt gebruikt door de clients om routes te kunnen genereren en berichten te kunnen verzenden.

```
Get all available nodes
url:            /node
method:         GET
status code:    200 (OK)

Response body should be conform NodesDTO, which is a list of NodeDTO. For example:
{
    [{"destination": {
        "hostname": "xxx.xxx.com"
```

```
        "port": 25010},
    "key": "9H+5ObOd2IO/KQcGTTnKR5h3F1DrWFjXt0oud+zlL0Q="},
    "destination": {
        "hostname": "yyy.yyy.com"
        "port": 25010},
    "key": "9H+5ObOd2IO/KQcGTTnKR5h3F1DrWFjXt0oud+zlL0Q="}]
}

Common exceptions:
status 400:     Bad Request
```

Figuur 36: Endpoint definitie Opvragen actieve nodes

7.3. Sequence Diagrams

Wijziging 11 jun: Uitgebreidere inleiding

In dit hoofdstuk zijn sequence diagrams opgenomen om de flow van de Node Directory Server inzichtelijk te krijgen.

7.3.1. UC Exchange message

Wijziging 11 jun: Uitgebreidere uitleg

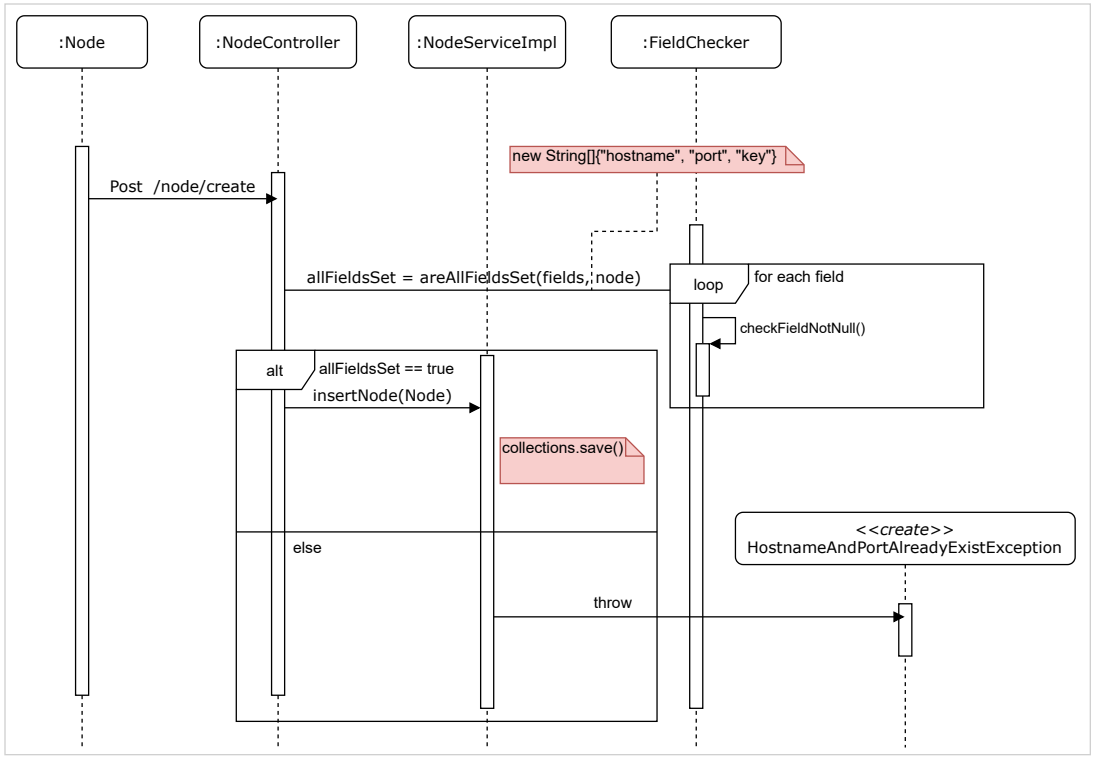
Voor iedere endpoint die wordt aangeboden binnen de use case is er een sequence diagram gemaakt om de flow te verduidelijken.

Node aanmelden

Wijziging 11 jun: Uitgebreidere uitleg

In Figuur 37 wordt de flow van het aanmelden van de node weergegeven.

Wijziging 11 jun: Diagram klopt(e) niet en is veranderd om wel consistent te zijn met de code.

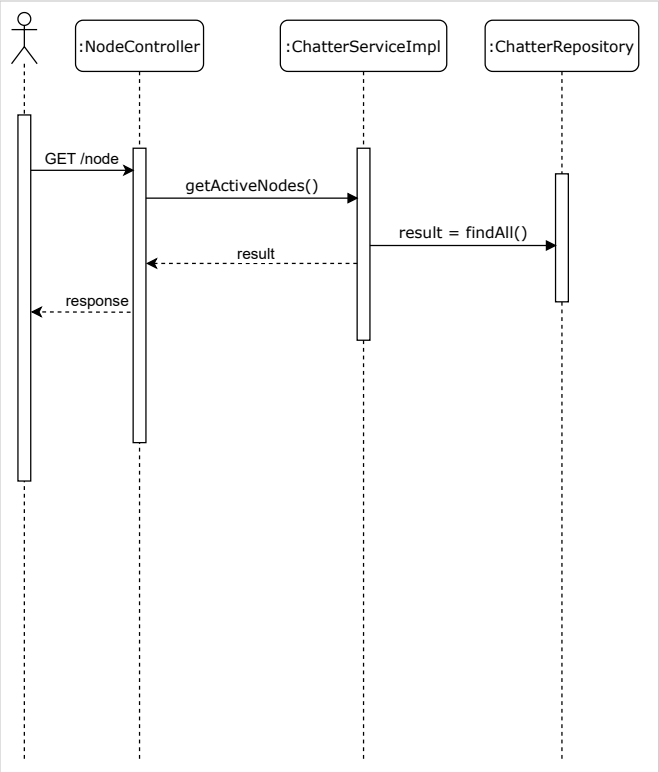


Figuur 37: SD: Node aanmelden

Opvragen actieve nodes

Wijziging 11 jun: Uitgebreidere uitleg

De flow van het opvragen van alle nodes wordt weergegeven in Figuur 38.

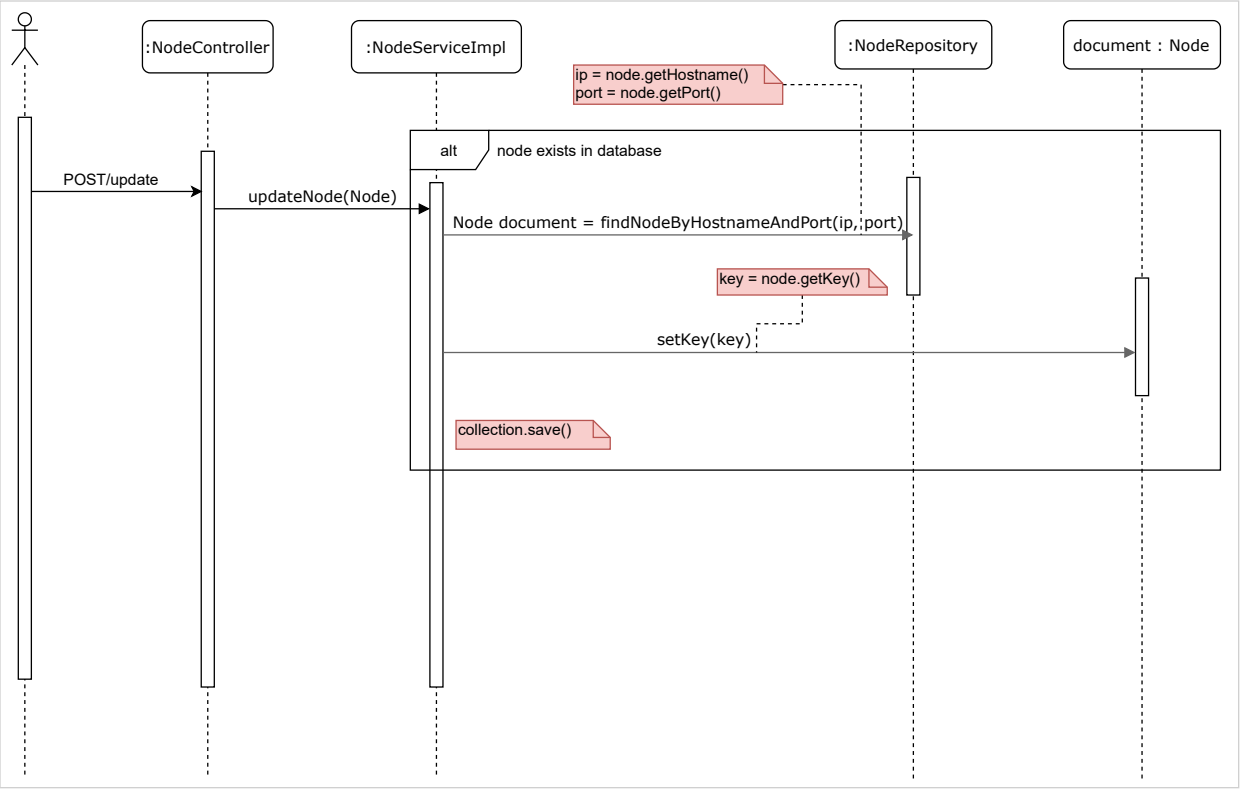


Figuur 38: Sequence diagram opvragen actieve nodes

Update key

Wijziging 11 jun: Uitgebreidere uitleg

In Figuur 39 wordt de flow van het updaten van de secret key weergegeven.



Figuur 39: Sequence diagram updaten node key

7.4. Ontwerpbeslissingen voor het deelsysteem

Wijziging 11 jun: Toelichting gemaakte keuzes van Java Springboot i.p.v. andere frameworks en Lombok om geen Getters en Setters te hoeven gebruiken. Verder was er een uitgebreidere inleiding gemaakt

In dit hoofdstuk zijn ontwerpbeslissingen opgenomen die betrekking hebben op de implementatie van de Node Directory Server.

7.4.1. Gebruik database bij uitvoering testen

Beslissing	Beschrijving
Probleem/Issue	De database query's moeten getest worden met een database. De vraag is met welke database er getest wordt.
Beslissing	Er is gekozen om met een replica van de productiedatabase te testen. Dit houdt in dat de database properties hetzelfde zijn op de database naam na. Het overschrijven van de properties is gedaan door middel van een annotatie in de tests.
Alternatieven	• Het testen op een in memory database • Het testen op de productiedatabase
Argumenten	• Het is niet verstandig om op de productiedatabase te testen. • De in memory database verschilt sterk van de Mongo database. Er kan dus beter met een Mongo database getest worden.
Gemaakt op	08 Jun 2020
Gemaakt door	@ Stefan Anbeek

Tabel 26: Keuze gebruik database bij uitvoering testen

7.4.2. Exception handlers

Beslissing	Beschrijving
Probleem/Issue	Hoe worden exceptions afgevangen in de code waardoor er een correcte response gegeven wordt.
Beslissing	Er is gekozen om een exception handler toe te passen. De handler is opgenomen in het bestand controllers/ExceptionHandlerController.java.
Alternatieven	• Het geven van een HTTP response vanuit de code.
Argumenten	• Door gebruik te maken van een exception handler heeft de code geen verstand van het protocol dat gebruikt wordt. • Er kan gemakkelijk een exception handler aangemaakt worden alle mogelijke excepties.
Gemaakt op	09 Jun 2020
Gemaakt door	@ Stefan Anbeek

Tabel 27: Beslissing: Exception handlers

8. Design Chatter Directory Server

De Chatter Directory Server heeft als doel om de gebruikers van het systeem op slaan, te verifiëren en om hun IP-adressen op te halen om zo het netwerk compleet te maken. Zoals in het Software Architecture Document beschreven is deze server uitgewerkt met behulp van Java Spring. Met behulp van dit framework is een REST-webservice opgezet waarmee gebruikers via het Node netwerk kunnen communiceren. In dit hoofdstuk wordt middels een class diagram, endpoint definities, sequence diagrams en ontwerpbeslissingen toegelicht hoe het systeem in elkaar zit.

8.1. Design Class Diagram

In Figure 2 wordt het class diagram weergegeven van de Chatter Directory Server.

Wijziging 11 juni: rechtgetrokken met code

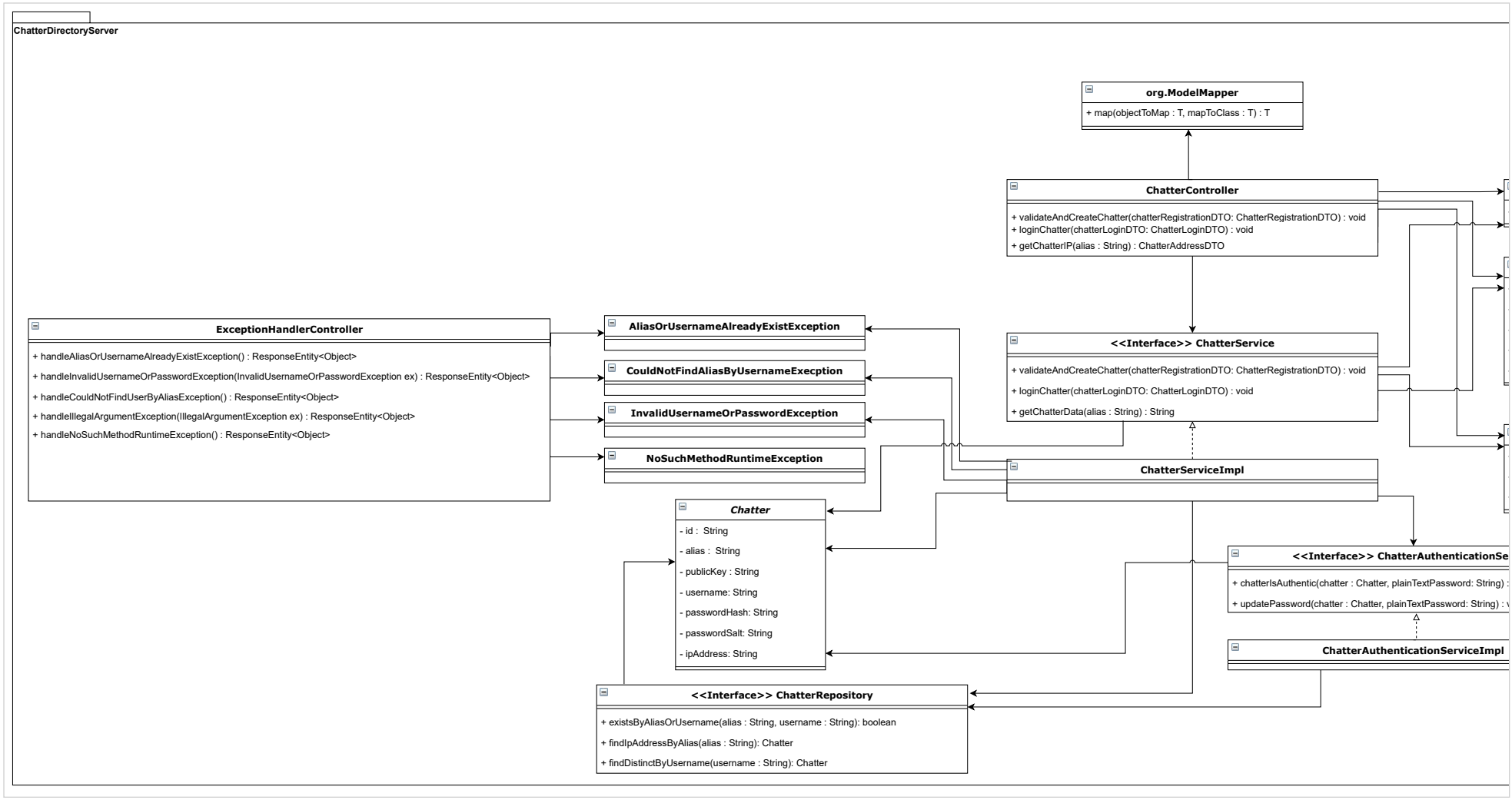


Figure 2: Design Class Diagram

8.2. Endpoint definities

Wijziging 11 juni: endpoints rechtgetrokken met code

Dit hoofdstuk geeft inzicht op de endpoints die aan te roepen zijn op de Chatter Directory Server. De endpoints zijn gedocumenteerd met een simpele documentatiemethode die inzicht geeft op de eigenschappen van de endpoint, de data die verwacht wordt en geretourneerd wordt en de meest voorkomende excepties.

8.2.1. UC manage account

Er zijn op de Chatter Directory Server twee endpoints gemaakt als onderdeel van de usecase manage account.

Registreren gebruiker

Deze endpoint is verantwoordelijk voor het opslaan van de gegevens die voor het netwerk van belang zijn. Deze endpoint wordt gebruikt door de clients. De gegevens worden door de nodes gebruikt om berichten naar de juiste alias te sturen.

```
url:            /chatter/create
method:         POST
status code:    201 (Created)


Request body need to be conform the ChatterRegistrationDTO model. Example in JSON:

{
    "alias": "TEST_ALIAS",
    "publicKey": "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAil95F4w0F0xcn8teY/LqT98Eo0/U7y0u6qMyYXqnXoQ/BA22rUU8GVF4WOHgn4VE1LwYPJdv6TIBmd1IbTTlzF9+NMPHs1njxrcY4NpWTsQc5diyU3ulh/bD0EY5buH
    "username": "TEST_USERNAME",
    "password": "TEST_PASSWORD",
    "ipAddress": "TEST_IP"
}


Of note here is that the publicKey should be a base64 encoded string representing the bytes of the public key

Common exceptions:

status 400:     Bad Request
message:        "Fill in all fields"

status 400:     Bad Request
message:        "Alias or username already exist"
```

Code 3: Endpoint: Registeren Gebruiker

Login chatter

Deze endpoint is verantwoordelijk voor het inloggen van de gebruikers. Deze endpoint wordt gebruikt door de clients. Het ip wat wordt meegegeven wordt gebruikt door de nodes.

```
url:            /chatter/login
method:         PUT
status code:    200 (Ok)


Request body need to be conform the ChatterLoginDTO model. This is filled with username, password and ip address. Example in JSON:

{
    "username": "TEST_USERNAME",
    "password": "TEST_PASSWORD",
    "ipAddress": "TEST_IP"
}


Common exceptions:

status 403:     Forbidden
message:        "Username or password is incorrect"

status 400:     Bad Request
message:        "The given IPv4 address is invalid"
```

```
status 400:     Bad Request
message:        "Not all fields are filled in. Expected:
                 {
                    "username": (String),
                    "password": (String),
                    "ipAddress": (String)
                 }"

status 500:     Internal Server Error
message:        "A method was unknown"
```

Code 4: Endpoint: Login chatter

8.2.2. UC exchange message

Er is op de Chatter Directory Server één endpoint gemaakt als onderdeel van de usecase exchange message.

Get chatter IP address

De endpoint "get chatter IP address" is verantwoordelijk voor het ophalen van een IP-adres van een gebruiker op basis van zijn alias. Deze endpoint wordt aangeroepen door exit nodes om berichten te kunnen versturen naar de ontvanger.

```
url:            /chatter/get-ip/{alias}
method:         GET
status code:    200 (Ok)


Response body send back ChatterAddressDTO by mapping Chatter. This is filled with ipAddress. Example in JSON:

{
    "ipAddress": (String)
}


Common exceptions:

status 404:     Not found
message:        "Could not find user by alias"
```

Endpoint 1: Get IP

8.3. Datamodel

In de Chatter Directory Server worden chatters opgeslagen in een Mongo database. In Figuur 40 is te zien wat de inhoud is van een chatter document in de database.

Chatter data

```
{
   "title":"Chatter",
   "description":"Een chatter zoals deze gebruikt wordt in de Chatter Directory Server.",
   "attributes":{
      "alias":{
         "type":"string",
         "description": "De alias van de gebruiker.",
         "example":"John_Doe"
```

```
      },
      "publicKey": {
         "type": "string",
         "description": "De public key van de contact om berichten mee te versleutelen.",
         "example": "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsvX9P58JFxEs5C+L+H7WduFSWL5EPzber7C2m94klrSV6q0bAcrYQnGwFOlveThsY200hRbadKaKjHD7qIKHDEe0IY2PSRht33Jye52AwhkRw+M3xuQH/7R8Lydn
      },
      "username": {
         "type": "string",
         "description": "De gebruikersnaam van de gebruiker.",
         "example": "John_Doe"
      },
      "passwordHash": {
         "type": "string",
         "description": "De gehashte wachtwoord van de gebruiker.",
         "example": "CYzDD9DzGlgk1U8y9Dh/sgWd3Kh8UmC"
      },
      "passwordSalt": {
         "type": "string",
         "description": "De salt bijbehorende de opgeslagen wachtwoord hash.",
         "example": "$2a$10$vPZ3EKgL8zt4M2TrlcptA."
      },
      "ipAddress": {
         "type": "string",
         "description": "Het IP-adres van de gebruiker waarmee hij met het node netwerk verbind.",
         "example": "95.17.46.221"
      }
   }
}
```


Figuur 40: Chatter

8.4. Sequence Diagrams

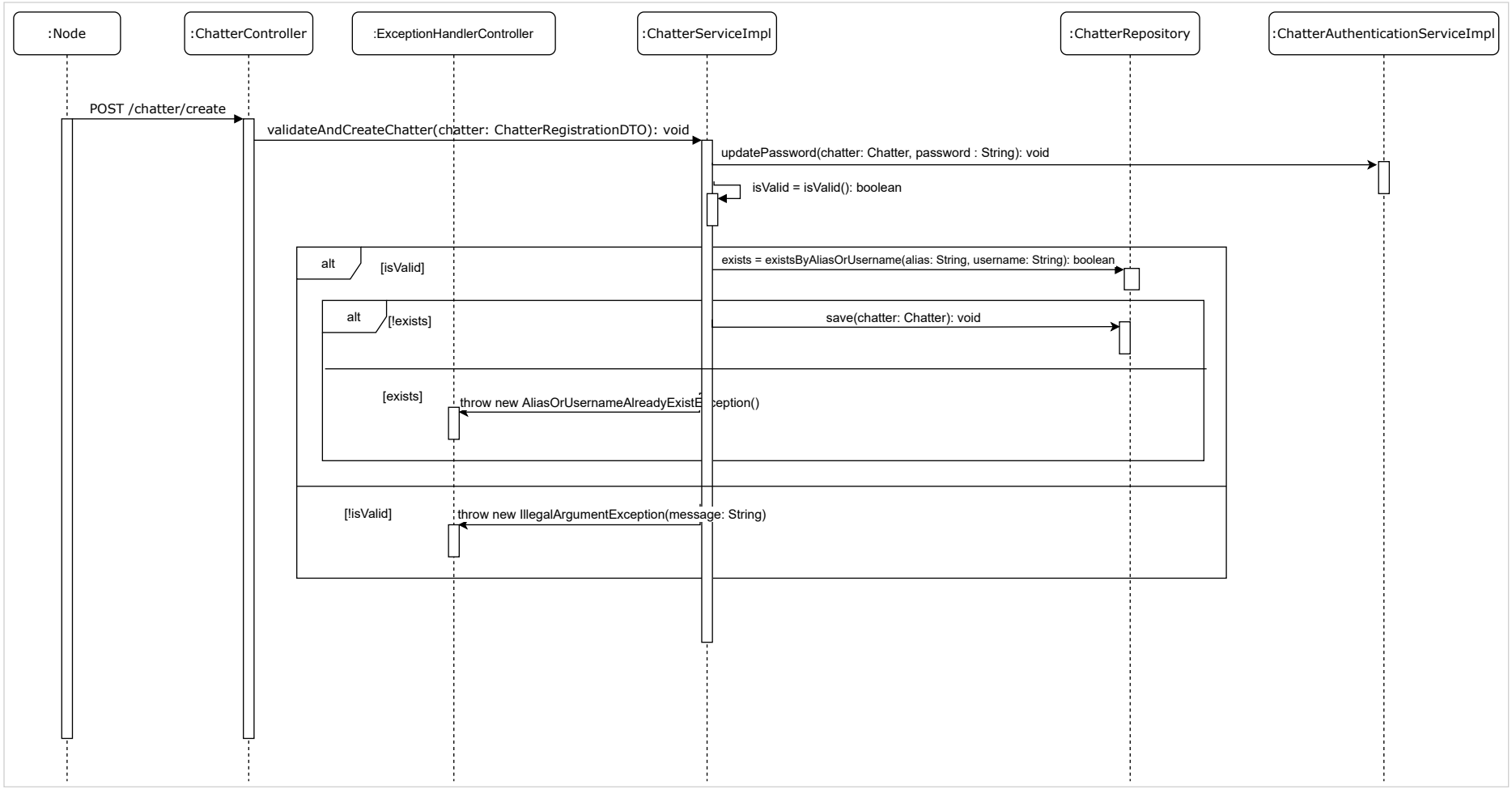
In dit hoofdstuk zijn sequence diagrams opgenomen om de flow van de Chatter Directory Server inzichtelijk te krijgen.

8.4.1. UC Manage account

Er zijn twee sequence diagrams opgesteld ter ondersteuning van de usecase manage account.

Registreren gebruiker

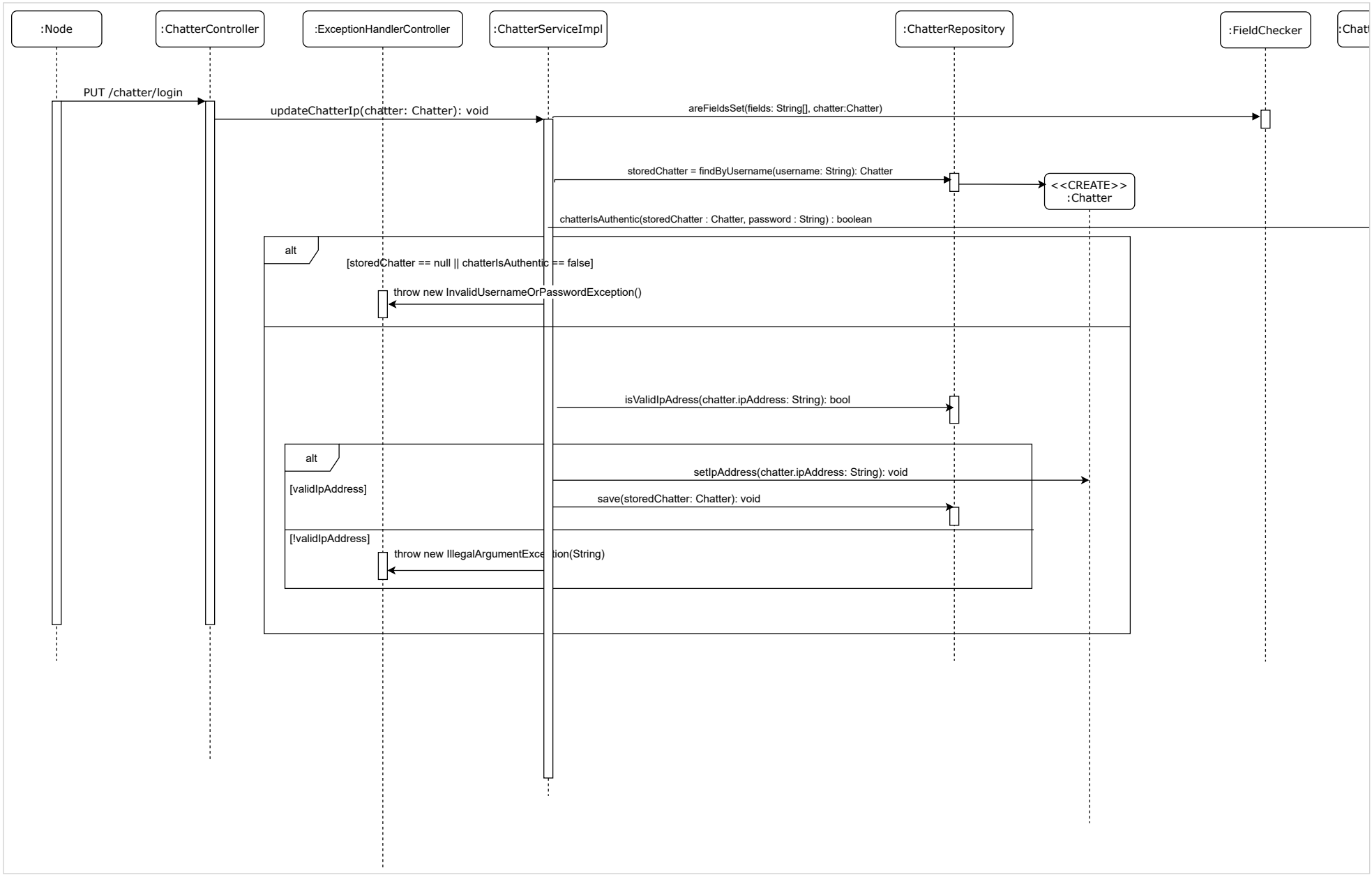
In Figuur 41 is het sequence diagram opgenomen voor het registreren van een nieuwe gebruiker.



Figuur 41: SD: Registreren gebruiker

Login

In Figuur 42 is het sequence diagram opgenomen voor het inloggen van een gebruiker. Hierbij wordt het IP-adres van de gebruiker geüpdatet zodat berichten naar het juiste IP-adres gestuurd worden.

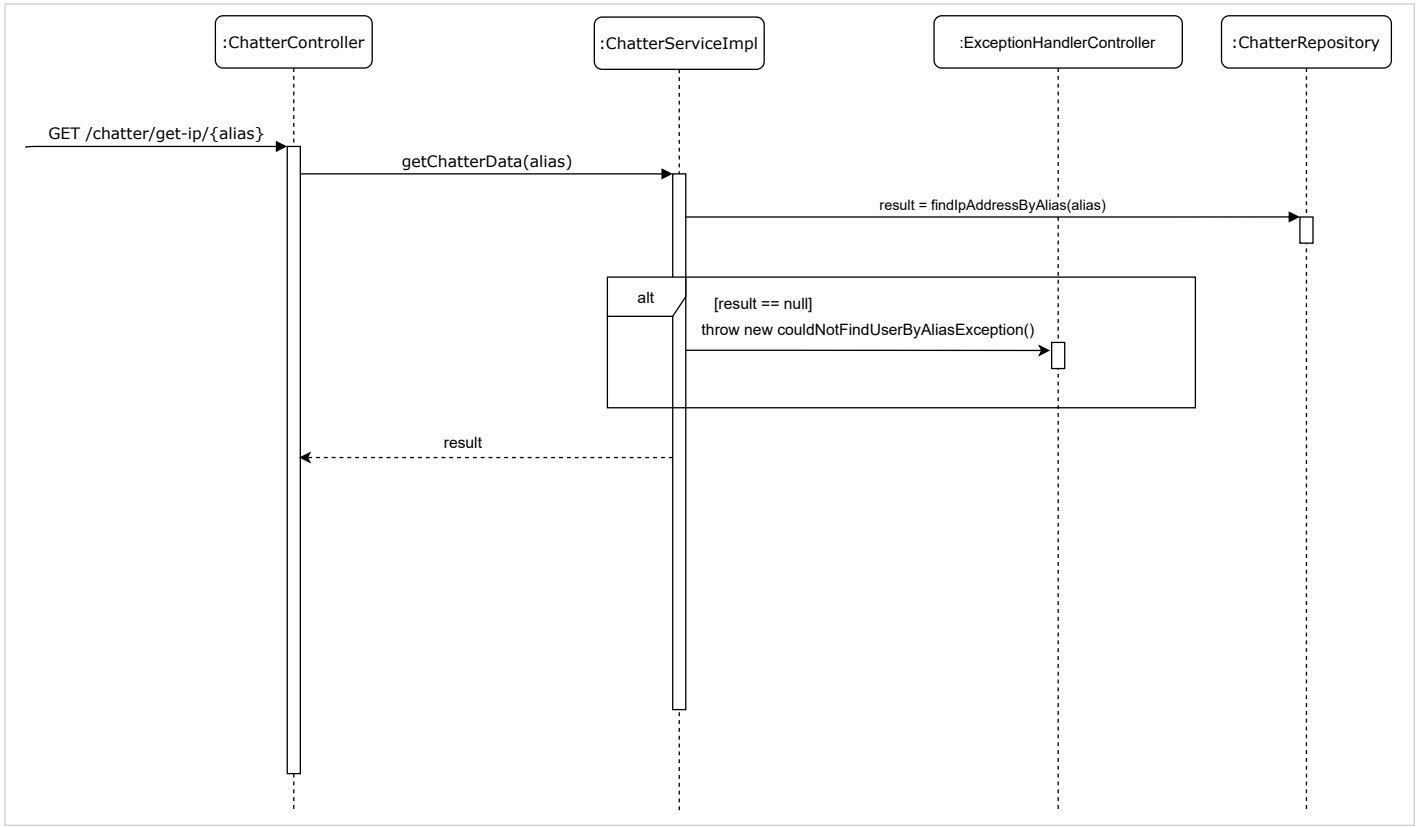


Figuur 42: SD: Update IP

8.4.2. UC exchange message

Get chatter IP address

Wijziging 11 juni: omgetoverd naar inloggen



SD 1: Get IP

8.5. Ontwerpbeslissingen voor het deelsysteem

In dit hoofdstuk zijn ontwerpbeslissingen opgenomen die betrekking hebben op de implementatie van de Chatter Directory Server.

Wijziging 11 juni: extra beslissingen toegevoegd

8.5.1. Gebruik database bij uitvoering testen

Beslissing	Beschrijving
Probleem/Issue	De database query's moeten getest worden met een database. De vraag is met welke database er getest wordt.
Beslissing	Er is gekozen om met een replica van de productiedatabase te testen. Dit houdt in dat de database properties hetzelfde zijn op de database naam na. Het overschrijven van de properties is gedaan door middel van een annotatie in de tests.
Alternatieven	• Het testen op een in memory database • Het testen op de productiedatabase
Argumenten	• Het is niet verstandig om op de productiedatabase te testen. • De in memory database verschilt sterk van de Mongo database. Er kan dus beter met een Mongo database getest worden.
Gemaakt op	26 May 2020 .

Beslissing	Beschrijving
Gemaakt door	@ Yuri Ruler

Tabel 28: Beslissing: Gebruik database bij uitvoering testen

8.5.2. Exception handlers

Beslissing	Beschrijving
Probleem/Issue	Hoe worden exceptions afgevangen in de code waardoor er een correcte response gegeven wordt.
Beslissing	Er is gekozen om een exception handler toe te passen. De handler is opgenomen in het bestand controllers/ExceptionHandlerController.java.
Alternatieven	• Het geven van een HTTP response vanuit de code.
Argumenten	• Door gebruik te maken van een exception handler heeft de code geen verstand van het protocol dat gebruikt wordt. • Er kan gemakkelijk een exception handler aangemaakt worden alle mogelijke excepties.
Gemaakt op	26 May 2020 .
Gemaakt door	@ Yuri Ruler

Tabel 29: Beslissing: Exception Handling

8.5.3. Deployment

Beslissing	Beschrijving
Probleem/Issue	Op wat voor server wordt de Chatter Directory Server gehost.
Beslissing	Er is gekozen om de server te hosten op de default Tomcat server.
Alternatieven	• Er zijn geen alternatieven overwogen.
Argumenten	• Spring maakt standaard gebruik van Tomcat server. Hiervoor hoeven geen instellingen aangepast te worden. Omdat er geen bijzonderheden zijn is er gekozen om dit bij de standaard te laten.
Gemaakt op	26 May 2020 .
Gemaakt door	@ Yuri Ruler

Tabel 30: Beslissing: Deployment

8.5.4. Password hashing

Beslissing	Beschrijving
Probleem/Issue	Hoe kunnen wachtwoorden veilig worden opgeslagen, zodat bij een database inbraak de wachtwoorden van gebruikers die zij ook gebruiken op andere servers niet gecomprimeerd zijn.
Beslissing	Er is gekozen om in de chatter directory server de wachtwoorden te salten en hashen met een langzaam algoritme (BCrypt).
Alternatieven	• Er is overwogen om wachtwoord hashes en salts op te slaan op de server. Maar het uitvoeren van het hashen en salten client side te doen.

Beslissing	Beschrijving
Argumenten	• Het hashen van wachtwoorden betekend dat de opgeslagen wachtwoorden niet gedecrypt kunnen worden, maar wel gebruikt worden om te vergelijken met opnieuw gehashte wachtwoorden. • Het salten en gebruik van een langzaam hashing algoritme zijn uitbreidingen op hashing, zodat er geen gemakkelijke massale pogingen gedaan worden om naar de wachtwoorden te gokken tot de goede gevonden zijn via rainbow table of dictionary attacks. • BCrypt als specifiek hashing algoritme heeft het voordeel dat de hoeveelheid van hashing iteraties later verhoogt kan worden. Dit betekent dat als met nieuwere snellere computers het algoritme niet langzaam en dus veilig genoeg is, de wachtwoordhashes later geupdate kunnen worden, de implementatie hiervoor hoeft niet nu uitgewerkt te worden. • Het uitvoeren van het hashen en salten client side heeft het nadeel dat er eerst een extra call naar de server gedaan moet worden om de salt op te halen. Het voordeel hiervan zou zijn dat de wachtwoorden over het netwerk niet makkelijk achterhaald kunnen worden, maar de betere standaard oplossing hiervoor is end to end encryptie / SSL. Als je salts client side zou willen opslaan zou je net zo goed gebruikers forceren om veilige rng wachtwoorden te gebruiken en password hashing over te slaan.
Gemaakt op	02 Jun 2020
Gemaakt door	@ Gino vd Bund

Tabel 31: Password Hashing

No labels

2 Comments

Yuri Ruler

- Inhoudsopgave een kopje geven
- Niet alle afbeeldingen/tabellen hebben een caption
- Aantal spelfouten
- Geen versiebeheer aanwezig
- Vrijwel alle kopjes staan in het Engels, dit consistent houden. Liefst alles in het Nederlands

Gestopt met reviewen aangezien het document niet voldoet aan de ICA Controlekaart.

Coen Hoogduin

- Done
- Done
- TODO
- TODO
- TODO