Björn Kirschner
Sebastian Schleemilch
Stefan Smarzly

Technische Universität München

GetInTUM - Door Access System

Final Documentation for Android Practical Course (F13)

July 11, 2015

Björn Kirschner

Sebastian Schleemilch

Stefan Smarzly

Abstract: blabla ...

TODO!!!

Contents

1 Introduction **2**

2 Project Team **4**
2.1 Björn Kirschner 4
2.2 Sebastian Schleemilch 4
2.3 Stefan Smarzly 4

3 Action Planning and Scheduling **4**

4 Project Flow **7**

5 Detailed project description **7**
5.1 Architectural Design 7
5.2 Registration with Back-end 8
5.3 Authentication between Smartphone and NFC Transceiver 8
5.4 Design Alternatives 8
5.5 Hardware Setup 9
5.6 Back-end 9
5.7 Protocols for Authentication between Smartphone and NFC Reader . 10
5.7.1 Public-Key Cryptography 10
5.7.2 SRFID: Extended Randomized Hash Lock 11

6 Implemented Functions **12**

7 Test Plan **12**

8 Conclusion **12**

9 Tutorial **13**
9.1 Install and Launch Backend Server 13
9.2 Install and Launch NFC Terminal 14
9.3 Use the GetInTUM Android Application 14

---

1 Introduction

This paper presents the final of *GetInTUM*, a solution to control access to university buildings via a mobile device communicating with a receiver at the door. This communication works via near field communication (NFC). Main contribution of the presented project is an Android application embedded in a holistic system which provides simple door access capabilities.

Problem description

Full-time students at TU München often face the problem that courses or seminars take place on a Saturday or a Sunday. Since buildings at the university campus are generally closed at weekends, students have to ring for the security personnel to open the door. But due to the number of students entering and leaving university buildings, door wardens hardly ever check the legitimacy of any entrant's business, i.e. if they really are students. This situation is unsatisfactory for all participants:

- Students needlessly have to wait for someone to open the door.
- Door wardens get distracted from more important work.
- Door wardens and university authorities cannot guarantee that entering people are students (or authorised persons in general).

Solution

In order to overcome afore-mentioned problems, this paper presents a solution which allows students to authenticate at a door system and are granted or denied access automatically. This approach helps to...

- reduce possible waiting times for students,
- unburden door wardens,
- and guarantee that only students and other authorised persons are granted access.

A few areas in TUM buildings already use an access mechanism via the student card. But for security reasons this student card solution is not ideal. Because the card only identifies via an imprinted unique ID, reading and duplicating the card is possible. For this reason we developed a smartphone solution which implements state of the art security mechanisms. This solution has the potential to not only be used for main door entrances to the university but also to control access to chairs or other restricted areas. It is an overall solution, relying solely on the user's smartphone.

Structure of this paper

The following chapter motivates the need for such a door access solution and sketches the project idea. Chapter 5.1 then details the proposed technical architecture. Due to a several possible approaches, chapter 5.4 justifies our design decisions. Finally, chapter 2 introduces all team members and chapter 3 outlines a project plan. Anpassen

2 Project Team

2.1 Björn Kirschner

Graduated in Information Systems at TU München. Is currently studying Information Sciences with a focus on IT security. Hence, secure communication and protocol design of the door access solution proposed in this paper will be of particular interest to him.

2.2 Sebastian Schleemilch

Made his bachelor in Mechatronics at the FAU Erlangen-Nuremberg. Right now, he is studying Automotive Software Engineering at TUM in the third out of four semester. He is excited about the combination of software and the interaction with physical problems, which also explains his previous education steps so far. Since this project will also be a mixture of pure software problems and the physical frontend (in this case: opening the door), Sebastian fits quite well into it.

2.3 Stefan Smarzly

Graduated in Informatics at TU München. Currently, Stefan is studying Informatics in the Master program at TUM with an emphasis on Robotics, Distributed Networks and Security. He is keen on working with Distributed Embedded Systems and to secure the communication between the devices. This project provides ideal conditions to apply his knowledge about deploying a secure Distributed System that physically interacts with its environment.

3 Action Planning and Scheduling

noch nichts gemacht hier...

...

Info zur tatsäc
lichen Dauer i
Chart

The Gantt chart in chapter gives a brief overview of all tasks necessary for the proposed project. It lists important milestones, estimates task lengths and assigns them to the three team members[1].

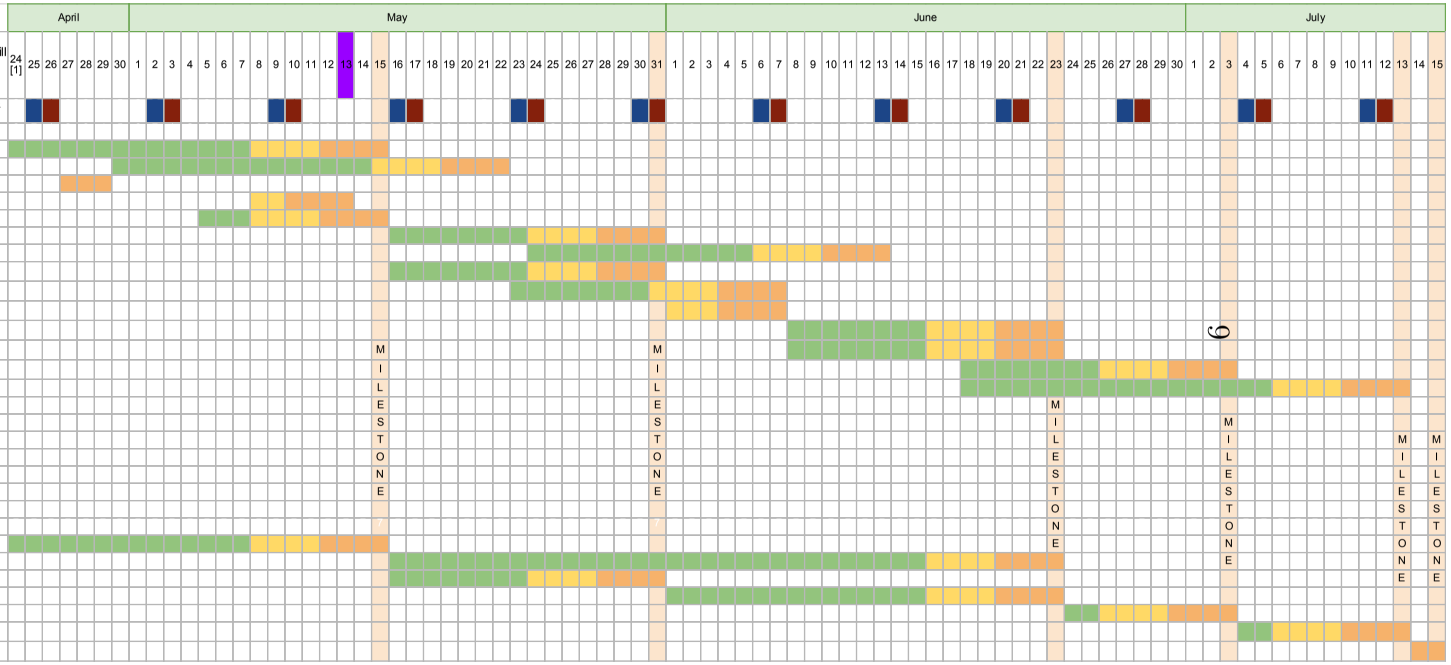
The chart includes tasks since the start of the project. These already finished steps focused on conceptual and organisational aspects. Most importantly, we obtained approval of important TUM stakeholders:

- Mr. Bernhofer from the TUM IT Management, responsible for identity management and connections to external IT systems. He provided us with information how to access central TUM services (Active Directory, Token management for TUMonline, ...).

[1] In this context, 'St' stands for 'Stefan', 'Se' for 'Sebastian' and 'Bj' for 'Björn'

- Mr. Recksiegel from the TUM physics department, who is the head behind an existing student card to NFC solution running in the physics building. He promised help with hardware issues and to finally push the integration of our smartphone to NFC solution into the already existing systems.

done	Category	Start	End	Doing	details	Days till End date
						Sat & Sun
	Tasks					
d	Plan	24-Apr-15	15-May-15	all	**Conceptual planning (architecture, ...)**	done
	Vision	4/30/2015	22-May-15	St, Se	**Explore technical NFC solutions (which reader, ...)**	9
d	Orga	27-Apr-15	29-Apr-15	all	**Get backup from TUM IT Management**	done
d	Orga	08-May-15	13-May-15	all	**Get backup from physics department (pilot)**	done
d	Plan	05-May-15	15-May-15	all	**Write idea report**	done
	Mech	16-May-15	31-May-15	St, Se	**Build NFC reader prototype (raspberry)**	18
	Code	24-May-15	13-Jun-15	St	**Implement reader component**	31
	Code	16-May-15	31-May-15	Bj	**Implement backend server**	18
	Code	23-May-15	07-Jun-15	Se	**Implement Android application**	25
	Vision	01-Jun-15	07-Jun-15	all	**Make final architecture decisions (protocols, ...)**	25
	Integrate	08-Jun-15	23-Jun-15	Bj, Se	**Integrate smartphone - backend communication**	41
	Integrate	08-Jun-15	23-Jun-15	St, Se	**Integrate smartphone - NFC reader communication**	41
	Integrate	18-Jun-15	03-Jul-15	all	**Overall integration**	51
	Test	18-Jun-15	13-Jul-15	all	**Testing and bugfixing**	61
	Milestones					
d	0	24-Jun-15	15-May-15		**Idea report and commitment of TUM stakeholders**	done
	1	16-May-15	23-Jun-15		**Communication Smartphone - NFC working**	41
	2	16-May-15	31-May-15		**Backend server (prototype) working**	18
	3	01-Jun-15	23-Jun-15		**Smartphone application (prototype) working**	41
	5	24-Jun-15	03-Jul-15		**Integration done**	51
	6	04-Jul-15	13-Jul-15		**Testing done**	61
	7	14-Jul-15	15-Jul-15		**Final submission**	63



4 Project Flow

any problems? Differences to the planned functionality?

TODO

problems: hatten wir ja ein paar: Zusammenspiel mit anderen Lesegeräten (Legic), ...

Differences to the planned functionality: gibt's bei uns eher nicht, oder?

Stefan: Ne, eigentlich sogar noch mehr Funktionalität und erhöhte Sicherheit/Anonymität als spezifiziert.

5 Detailed project description

5.1 Architectural Design

Vorschlag: Architectural Design nennen und vieles kürzen

GetInTUM's basic architecture consists of the following parts:

- The smartphone, including a NFC chip as well as an internet connection to register with the service. An internet connection is only be needed once during initialization of the system or when creating a new key pair in case of revocation. The smartphone must be able to create an asymmetric key pair and securely store it persistently.
- The Door Access System, also including a NFC chip which can send and receive. A simple NFC reader would not suffice as it would be vulnerable to replay attacks. The door system needs access an intranet in order to check if the student/employee has access to the specific area and to get other information for realizing security features like encryption of the communication. Therefore, some kind of computer is needed, interacting with the NFC hardware as well as with the backend.
- A backend system, which checks student ID, the corresponding TUMOnline token and fetches the generated public key from the smartphone. It finally stores the public key of every registered student. At every door access attempt it checks the TUM active directory if the person is still a student.
- The TUMOnline system, which provides a secure way to confirm that a user is real. This is done via a token that the user has to activate in TUMOnline.
- The TUM active directory (AD), which has enrolment information about every student and can confirm the status of a person.

The last two items are important for the understanding of the whole solution. They were of course already existing and were not altered for this project.

bis hier sollte jetzt passen

Figure 1 gives a holistic overview of the basic architecture (based on public-key cryptography as discussed in section 5.7.1):

brauchen wir die Klammer hier?

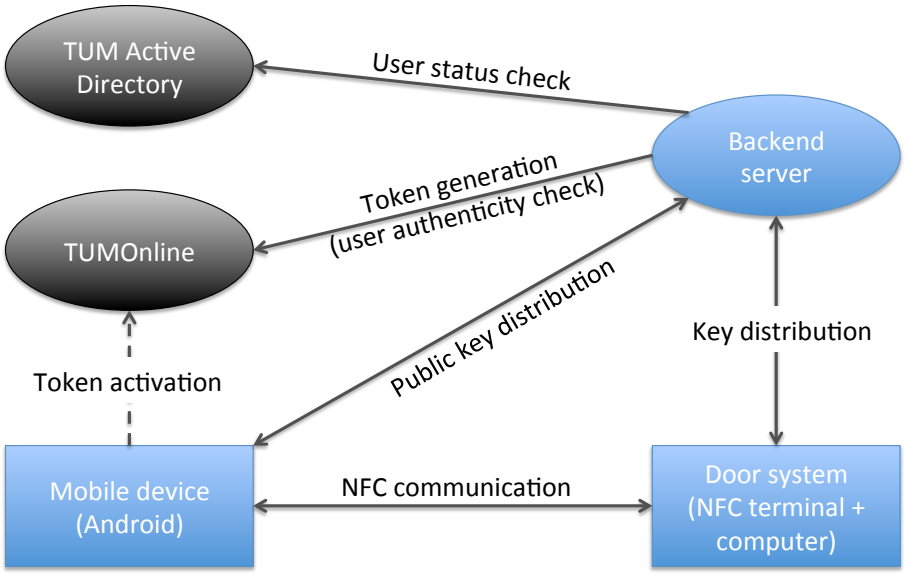


Figure 1: Architectural overview of the *GetInTUM* solution.

5.2 Registration with Back-end

To initialize the system, the user has to contact the back-end system once for authentication. The user provides his ID along with the user token that is generated by TUMOnline. The back-end system can check if the proposed access token of the user is valid or not via the TUMOnline Webservice API. If the user has sent the right access token, the back-end system will demand the public part of the user's asymmetric key pair (which has been created on the user's smartphone before starting the communication). In this step, the server will take particular care that the authentic user is in possession of the private part of the transmitted public key and doesn't pretend to be a different identity. All connections to the back-end are planned to be secured by TLS.

5.3 Authentication between Smartphone and NFC Transceiver

After the user has registered with the back-end system, he can communicate with the NFC reader. A three-way handshake assures a secure connection between smartphone and NFC reader. In order to verify the soundness of the procedure, in particular the authenticity of the communication partner, the NFC reader (i.e. the system behind it) will fetch the public key of the user who wants to authenticate from the back-end system. Furthermore, the Door Access System will check if the user has access to the requested area. It will finally unlock the door if all checks were positive.

5.4 Design Alternatives

Das und das vorherige Kap mergen?

alternativen rauß? Verwen detes Protoko muss aber dri bleiben

In the previous section, the overall architecture is presented without going into detail about the used hardware, the back-end or how secure channels are implemented. This section demonstrates competing drafts for the concrete design of central parts of our project. Due to dependent advantages and drawbacks, different approaches are reasonable for the same goal. These are discussed as follows.

5.5 Hardware Setup

A major part is a reliable communication link between an Android smartphone and a NFC reader. To reach this goal, the NFC reader must support common NFC standards (e.g., ISO/IEC 14443) and provide stable transmission properties. Therefore, we examine two different hardware setups to finally get a proper solution:

- Raspberry Pi with Explore-NFC PN521 reader cape by NXP [2] or PN532 compliant cape.
- Smartphone with NFC capabilities and Android 4.4 or newer.

Using the Raspberry Pi as underlying platform and a NFC transceiver, a cheap and powerful setup is possible. Providing a rich interface to interact with the environment, it allows an easy integration into existing door electronics. For communication, either the vendors of these NFC capes provide SDKs or free libraries exist to abstract low-level functionality. However, the protocol layer is often incomplete and needs tweaks. This might influence the stability in the communication link.

On the other side, a smartphone with NFC transceiver and Android 4.4 (or higher versions) offers all required APIs in a simple way to exchange data between two Android smartphones over NFC. Additionally, integrated capabilities like Wifi or Bluetooth allow enhanced applications. The downsides of this approach are much higher costs, missing interfaces to interact directly with most door electronics and required user interaction for certain tasks (e.g. OS upgrade).

5.6 Back-end

As introduced in section 5.1, a central part of the overall project is a reliable back-end. Its task is either to store and distribute public keys for the public-key cryptography implementation (see section 5.7.1) or to manage secret hashes as dynamic keys for the hash lock implementation (see section 5.7.2). It is also responsible for authentication of a new user using TUM Web Services to prevent identity theft.

We intend to use Node.js as technology for the back-end. Referring to projects it is already being used for, it provides a reliable and fast service. The exposed web service API allows a fast integration with Android and provides a secure communication channel using standards like TLS. For persistent storage, the key-value database MongoDB is the preferred choice.

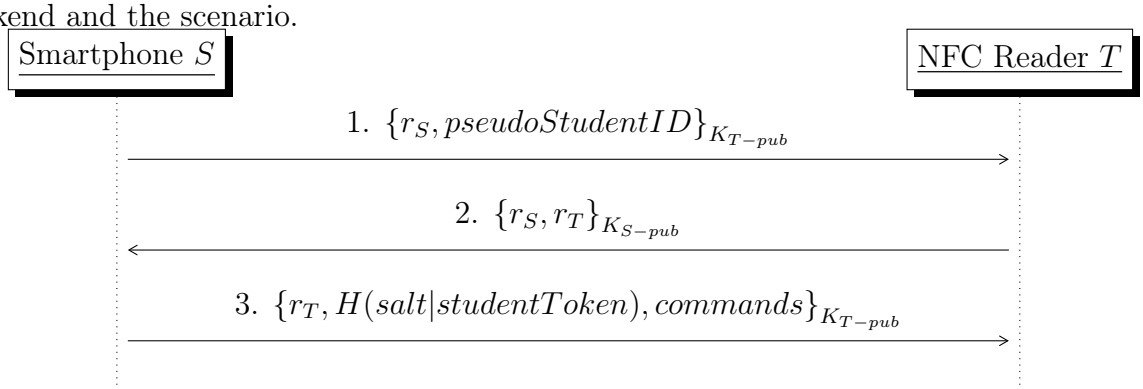
[2] `http://www.nxp.com/demoboard/PNEV512R.html`

5.7 Protocols for Authentication between Smartphone and NFC Reader

A critical aspect in our project is the communication between a smartphone S and the NFC component T. It is important that it is safe so that no unauthorized party can easily gain access or eavesdrop sensitive information like the student ID. By placing the reader's antenna in the inner side of the door, it should be safe from physical threats from outside. To ensure a secure communication on top of the data carrier, we plan to use one of the protocols introduced in this section. Which one to finally choose depends on multiple aspects like the capabilities of the NFC hardware (e.g. processing power), Android support or fault tolerance. This needs some testing done during the project.

5.7.1 Public-Key Cryptography

The first proposed protocol uses a Public Key Infrastructure to establish secure communication links between the participants. For authentication on both sides and mitigation of several attack scenarios, it uses some ideas of the Needham-Schroeder-Lowe Public Key protocol. Lowe contributes an important security fix we use. The scheme is enhanced to provide further security features specific for the utilised backend and the scenario.



Explanation:

1. $S \rightarrow R$:
   - S generates a random number r_S and sends it to T together with the $pseudoStudentID$, encrypted with the reader's public key K_{T-pub}.
   - The official student ID is replaced by the pseudonym $pseudoStudentID$, which is associated with the official ID of the student in the backend. Like this, a MITM attack would not allow to directly gain information about the requesting person's ID. Furthermore, this pseudonym association can be changed regularly.
2. $S \leftarrow R$:
   - The reader T looks up the public key K_{S-pub} based on the pseudo student ID. If it doesn't exist, further communication stops here.

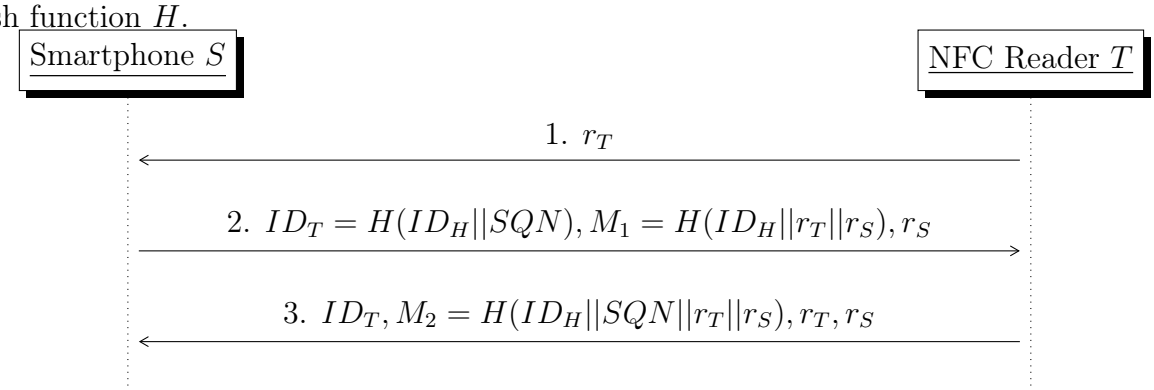
- On success, T generates a random number r_T and sends it back to S together with r_S and a hash of its own public key, encrypted with the user's public key K_{S-pub}.

3. $S \rightarrow R$:

   - If S receives a valid answer in step 2, it should be fine to proceed.
   - S sends r_T, a salted SHA-256 hash of $studentToken$ and data to T, encrypted with K_{T-pub}.
   - Salted SHA-256 student token $H(salt|studentToken)$: as the random salt is unique per user and only known to the back-end and the Android client, the terminal does not touch critical data. Still, by comparing the hash values received by the back-end on retrieval of the user's public key, and the one sent by the smartphone in this step, the identity of the user is proven indirectly.

5.7.2 SRFID: Extended Randomized Hash Lock

A second interesting protocol is the hashed-based security scheme by Walid I. Khedr[3]. This might be an alternative to the public key based system, because it is more lightweight. For cryptography on both sides, it only needs a strong one-way hash function H.



Details will not be presented here, because it is discussed in detail in the referred paper above. Nevertheless, some aspects have to be highlighted in the scope of this project:

- In contrast to the original protocol, the smartphone S also generates a random number r_S beside r_T from the reader. Consequently, it is more likely to identify a replay attack by a fake reader. This could optionally be reported by the app.
- The sequence counter SQN is incremented both on the back-end and the smartphone for each handshake. Like this, the identifier of the smartphone ID_T, derived from the shared secrets ID_H and SQN, is not the same on every connection to mitigate tracing. Still the reader / back-end is able to identify the communicating party, because it also knows the shared secrets

[3] `http://www.sciencedirect.com/science/article/pii/S1110866513000054`

ID_H and SQN. In case of a desynchronization attack, the relation can be re-established by asking the back-end to present the unique *studentToken* (from TUM Webservice).

6 Implemented Functions

7 Test Plan

Test coverage, Test methods, Test responsibilities — struktur!

...Das Ganze auf den Funktionen aus den letzten Kapiteln basieren lassen....

Due to the strict separation of the components *backend*, *smartphone* and *NFC terminal* testing could partly be initiated in early stages of the project.

Particularly the *backend* component was required to quickly reach a robust state because both other components heavily rely on its functioning. In order to guarantee this robustness, development followed a test-driven approach from the beginning. Functionality required in the *backend* was first defined in a specifications document. Subsequently test cases based on these specifications were written using *vows.js*[4]. Only then did we begin to actually implement the defined functionality.

... — ...

Test scenarios: high level
- user has TumOnline account, is student
- user has TumOnline account, not student anymore
- user has no TumOnline account

low level:
- testing all possible mistakes: requests with wrong / missing arguments
- sending broken / incomplete packets
- interrupting connections
- possible failures of involved systems. E.g. TumOnline token max reached, AD not responding, general connection problems, ...

...

8 Conclusion

Outlook?

- Can be adapted for more advanced rules. Do more fine-grained access control. E.g. doors at chair X only open if someone from chair X is allowed to.

- Die Lösung im Zusammenspiel mit dem Legic-Zeugs beschreiben. Das mit zwischen den zwei Readern hin- und herschalten. Wobei Teile dieser Problembeschreibung schon ins Kapitel "project flow" müssen.

- Könnte man in die TUM-Campus-App integrieren...

[4] Vows: Asynchronous behaviour driven development for Node. http://www.vowsjs.org

9 Tutorial

Before actually using the *GetInTUM* solution on a smartphone, both the *backend* and a *NFC terminal* have to be running. Hence this chapter starts with installation instructions for *backend* and *NFC terminal*.

General note: a command line starting with # indicates a root shell, a $ symbol indicates one without super-user privileges.

9.1 Install and Launch Backend Server

The server was implemented based on the JavaScript (JS) technologies *Node.js* and *ME(A)N.js* (M = *MongoDB*, E = *express.js* [web server], A = *angular.js* [currently not used], N = *Node.js*). Hence it requires a running *MongoDB* database and *Node.js* to be installed. This chapter serves as a recommendation for starting the server. Experienced users may also install some components locally (without the $-g$ option) or omit some parts.

1.) Install *Node.js*.

2.) Install *npm*, the Node Package Manager.

3.) Download *MongoDB*.

4.) Start *MongoDB* via

```
$ <path-to-mongodb>/bin/mongod
```

5.) Install *bower*, a package manager.

```
# npm install -g bower
```

6.) Install *grunt*, a JS task runner for automation.

```
# npm install -g grunt-cli
```

7.) Install *vows*, a JS framework for writing and executing test suites. This is only necessary if tests need to be run.

```
# npm install -g vows
```

8.) Install all *Node.js* dependencies of the *backend* server. *npm* automatically loads all dependencies defined in *package.json* and stores them locally.

```
$ npm install
```

9.) Finally, **run the server** via the *grunt* client (optionally with the environment variable *NODE_ENV=test*).

```
$ grunt
```

(10.) Optionally, run the *vows* test suite (change to */test* directory first; *NODE_ENV=test* is required for some unit tests).

```
$ vows interfaces-test.js --spec
```

9.2 Install and Launch NFC Terminal

...

9.3 Use the GetInTUM Android Application

...