



Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Automation and Applied Informatics

Soma Lucz

Developing a graph-based, domain-specific social network

MASTER'S THESIS

<https://www.diplomatiq.org>
<https://app.diplomatiq.org>
<https://api.diplomatiq.org>

Supervisor

Márk Asztalos, PhD

Budapest, 2020

Contents

Contents	iii
Hallgatói nyilatkozat	ix
Kivonat	x
Abstract	xi
1 Introduction	1
1.1 Context	1
1.2 Problem statement and requirements	2
1.3 Objectives and contributions	3
1.4 Structure of this thesis	4
2 Preliminaries and related work	5
2.1 The Model United Nations framework	5
2.1.1 Introduction	5
2.1.2 History	6
2.1.3 MUN in numbers	7
2.1.4 Networking within Model United Nations	7
2.1.5 Administration of a Model United Nations conference	8
2.1.6 Personal experience: Budapest International Model United Nations	13
2.2 Social networks	13
2.2.1 Introduction	13
2.2.2 Connection with graph theory	13
2.2.3 Social networking services	13
2.3 Examples for domain-specific social networks	15
2.4 MyMUN	16
2.4.1 Introduction	16
2.4.2 Organizational features	16
2.4.3 Features for participants	18

2.4.4	Business model	21
2.5	MUNPlanet	22
3	Chosen technologies	23
3.1	Spring Boot	23
3.2	Angular	24
3.3	Neo4j	24
3.3.1	The property graph data model	25
3.3.2	The Neo4j graph database	25
3.3.3	Cypher	26
3.4	Git and GitHub	27
3.5	Microsoft Azure	27
4	Building and securing a company-level, production-grade infrastructure	29
4.1	Introduction	29
4.2	Naming	30
4.3	Brand	30
4.4	Domain name	31
4.4.1	Overview and purchasing the domain name	31
4.4.2	Security	31
4.5	TLS infrastructure	31
4.5.1	Overview	31
4.5.2	Purchasing a TLS certificate	32
4.5.3	Security	32
4.6	Email infrastructure	33
4.6.1	Overview and usages	33
4.6.2	Sending and receiving individual emails	33
4.6.3	Transactional and marketing emails	34
4.6.4	Securing SendGrid access	34
4.6.5	Securing emails sent by Diplomatiq	34
4.7	Source code management	35
4.7.1	Registering the Diplomatiq GitHub organization	35
4.7.2	Standardized, organization-wide documents and configurations	36
4.7.3	Development and release model	38
4.7.4	Automated dependency management	39
4.7.5	Continuous integration and continuous delivery	40
4.7.6	Code analysis with SonarCloud	41
4.8	Node Package Manager (NPM)	42
4.8.1	Overview	42
4.8.2	Creating and securing an organization	43
4.9	Communication and social media presence	43

4.9.1	Gitter	43
4.9.2	Slack	44
4.9.3	Facebook	44
4.10	Procuring the Neo4j database software	44
4.10.1	Licensing overview	44
4.10.2	Acquiring the startup license	45
4.11	Building and securing the server infrastructure	45
4.11.1	Platform overview	45
4.11.2	Creating a directory for Diplomatq	45
4.11.3	Naming conventions	47
4.11.4	Creating a subscription and a support subscription	47
4.11.5	Structuring resources	47
4.11.6	Securely storing secrets and credentials	48
4.11.7	Setting up the infrastructure for the website	49
4.11.8	Setting up the infrastructure for the front end application	52
4.11.9	Setting up the infrastructure for the back end application	52
4.11.10	Setting up the database infrastructure	54
4.11.11	Networking and security	55
4.11.12	Resource locking	56
5	Overview and development of supportive libraries	59
5.1	crypto-random	59
5.1.1	Introduction	59
5.1.2	Features and operation	60
5.1.3	Discrete uniform distribution	60
5.1.4	Testing	61
5.2	convertibles	61
5.2.1	Introduction	61
5.2.2	Features	61
5.2.3	Testing	62
5.3	resily	62
5.3.1	Introduction	62
5.3.2	Features	62
5.3.3	Reactive policies	62
5.3.4	Proactive policies	64
5.3.5	Testing	65
5.4	project-config	65
5.5	eslint-config-tslib and eslint-config-angular	66
6	Overview and development of the Diplomatq application	67
6.1	Introduction	67

6.1.1	Personal goals and outline of this chapter	67
6.1.2	Vision	67
6.1.3	Target audience	68
6.2	Long-term feature goals against competitors	68
6.2.1	Introduction	68
6.2.2	Features for entities organizing conferences, associations, and clubs	69
6.2.3	Features for entities participating in conferences	72
6.2.4	Features for non-MUN entities	72
6.3	Specification and implemented features	72
6.3.1	Introduction	72
6.3.2	Registration	73
6.3.3	First and second factor of authentication	73
6.3.4	The Dashboard	73
6.3.5	The navigation bar	74
6.3.6	Applying to a conference	74
6.3.7	Organizing a conference	74
6.3.8	Cancelling a conference application	75
6.3.9	Cancelling a conference	75
6.3.10	Password change	75
6.3.11	Account deletion	75
6.3.12	Password reset	76
6.4	Development and implementation details of the back end	76
6.4.1	Introduction	76
6.4.2	Setting up an empty Spring Boot application	76
6.4.3	API concepts	76
6.4.4	Implementation details of the API	78
6.4.5	Documenting the API	79
6.4.6	Domain data model	79
6.4.7	Layering the application	81
6.4.8	Error handling	81
6.4.9	Configuration	82
6.4.10	Packaging and deployment	82
6.5	Development and implementation details of the front end	82
6.5.1	Introduction	82
6.5.2	Creating an empty Angular application	83
6.5.3	Structuring the application	83
6.5.4	API access and error handling	83
6.5.5	Packaging and deployment	84
7	Security of the Diplomatiq application	85
7.1	Introduction	85

7.1.1	Motivation	85
7.1.2	Scope and approach	85
7.2	Implemented cryptographic structures	86
7.2.1	DiplomatiqueAEAD	86
7.3	Applied web platform security measures	88
7.3.1	Cross-Site Scripting (XSS)	88
7.3.2	Content Security Policy (CSP)	88
7.3.3	Cross-Origin Resource Sharing (CORS)	88
7.3.4	Feature Policy	89
7.3.5	Referrer Policy	89
7.3.6	Preventing click-jacking	89
7.4	Authentication and authorization	89
7.4.1	Introduction, motivation, and terminology	89
7.4.2	Why not OpenID Connect or OAuth 2.0?	90
7.4.3	Requirements	91
7.4.4	Protecting endpoints	92
7.4.5	User authentication	93
7.4.6	Request authentication	95
7.4.7	Client version identifier	98
7.4.8	Client clock discrepancy	98
7.4.9	Signing requests	99
7.4.10	Implementing request authentication with Spring filters	100
7.4.11	Implementing controller authorization	102
7.5	Encrypting sensitive database data	103
8	Conclusion and future work	105
8.1	Summary of contributions	105
8.2	Future work	106
	Acknowledgements	107
	References	109
	Appendix	119
A	Lists	119
A.1	List of implemented projects	119
B	Tables	119
B.1	Email addresses configured for diplomatique.org	119
C	Figures	120
C.1	Server calls for elevating a session to password assurance level . . .	120
C.2	Server calls for elevating a session to multi-factor assurance level .	120

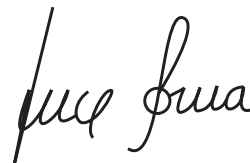
C.3	Operation of the base headers checker filter	121
C.4	Operation of the clock discrepancy filter	122
C.5	Operation of the signature verification filter	123
C.6	Operation of the authentication filter	124

Hallgatói nyilatkozat

Alulírott **Lucz Tamás Soma** szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2020. május 31.



Lucz Tamás Soma
s.k.

Kivonat Napjaink globalizált világának működésében kulcsfontosságú szerepet tölt be a diplomácia. Diplomátává válni hosszú folyamat, mely korai elhivatottságot kíván – gyakran középiskolás vagy egyetemista korban, világszervezetek munkájának tanulmányi célú szimulációjában való részvétellel kezdődik egy karrier. Egy leendő diplomata karrierjét támogatva nemcsak betekintést nyújthatunk az általa is formált közös jövőnkbe, de hosszú távon annak alakításában is részt vehetünk. Az összes leendő diplomata karrierjét tekintve a lehetőségek tárháza határtalan, az ezzel járó felelősség pedig hatalmas.

A *Model United Nations (MUN)* keretrendszerben világszerte évente több száz nagyságrendben megrendezett konferenciákon résztvevő középiskolás és egyetemista diákok az Egyesült Nemzetek Szervezete (ENSZ) mindennapi munkájának formális szimulációján keresztül tanulhatnak diplomáciáról, nemzetközi kapcsolatokról, világpolitikáról – kockázatmentes, tényekre és információkra alapozott vitakultúrát kultiváló környezetben, gyakran tapasztalt karrierdiplomata támogatásával.

A világ MUN-közösségének összefogására több szoftveres kísérlet is született. Ezek többnyire egy-egy problémára igyekeznek elszigetelt megoldást adni, így kapcsolatépítésre, konferenciák szakmai szervezésére, illetve rendezvények általános adminisztrációjára eltérő – gyakran házon belüli – szoftverek használatosak. Ezen alkalmazások nem kötik össze a közösség egészét, és nem adnak teljes megoldást az adminisztratív problémákra sem.

Dolgozatomban kifejtem, hogyan megtervezem, lefejlesztem, és webalkalmazásként publikusan elérhetővé teszem a *Diplomatiq* nevű, MUN-konferenciák szervezésére alkalmas közösségi hálózat funkcionalitásban kezdetleges változatát. A *Diplomatiq* hosszú távú célkitűzése az, hogy a diplomata elsődleges közösségi platformjaként nyújtson integrált megoldást az MUN-világban felmerülő adminisztratív problémákra.

A tervezés és fejlesztés teljes folyamata alatt fókuszban tartottam két alapvető szempontot. Az első szempont, hogy a rendszer „használatra kész” minőségben készüljön el, és később igény szerint bővíthető legyen további közösségi, adminisztratív, illetve valós idejű adatalemzési funkcionalitással. Ennek célja, hogy a szoftver a jövőben az MUN-szcénán kívül valódi diplomáciai alkalmazásokban is helyt tudjon állni. A második szempont – a tárolt személyes adatok érzékenysége, illetve a szoftver leendő alkalmazási lehetőségeinek figyelembevételével – az, hogy a rendszer már a kezdetektől modern, réteges, kriptográfiai biztosítékokat nyújtó biztonsági architektúrára alapozva készüljön el.

A rendszer tervezése és fejlesztése során a mérnöki szempontokon felül arra is figyelmet fordítottam, hogy a *Diplomatiq*, mint majdani vállalkozás az elvégzett munkámra egyszerűen ráépíthető legyen. A szoftver lefejlesztéséhez és publikációjához szükséges előfizetéseket, szolgáltatásokat és rendszereket mind olyan megfontoltsággal választottam ki és integráltam, mintha egy vállalkozást indítanék el.

Abstract Diplomacy plays a key role in the operation of today's globalized world. Turning into a diplomat is a long process and involves early dedication — careers often start in high schools or universities, by students taking part in academic simulations of various intergovernmental organizations' work. Supporting *a* prospective diplomat's career not only enables us to peek into the future through them, but in the long run, we can also take part in jointly shaping tomorrow's world. Considering *all* prospective diplomats' careers, the possibilities are endless, and the associated responsibility is immense.

The world of junior diplomats mostly consists of conferences — annually hundreds of them, worldwide — held within the framework called *Model United Nations (MUN)*. During these events, high school and university students formally simulate the everyday work of the United Nations (UN), which enables them to learn about diplomacy, international relations and world politics — in a risk-free environment, cultivating debates based solely on facts and information. These conferences are often attended by experienced senior diplomats as well, with the goal of supporting and educating the future generation.

There has been several software attempts for bringing the MUN community together. Most of these attempts solve one isolated problem of the collective at a time: social networking, organizing the professional part of conferences, and administering the actual events usually involves several different — often primitive, in-house — software. These applications neither link the community together, nor do they offer a complete solution to administrative problems of MUN events.

In this thesis I design, implement and publish an initial version of *Diplomatiq*, a social network software system for diplomats, suitable for organizing MUN conferences. The long-term goal of *Diplomatiq* is to provide an integrated administrative solution to the MUN world, while being the sole professional networking platform for its diplomat users.

During the whole process of the design and implementation, I focused on two key points. The first point is that the system should be implemented in production-grade quality, and it should be extensible with further social, administrative, and real-time data analytics features as needed. The goal of this is to eventually cover the needs of real-world diplomatic applications as well, outside the MUN scene. The second point — considering the sensitivity of stored personal information, and also potential future applications — is that the system should be implemented upon a modern, layered security architecture, which provides cryptographic assurances in terms of application and data security.

Besides engineering aspects, I also paid attention to being able to build *Diplomatiq* as a prospective company upon my work. Subscriptions, services and systems needed for the implementation, publication and production operation were chosen and integrated with the same amount of consideration as I was starting company.

Chapter 1

Introduction

1.1 Context

Diplomacy is the art and practice of conducting negotiations between nations and nationwide entities [1]. It is a complex system, where involved parties like governments and NGOs¹ engage in formal discussions, aspiring to *peacefully* influence the status quo of international relations along their interests. Parties are represented by selected, often professionally trained career diplomats, forming a diplomatic delegation.

Besides diplomacy, there are other tools for leveraging international relations. This set of tools, tactics and strategies is collectively known as foreign policy, and is usually directed by political leaders [2]. Foreign policy is often collated with diplomacy as a synonym, but the two are not identical. Diplomacy is a key instrument of foreign policy, and foreign policy is a superset of diplomacy. In order to achieve the objectives of a nation, tools of foreign policy can include espionage, threats, sabotages, wars, and other means of violence, as well as diplomacy.

Throughout this thesis, I consider diplomacy as the nonviolent elements of foreign policy: the system, methods and infrastructure of governments and NGOs peacefully interacting with each other, in order to influence international relations along their own objectives. Although most diplomacy materializes in confidence between parties, this thesis exists in the context of publicly conducted diplomacy, more narrowly in the context of the United Nations (UN), which — having 193 sovereign member states — is the largest intergovernmental organization in the world [3].

Being a powerful diplomat requires experience in various fields. Diplomats need strong organizational and leadership skills, as well as proficiency in written and oral communication for efficient negotiations. They must be able to stay rational and decisive in stressful

¹non-governmental organizations

situations, besides the capacity to quickly process and integrate information into their decisions [4]. These skills can be developed in specialized educational institutes, usually offering graduate programs [5].

Apart from professional programs designed to train already graduated career diplomats, there are other ways to gain diplomatic experience. One of these is taking part in academic simulations of the United Nations' everyday work. For high school and university students, the Model United Nations (MUN) framework² offers hundreds of independent conferences annually, worldwide [6]. On these few-day-long events, participants become diplomatic delegates. They are placed in UN-like committees and assigned countries to represent. Assignments are published in advance, along with the topics the committees will discuss. This enables delegates to perform research and develop their positions before the conference, usually staying true to the actual position of their represented country. During the conference, delegates discuss their positions in the committees, conforming to the formalities of the real-world United Nations, like western business attire and the method of moderated formal debate. By the end of the conference, each committee produces a formal, UN-like *resolution*: a document summarizing the results of the debate and formulating measures for resolving the international issues presented to the committee.

1.2 Problem statement and requirements

Since even a medium-sized conference welcomes hundreds of international students, who need accomodation, meals, personalized conference accessories like badges and placards, topics to debate, merchandise, and afterwork entertainment, an MUN conference is a heavy organizational burden, requiring months of preparation. Most conferences are driven, prepared, implemented, and executed by voluntary, unpaid students of an educational institution — a high school or a university —, as an extracurricular activity, with additional help of their teachers. Professional event planners, IT and data administrators or other experts are usually not involved. Also, the staff rotates relatively fast as organizing students graduate and leave the institution, making it harder to reuse last year's experience.

Although in general conferences are self-sustaining by making use of participation fees, the execution quality of the event depends on the creativity, enthusiasm, and personal experience of the students at the top of the managerial hierarchy, rather than a solid financial basis. This results in the lack of ability to build modern, automated organizational tools, which ultimately causes data management to be cumbersome and insecure — even though the major part of the organizational work is indeed data management and batch processing. A customizable software system offered as a rationally priced service, tailored to the administrative needs of MUN conferences could greatly reduce this organizational burden by providing easy-to-use data management and other administrative features.

²The concept of Model United Nations is detailed in Section 2.1.

Aside from the organizational concerns, MUN conferences provide outstanding networking opportunities to both the participants and the organizers. Participants working themselves towards a diplomatic career can substantially benefit from building global acquaintances among their future colleagues. Experienced career diplomats attending MUN conferences as guests can open doors for prospective diplomats which no education can. Professional networking among future and current diplomats can be supported and facilitated well by a suitable software system.

Inspecting current solutions, there is no software system on the Internet, which solves the administrative problems of MUN conferences, while making use of the great networking potential of the MUN framework. Implementing such a system would appreciably further global diplomacy in the long run.

The complexity of an envisioned software system capable of solving all problems and exploiting all opportunities of the MUN framework is immense. Besides social networking, such an application would need built-in features for event planning, project management, flexible data handling, batch processing, and dealing with a hierarchy of roles and responsibilities, as well as further MUN-specific features for administering the diplomatic role-playing aspects of a conference. Therefore the formulated feature requirements to be implemented in the scope of this thesis was minimal. As I focused on the engineering aspects, my goal was to set up a solid, secure, and production-grade architectural and infrastructural foundation for my envisioned application — and for a prospective company that will maintain my application. My other goal was to build a highly secure, production-ready application upon that foundation, providing an extensible but basic feature set for organizing MUN conferences, while facilitating networking among participants.

1.3 Objectives and contributions

In this thesis I present an elementary version of the *Diplomatiq* social network software system, suitable for organizing MUN conferences. On the one hand, I will refer as *Diplomatiq* to the software system itself, and on the other hand, to the prospective company conducting the maintenance, marketing and sales operations of the software system. Outside the context of this thesis, the social network is the first step of a long-term plan involving global consumption of public data, for producing real-time diplomatic prognoses and analyses.

My first objective was to design and implement Diplomatiq on a solid, production-grade foundation, with a minimal feature set, which can be extended with further social networking, administrative, and real-time data analytics capabilities as needed. Considering the sensitive personal information stored in the system, the prospective applications of Diplomatiq, and my deep interest in cryptography and computer security, my second objective was to build the system upon a modern, layered security architecture, which provides cryptographic assurances in terms of application and data security.

My contributions include the following:

- I designed, built, secured and paid a company-level production infrastructure for the development, testing, production operation and maintenance of Diplomatiq, including several kinds of supportive infrastructure.
- I designed an application security framework addressing authentication, authorization, and data protection, which provides cryptographic assurances regarding access control and the confidentiality and integrity of sensitive data.
- I designed, implemented and published an elementary version of the Diplomatiq social network software system as a client-server application, using graph database technologies and the aforementioned security framework.
- I developed several supportive libraries outside the Diplomatiq software along the way. I published the built artefacts of these libraries with detailed documentation, for free use in the open-source community.
- I published the source code of all my contributions as separate open-source projects, centralized under one project organization, called Diplomatiq.

1.4 Structure of this thesis

The thesis is structured as follows:

- *Chapter 2* summarizes the preliminary knowledge needed for a high-level understanding of this thesis. It details the concept of Model United Nations and my personal experience with MUN. It also defines the idea of a social network. Then it gives examples for domain-specific social networks, and presents existing software solutions for the MUN community.
- *Chapter 3* introduces the main technologies, which I built Diplomatiq on.
- *Chapter 4* describes my approach of building and securing a production-grade infrastructure supporting the development, testing, and public operation of Diplomatiq, the social network.
- *Chapter 5* gives an overview about the produced supportive libraries. It unfolds the reasons of their existence, as well as their features and implementation details.
- *Chapter 6* demonstrates the Diplomatiq social network application with its specification, client-server architecture, features and development methods, and implementation details.
- *Chapter 7* reveals the applied cryptographic and other security measures I built into Diplomatiq, in order to protect the system and user data from unauthorized access, from the API to the database level.
- *Chapter 8* concludes the thesis and presents possible future directions.

Chapter 2

Preliminaries and related work

This chapter summarizes the preliminary knowledge needed for a high-level understanding of this thesis. It details the concept of Model United Nations and my personal experience with MUN. It also defines the idea of a social network. Then it gives examples for domain-specific social networks, and presents existing software solutions for the MUN community.

2.1 The Model United Nations framework

2.1.1 Introduction

The Model United Nations (MUN) framework is an academic simulation of the everyday operation of the United Nations (UN). It is typically an extra-curricular activity materializing as annual, few-day-long conferences organized by students of high schools or universities. Participants welcomed from all over the world take on the roles of assigned nations' UN delegates, forming diplomatic delegations with their peers. There are hundreds of such conferences taking place every year [6]. While a medium-size conference has a few hundred participants and another few hundreds of organizers, the current largest Model UN conference — The Hague International Model United Nations (THIMUN) — attracts over 3,200 students from around 200 schools, from more than 100 different countries [7].

Delegates are placed in UN-like committees, where they discuss topics related to international issues and conflicts by the methods of moderated formal debate. The conduct of the debates and the conference is specified in the *Rules of Procedure*, a conference-specific, formal regulation derived from a similar document of the United Nations [8]. The result of the debate is a UN-like *resolution*: a formal document expressing the opinion or will of a committee. Resolutions are generally recommendations, but in some cases — like in case of a resolution adopted by the Security Council: the UN body with “primary responsibility for the maintenance of international peace and security” [9] — the adopted resolution is legally binding for all member states. Although MUN resolutions of course are never

legally binding, larger MUN conferences like THIMUN forward their adopted resolutions to the UN. These forwarded MUN resolutions are occasionally formulated into real-world UN resolutions after further debate and amendments.

The country and committee assignments are known in advance, which enables delegates to perform research and develop their positions before the conference. Students usually build their stances upon the actual standpoints of the countries they represent, but this is not a requirement. Since usually all positions of a given country's diplomatic delegation is assigned to students arriving from the same school, delegates representing the same country can construct complex nationwide strategies across different committees by cooperating with each other in advance.

The larger a conference is, the more possibilities it has regarding the simulation of the actual workings of the UN, or — as the community reinvents itself — even other intergovernmental bodies, like subsidiaries of the European Union. Even though the UN has only 193 member states [3], simultaneously simulating the whole operation of the six main organs¹, and the Secretariat of the UN requires much more participants.

2.1.2 History

The history of Model United Nations dates back to the early 20th century. The first similar event is believed to have been held in November 1921 by the Oxford International Assembly [12]. It was based on the operation of the League of Nations, the first worldwide intergovernmental organization founded by the Allied powers after the First World War to maintain world peace. Although the League of Nations was formally disbanded in 1946 and its powers were transferred to the United Nations established in 1945, the organization marks an important milestone of intergovernmental cooperation [13].

The first well-documented Model League of Nations conference was organized by the Harvard International Assembly in 1923. It featured the same basic characteristics that modern MUN conferences have: it was organized by an academic institution, had moderated formal debate about international conflicts in committees, and had resolutions adopted as the result of the work conducted during the conference [12].

The era of Model United Nations started in the 1950s with the establishment of the first high school MUN, Berkeley Model United Nations in 1952, and two other MUNs founded by Harvard University: Harvard Model United Nations in 1953 and Harvard National Model United Nations in 1954. The founding of The Hague International Model United Nations in 1968 led to the global expansion of high school MUN conferences [14]. THIMUN was the first MUN in Europe, and is today's largest MUN conference [7]. In 1991, the Harvard

¹The six main organs of the UN are: the General Assembly with several subsidiary boards, commissions, committees, councils, panels, working groups and others [10]; the Security Council; the Economic and Social Council; the Trusteeship Council; the International Court of Justice; and the Secretariat [11].

WorldMUN, a university level MUN rapidly accelerated the spread of university-level MUN. In 2007, the actuation of the BestDelegate.com portal significantly increased the online existence of MUN, providing research and preparatory resources for delegates attending MUN conferences. Founding of MyMUN, an MUN-specific registration and administration system, and MUNPlanet², an MUN-specific social and knowledge-sharing network furthered the presence of Model United Nations on the Internet [14].

2.1.3 MUN in numbers

I have not found any databases, publications, or studies, which would yield satisfactory statistics about Model United Nations. However, according to several portals, websites and Facebook-pages, we can make assumptions about the worldwide spread of MUN.

Conferences

At the time of writing this thesis, MyMUN lists 2,457 conferences from December 2012 until today [6]. Calculating with roughly 8 years, and with the broad simplification that there were an equal number of conferences organized every year, there are annually more than 300 conferences worldwide, at least based on solely the data of MyMUN. Considering that MyMUN covers only a small fraction of all MUNs in the world, the number of annual MUN conferences likely goes well into the order of thousands.

Participants

MyMUN claims having over 100,000 registered members and over 900,000 yearly visitors [17]. The Facebook page of MUNPlanet [16] has over 150,000 followers. According to a 2007 calculation [18], there are 180,000 MUN participants in the United States only. The BestDelegate.com portal is used by over 750,000 people worldwide [19]. Considering the numbers above and MUN's increasing popularity, I would assume that millions of unique students attend Model United Nations conferences every year.

2.1.4 Networking within Model United Nations

MUN conferences offer a number of networking opportunities. First of all, delegates attend committee sessions, where they debate international issues, cooperate in producing resolutions, and leverage simulated international relations along their represented countries' best interests. Therefore the main contact point among them is work: they get to know fellow delegates by observing their leadership, public speaking, and negotiation skills in a competitive field. Committee and lunch breaks enable them to further their acquaintances during the day either professionally or personally.

²At the time of writing this thesis, the website of MUNPlanet [15] is not reachable, and its Facebook page [16] having more than 150,000 followers received its last update in June 2019.

Professional diplomats attending conferences as guests can also take part in committee sessions as observers, or as actual delegates or chairpersons, but they are more likely to attend the official ceremonies or soirées³. This way, delegates can interact with professional diplomats without unnecessary formalities of real-world diplomacy. Career diplomats are renowned to have appreciable social skills [4], which in this setting helps further mellow the atmosphere, and leads to fruitful conversations between generations.

Besides professional programs like debate sessions and official ceremonies, conferences provide a number of other opportunities for delegates to get acquainted with each other. Events like organized sightseeings and afterwork parties allows building informal bonds alongside professional ones.

Aside from maintaining virtual friendships, members of the MUN community often harmonize their conference participations to meet with their foreign acquaintances. Since there is no suitable networking platform for this scenario, participants from different countries usually keep connected and communicate via general social networks, like Facebook. Due to the lack of Facebook's MUN-specific capabilities, a large part of the networking effect is lost, as delegates are not given automated suggestions on which conferences to attend based either on their circle of friends, or on their previous Model UN experience.

2.1.5 Administration of a Model United Nations conference

The general administration of a conference can be divided to two distinct parts. The *professional division* involves administrative tasks related to the Model United Nations framework itself: composing the discussed topics, assigning countries and committees to members of delegations, conducting the actual debates and formal ceremonies, and other features associated with diplomacy. The professional division encompasses everything inside the simulation, where the participants are in their diplomatic roles. The *organizational division* covers the real-world event outside the simulation: travel and hotel arrangements, meals, conference accessories, merchandise, entertainment, among others. According to my personal organizational experience to be detailed in Section 2.1.6, the two divisions are separate responsibilities requiring completely different experience, and thus are best to be kept as isolated as possible.

Similarly to the United Nations, the chief administrative officer of an MUN conference is usually the *Secretary-General*, responsible for organizing, administering and conducting the conference. In case of the aforementioned two-division administrative approach, the Secretary-General is responsible only for the professional division of the conference, and reports to the *Conference Manager* or *Project Manager*, who leads the organizational division, and is responsible for the entire conference. Henceforth I refer to the organizational and professional divisions together as management.

³elegant evening party, usually with snacks and drinks

In the following sections I detail the procedure of organizing a medium-size, high school-level MUN conference with the two-division administrative approach, broken down to distinct, preemptive phases. Most of the following is based on my personal experience, but it also contains parts I learned from event planners or other Model United Nations conference organizers.

Preliminary arrangements

Once the management of the previous year's session agreed upon their successors, the new management starts negotiations with the headmaster of the organizing school. They settle the dates for the conference, discuss necessary resources the school can provide, and start or continue securing external locations⁴ for larger conference ceremonies involving all participants, which the school usually cannot host.

After the initial negotiations, the management announces the conference's subsequent session in the host school with its date and vacancies in the organizational structure, and starts coordinating interviews with eventual applicants wishing to take part in organizing the conference.

Participants' application to the conference

After the professional division agreed on the UN bodies to be simulated, they publish preliminary committee assignments with high-level topics on the conference's website or Facebook page. As the management opens the application for participants, delegations and individual delegates apply to the conference specifying their preferences of country and committee assignments. Since high school students rarely travel abroad alone, a delegation is usually accompanied by a couple of teachers of the applicants' school. The accompanying teachers are recorded into the registration system as well as the students.

The application procedure comes with significant paperwork. The management consisting of junior members usually lack the experience and resources for efficiently and securely processing sensitive personal information⁵ of hundreds or thousands of applicants, and a school's IT division cannot be expected to be prepared for developing a suitable system either. Thus it is common that the registration is implemented with a simple online form populating insecure spreadsheets, or a primitive in-house application often making data management only more complex. Ready-to-use software like MyMUN are often not customizable enough to be useful for conferences generally requiring somewhat tailored solutions, which leads to the usage of multiple different applications. This can cause more pain than gain by requiring manual data maintenance in unintegrated systems.

⁴External locations are often needed to be secured years in advance.

⁵Registration data usually includes the applicant's name, birthdate, email address, phone number, and home address. It is also common to require the applicant's social security number or even the serial number of their passport for speeding up any immigration-related checks and administration.

Accepting or rejecting applications & finalizing country and committee assignments

Following the closure of the application procedure, the management decides about the acceptance or rejection of prospective participants. As applicants usually register as part of their school's delegation consisting of multiple delegates, it is common that the application of an entire delegation gets accepted or rejected together.

Whether to accept a delegation's application to the conference is decided by considering various factors: these can include the delegation's level of expertise — often with regards to the reputation of previously attended conferences —, the ability of fulfilling their preferences of represented countries provided at registration, and aspects of diversity, among others. In case of more mature conferences, committees are often classified by their members' expected level of MUN experience. A beginner delegate taking a seat in an expert-level committee is inexpedient, as it sets back the efficiency of the debate, or vice versa, it prevents the delegate from staying on their learning curve.

Accepting delegations happens in the interest of filling committee seats — predefined seats of countries represented in committees — in the best possible combination with regards to expertise, personal preference, diversity, and the prospective efficiency of the committee's work. Creating the final assignments of delegations and country-committee pairs can be challenging if done manually, considering the volume of input data, and the factors needed to be taken into account. Most conferences do the entire assignment process by hand, in spreadsheets, along a malleable set of priorities. This usually means playing with the synthesis of the committees until all delegations desired by the management can be offered a place to the conference close to their preferences.

This process can be automated, if the management composes a definitive set of priorities on how to assign already fixed committee seats — country-committee pairs — to the applied delegates as individuals. The problem of creating the assignments with the aforementioned conditions is a fundamental problem of combinatorial optimization: it is called *maximum weighted bipartite matching* or *assignment problem* [20]. Mathematically speaking, we are looking for the maximum-size matching in a weighted bipartite graph, where the sum of the edges' weight is maximal. One part of the graph is the set of applicants, and the other part is the set of committee seats. The edges between parts represent possible assignments, and their weights are “goodness” values based on assignment priorities composed by the management. Since this approach does not cover the possible requirement of all delegates from the same school needing to represent the same country, it is not always satisfactory.

If same-school students need to represent the same country, the process can still be automated with maximum weighted bipartite matching, but with an additional constraint: the management must restrict the application to fixed-size delegations. In this case, a participant school's fixed-size delegation applies to the conference as a whole, specifying their preferences from the set of same-size delegations of represented countries on the

conference. This way, the bipartite graph's two parts consist of school delegations applied and country delegations represented on the conference. An edge between the two parts means that the two kinds of delegation are of the same size, so they can be assigned to each other. The weight of an edge is the same "goodness" value based on assignment priorities composed by the management. This approach needs a second application step: after applied school delegations were assigned country delegations represented on the conference, the applicants of the delegations need to distribute the assigned committee seats among themselves.

Payment

The participation fees are collected after the registration, generally in multiple parts. International bank transfers executed by students and teachers usually renders the work of the conference's accounting department challenging, as it is often difficult to identify which delegation made which payment. Due to the general lack of integrated payment solutions in the MUN scene, conferences often face a heavy burden of financial paperwork.

A conference management system with integrated payment processing and accounting features could solve these organizational problems. It would also provide a cleaner experience to the participants: they could pay instantly and individually, without the troubles of needing to transfer the money in groups, as whole delegations.

Travel, hotel, meal and other arrangements for participants

With the closure of the acceptance and the assignments procedure, the final list of students and their accompanying teachers attending the conference becomes available, and the management can start working on participants' personalized experience. Though habits differ, most conferences offer various convenience services for additional fees, such as transport within the city, accommodation, meal arrangements, and tailored sightseeing or other entertainment programs. This causes lots of additional paperwork with regards to the delegation's arrival and departure dates, hotel or other kinds of accommodation preferences, and several kinds of meal allergies and eating habits⁶.

Similarly to the registration, most of the organizational paperwork in connection with convenience services is done manually. A capable software system possessing all information of a delegation's precise schedule, as well as their hotel, meal and entertainment preferences, could automate all this paperwork. This would allow the management to focus on the quality of the provided services instead of administration. Also, it would open additional financial possibilities. Following preliminary arrangements between the conference management and hotels or other service providers, the applicants could purchase

⁶Since participants arrive from all around the world, conferences generally offer multiple types of menus respecting various cultures and allergies.

their necessities by selecting them from a list of integrated services during the application procedure. This would endorse further financial cooperation between parties — the conference, the company providing conference management software, and the service providers —, while providing a simple, one-step payment flow for the participants.

Personalized conference accessories

MUN conferences formally require participant students, teachers, and all other conference personnel to possess various personalized items. Every person — participants, guests, organizers, staff — should wear an official, conference-issued badge during all events of the conference. The badge usually describes the person's name and title — either a diplomatic one within the simulation in case of participants or guests, or an organizational one in case of organizers or members of staff. Participant delegates, chairpersons, teachers, guests and everyone else attending committee sessions should have official, conference-issued placards in front of them, describing their represented country or position within the committee. Placards are used for voting in larger committees, as well as for identifying delegates from the distance.

Producing hundreds or thousands of personalized items without experience and proper tools can be time-consuming. Management teams generally possess neither the experience nor the tools, which leads to hours or days of manual work of creating personalized elements, one by one. Besides being error-prone, this process consumes valuable organizational resources. A capable software possessing all necessary information about participants and organizers could automate this manual work.

Conducting the conference

Following a half-year or longer period consisting of planning and preparation only, conducting the conference itself is usually not a real challenge. Administratively, the management needs to record whether incoming delegations signed in, and received their conference packages, but from the opening ceremony through the committee sessions and parties to the closing ceremony, the conference generally goes by itself, being already scheduled well.

Processing and storing resolutions and conference data

Few MUN conferences pay attention to the collection and storage of conference data. As management teams usually lack the resources and experience for keeping digital or paper-based documents, produced resolutions and other conference data is generally not preserved. A capable conference management system could offer solutions for this: with online resolution editing and automated video recording features, the management would not even need to collect conference data, because all of it would immediately be saved in the conference management software's permanent online database.

2.1.6 Personal experience: Budapest International Model United Nations

The previous section's points are mostly based on my personal organizational experience regarding Model United Nations. My former high school, József Eötvös Secondary Grammar School, located in the 5th district of Budapest, first organized *Budapest International Model United Nations (BIMUN)* in April 2011.⁷ BIMUN was among the first large-scale international MUN conferences in Hungary [21]. Previously I had attended several foreign MUN conferences, and I was looking forward to taking part in the organizational process of an international conference welcoming hundreds of students and diplomatic guests. Over the course of 6 years, I fulfilled several positions: I was a photographer, team lead, deputy Secretary-General, and part of the chief management as an executive advisor.

2.2 Social networks

2.2.1 Introduction

The term social network is used in social sciences, denoting a network of linked individuals or organizations, connected by social relations and interactions [22]. This thesis focuses on another aspect of social networks: online software providing networking, information sharing and messaging features for participating individuals or organizations. The two interpretations are related: users of a social network software form a social network in the scientific sense, and a social network can be loaded into a social network software for analysing patterns, or simply for facilitating interactions among social actors. Analysing social networks can be useful in various fields, including organizational studies and information sciences, and also in diplomacy [23].

2.2.2 Connection with graph theory

Since social networks essentially are connected entities, an obvious choice for modeling such networks is using graphs, where vertices are individuals or organizations, and edges are connections or interactions between them. This way networks are easier to visualize and understand [24], and graph algorithms are useful for various kinds of analyses [25]. By using different kinds of edges in the same graph, several kinds of connections can be represented within the same entity set, allowing to build a multi-dimensional model of the network. This can lead to deeper insight into network structure.

2.2.3 Social networking services

In this thesis I consider social networking services as social networks with associated features the network members can utilize, offered as mass-available services on the Internet.

⁷The school has organized BIMUN conferences every year since then. Unfortunately, the tenth anniversary session of BIMUN, which would have been held this year, got cancelled due to the COVID-19 pandemic.

Besides the core networking activity of building relationships within the network itself, associated features can include messaging, information sharing, and collaborational functionality, among others. Usually the goal of a social network is to provide the possibility of interaction among participants, beyond actual in-person interactions. This enables individuals to connect with others regardless of physical distance.

Social networks can be categorized into four main types [26], but with regards to the focus of this thesis, I divide the set of social networks into two parts.

- *Generic social networks* have no special characteristics or determined target audience: they offer a set of generic features for people to build and maintain virtual relationships with or without personal acquaintance.
- *Domain-specific social networks* have a determined target audience: they offer specialized features for members of the target audience in line with specific goals of usage in a given domain.

Real-world examples for generic social networks include Facebook [27], Twitter [28], and Instagram [29]. Facebook is the biggest social networking service in the world, having approximately 2.6 billion monthly active users [30]. It has several kinds of features for sharing ideas, photos, live or recorded videos, connecting with friends, and informing others in various ways. It also offers basic event-handling capabilities for facilitating in-person social interactions. Facebook's business model is based on targeted, personalized advertisement: it offers free features for users, who receive ads in their newsfeed and on other interfaces [31]. The platform has been continuously learning user preferences, and personalizes ads according to the user's assumed needs. Facebook's financial income is mostly based on the fees paid by businesses for publishing and targeting their advertisements in various applications of the social networking platform.

Twitter is a social network that gained popularity by providing microblogging features. Users can publish short "tweets": posts with at most 280 characters.⁸ It is asymmetric: users can follow other users instead of needing to get virtually acquainted, although for private profiles, users need to be explicitly granted access. Twitter has approximately 166 million daily active users [33], and builds its revenue upon advertising and data licensing [34].

Instagram is a social network primarily used for sharing photographs capturing important moments. Despite the fact that it is mainly used for image sharing, Instagram can still be regarded as a generic social network, because it is not to be interpreted within a specific domain, and has no determined target audience. Similarly to Twitter, it is asymmetric, but also has private profiles. Instagram was acquired by Facebook in 2012 [35], and shares the same basic characteristics regarding its business model: users receive sponsored posts and stories as personalized advertisements.

⁸Until 2017 November, the maximum length of a tweet was 140 characters [32].

Diplomatiq is a domain-specific social network. Its domain is diplomacy, its target audience is the set of junior and senior diplomats and other people working in diplomacy, and its specialized features — currently a very basic set of functionalities — include organizing MUN conferences. Broadening the capabilities of Diplomatiq will expectedly not alter, only broaden its domain-specificity.

2.3 Examples for domain-specific social networks

As previously described in Chapter 2, this thesis regards domain-specific social networks as networks having a determined target audience, offering specialized features for members of the target audience in line with specific goals of usage in a given domain. Real-world examples for domain-specific social networks include LinkedIn⁹ [36], DeviantArt¹⁰ [37], and also Diplomatiq¹¹.

LinkedIn is an employment-oriented social network, facilitating professional networking: a connection between profiles usually represent real-world professional relationships. Having nearly 690 million registered members [36], it is the largest network of its kind. LinkedIn is a domain-specific network: its domain is human resources and employment, and its main target audience is job seekers and employers or recruiters. Besides general social network features like content sharing and messaging, job seekers can create professional profiles similar to a curriculum vitae, find jobs by specific criteria, and apply to job opportunities on the platform. Employers can list job opportunities on LinkedIn, while recruiters can interact with job applicants in the form of formalized or personalized messages. One interesting feature of LinkedIn's social network is *endorsements*: people can positively endorse each other's skills, allowing recruiters to select applicants based on community-approved claims, besides their education, previous work experience, and personally listed skills. LinkedIn's revenue is primarily based on a subscription model for its premium recruiting tool, which allows recruiters to access additional data about job seekers otherwise not available on the platform [38]. Apart from its recruiting tool, LinkedIn generates revenue from personalized advertisement as well.

DeviantArt is a domain-specific social network for sharing several kinds of artwork. Its domain is art and its target audience is artists and art lovers. Its features support sharing artwork of traditional and digital drawing and painting, photography, literature, filmmaking, among others. DeviantArt has over 45 million monthly unique visitors [37]. It generates its revenue by advertising, offering subscription-based memberships, brand partnerships, and producing prints for artwork published on the platform [39].

⁹<https://www.linkedin.com>

¹⁰<https://www.deviantart.com>

¹¹<https://app.diplomatiq.org>

2.4 MyMUN

2.4.1 Introduction

MyMUN is a domain-specific social network, claiming to be “the ultimate MUN database, conference management tool, and social network” [17]. In the short run, it is a potential competitor of Diplomatiq.¹² Its domain is Model United Nations, its target audience is organizers and participants of MUN conferences, and it provides functionality both for organizing an MUN conference, and for preparing to one as a participant delegate.

MyMUN’s website does not contain detailed company data, but according to other sources, the company was founded in 2014 [40] and seems to be an established startup, based on its 17 employees and \$3 million of annual revenue [41]. From this point forward, every information disclosed in Section 2.4 originates from my personal experience of using MyMUN via its website [17], therefore I disregard references until the end of this section.

2.4.2 Organizational features

The platform offers a number of features for making the conference organization easier. However, having tried out the software in a variety of different scenarios, I experienced the application to frequently produce failures regarding even its basic functionality.¹³

Registering an MUN society and a conference

In MyMUN, the first step of organizing an MUN conference is to register an organization, called “MUN society”. The user registering the MUN society becomes the sole administrator of the organization, and the only administrator of all MUN conferences hosted by the organization. Even though further organizer members can be invited to administer a conference, no access control is offered by the platform: invited organizers have the same permissions as an administrator. Since MUN organizational staff usually rotates quickly, and transferring the ownership of an MUN society or conference is not possible on MyMUN, this approach is not suitable for conferences hosting sessions over multiple years.

An MUN society can host multiple *conferences*. For registering a conference in the system, the user needs to supply the conference’s name (e.g. Budapest International Model United Nations) and codename (e.g. BIMUN), number of expected delegates on a session, contact information, and the start and end dates of the conference. Sadly, the platform does not allow the conference to have *sessions* as child entities, which means that a new conference

¹²Diplomatiq will outgrow the world of Model United Nations in the long run, which means it cannot be collated with MyMUN, as the two services will operate over different domains.

¹³As the website does not list any means of contact for feedbacks about the software, and I did not find any possibility to contribute to the closed-source MyMUN project, I have contacted the Chief Technology Officer via email, with a list of experienced failures and detailed reproduction steps.

entity needs to be registered for sessions of the same conference in different years. This excludes the possibility of e.g. tracking participants' performance on the same conference over different sessions, because there is no entity in the data model to connect the different sessions stored as separate conference instances. After submitting the necessary information, the conference gets immediately listed in the Discover section, allowing delegates and other participants to apply.

Conference application procedure

MyMUN offers various settings for the application procedure. Additional application data can be queried from applicants in the form of short text, long text, numeric, multiple choice, checkbox or file inputs. The organizers can manage the application of individual delegates, delegations, committee chairpersons, observers, and faculty advisors. The platform does not offer creating and managing custom roles beyond the previously listed ones, but each role's application deadlines and additional questions can be adjusted separately. Even though applications can be accepted or rejected, the system does not offer an automated way to accept or reject applications with regards to the best combination of applicants and country-committee assignments based on priorities. In MyMUN, a participant's application is either accepted or rejected, and the question of the applicant's country-committee assignment is addressed in a later organizational phase.

Committee and country assignments

The platform uses the term *Country Matrix* for denoting a matrix having committees on the X axis, and countries (committee seats) on the Y axis¹⁴. In the Country Matrix, organizers can assign accepted applicants to seats, or accepted applicant delegations to country delegations, if their sizes allow. The *Assignment Wizard* provides an automated way to assign committee seats to most of the delegates and delegations based on their preferences provided at the application, but the wizard cannot be configured to take into account additional priorities or restrictions. And even if it could, it still operates on the set of already applied participants, disallowing acceptance with regards to the best combination.

Handling finances

The financials of conference participation fees can be managed with MyMUN in various ways. Payment settings enable to adjust fees for different payment classes of different kinds of participants separately — delegates, head delegates, committee chairpersons, faculty advisors, and delegations —, but it is not possible to add custom payment classes. Participants can pay individually or in groups, with credit/debit cards, PayPal, or bank transfer, directly through MyMUN. The platform allows to issue refunds as well. Also, promotions and coupons can be offered to participants.

¹⁴The term *country matrix* is not specific to MyMUN, it is a common term in the MUN scene.

The organizational dashboard details a financial summary with all received and refunded payments broken down to payment classes. The cash flow view shows the financial reserves and the predicted income of the conference. Payment history can be exported with financial identifiers, so accountants of the conference can easily process payments.

Automated notifications and reminders

Relevant events of the application process are confirmed via automated email notifications. The confirmed events are the submission of an application, the acceptance or rejection of an application, and the assignment of a committee seat to an accepted applicant. MyMUN offers basic email notification templates for these events, which can be completed with additional HTML or text content in the settings.

Statistics

Organizers can create custom views of conference data stored in MyMUN. Custom views can include several metrics based on application type, application status, and other characteristics. Displaying such customized data can help organizers to decide if the conference met predefined numeric goals, e.g. diversity, distribution of genders in committees, etc.

Advertisement and marketing

Although most conferences usually have their already established audience¹⁵, some need more than organic reach only. MyMUN provides a variety of promotional packages for boosting the visibility of an event. Packages include advertising on the platform itself by highlighting the conference on one of the promotional pages, on MyMUN's social media accounts, and through email campaigns. Packages can be optimized, and one can also compose a fully customized marketing campaign of the individual promotional items.

2.4.3 Features for participants

Listing MUN conferences

Students can browse among conferences in the *Discover* section of MyMUN. This section consists of several views: the main view is a page highlighting upcoming and featured conferences. Events are categorized by continent, and featured conferences are displayed at the top of the page, and across categories. Besides the highlights view, other views display the conferences on a map, in a calendar, or in a list view providing filtering capabilities.

¹⁵With proper networking skills and some personal popularity, there is usually no need for significant marketing efforts. It is usual that organizers organically pre-promote their conferences in the MUN community, either in-person on other events, or on social media. On the very first session of BIMUN in 2011, the number of applicants was five times the conference capacity, as organizers put serious efforts into organic marketing among their friends and acquaintances.

All views allow visitors to apply to displayed conferences. The most complete display of events stored in the system is in the regular list view, where also the most information is displayed of individual conferences at once. Conferences can be ordered by name, date, location, number of delegates, fee, and rating.

Applying to a conference

The application procedure starts with checking and modifying one's personal data as required. This step ensures that the application is performed with up-to-date information, in case any personal identifiers or characteristics — e.g. diet or allergies — stored in the system became obsolete. Data saved during this step is also updated globally.

In the second step, the applicant chooses their role on the conference. This can be delegate, member of a delegation, chairperson, faculty advisor or observer. Some of the roles cannot be selected if preliminary conditions are not met, e.g. users with the occupation student cannot apply as a faculty advisor. If the applicant chooses to be a member of a delegation, then they need to supply its name, and then the delegation's head delegate needs to confirm the membership in the system.

The third step lists the terms and conditions of a conference. Having contractual arrangement between conferences and participants is not ordinary, therefore this step is included only if the conference formulated and submitted such documents to MyMUN. If there are any, accepting the conditions is a requirement of the application.

The applicant needs to supply their committee seat preferences in the fourth step, first choosing the committee from the set of the committees offered by the conference, then the represented country within the committee with a search-assisted drop-down menu¹⁶. The minimum and maximum number of necessary assignment preferences is conference-specific. As described before, the software's *Assignment Wizard* feature claims to assign committee seats to applicants closest to their preferences overall.

Additional conference-specific questions need to be answered in the fifth step. Most conferences ask about applicants' personality, MUN society membership and previous MUN experience. Usually the goal of additional questions is to evaluate an applicant's attitude and commitment. Larger conferences implement sophisticated questionnaire often requiring serious professional background to fully answer.

The sixth step requires the applicant to formulate a motivational letter in which they introduce themselves and describe their personal and professional interest in the conference. The answer's input field supports long text formatting with optional images, links and other kinds of attachments. A separate document acting as the motivational letter itself cannot

¹⁶A search-assisted drop-down menu allows users to choose a value from a set, while providing search capabilities in the set.

be uploaded, meaning that applicants need to prepare this document online, although it is automatically saved during editing.

In the seventh and final step applicants can review their application before submission. All steps' information can be separately edited, navigating the process back to the step to be amended. The application procedure can be either finished by submitting the application or aborted by withdrawing the application and deleting all saved data.

Offers during the application process

MyMUN offers several kinds of services already during the application process. In a continuously visible separate sidebar, hotel, flight and health insurance offers are shown. Hotels, motels and other accommodation opportunities are displayed on an interactive map covering the vicinity of the conference, indicating the event's exact location. Selecting an accommodation option reveals its price and community rating, and the offer can be reserved instantly, after the user was navigated to the accommodation service provider's website. Besides generic offers, conference-specific contractual accommodation offers can be displayed in a highlighted manner. Flights and other means of travel are also directly offered to applicants by integrating external service providers into MyMUN. Besides travel, the platform offers conference-specific health insurances as well.

Research and study guides

Model United Nations is an academic activity, and thus it involves conducting research before attending a conference. MyMUN offers a virtual library of study guides and position papers¹⁷, which will be extended with a section for drafted resolutions as well. The platform's position paper database counts over 14,000 entries. MyMUN claims that it will also serve online courses on Model United Nations in general, and specifically on preparing to conferences as a delegate.

Travel services

MyMUN promotes and cross-sells several kinds of travel services, some in cooperation with conferences. Services include flights, accommodation, health insurance, car rental, and adventure tours. A so-called *group service* offers a package of integrated services tailored to delegations: after receiving all data from the head delegate or a faculty advisor, MyMUN acts as a travel agency, and makes reservations for flights and accommodation, books required travel insurances, and sends a detailed travel offer to the delegation.

¹⁷An MUN position paper summarizes the position of the delegate's country regarding a given topic or issue. Its purpose is to prove the delegate's or delegation's level of preparation before the conference, and to serve as a topical fact sheet during the event.

2.4.4 Business model

The financial basis of MyMUN lies on paid conference organization features, conference advertisement and marketing, the cross-sales of travel services, and on a three-level subscription-based membership in the *Delegates Club*, a closed circle of MUN participants offering exclusive content, savings and insurance benefits.

Paid conference organization features

MyMUN offers two plans for conference organizers. In the free *Trial* plan, basic features are available, like listing a conference on the platform, adding committees and organizers, and the management of chairpersons' application. The Trial plan is not sufficient for assisting in the organization of an MUN conference from the beginning to the end, encouraging organizers to subscribe to the *Professional* plan, which allows to utilize all features of the platform. The Professional plan costs 4% of the value of each financial transaction actuated through MyMUN's payment system, but minimally €4 per transaction. It is invoiced either directly to delegates, integrated to the online payment procedure of a conference application, or to the conference, if online payments are not available in a given area of operation.

Conference advertisement and marketing

As described previously, MyMUN allows organizers to promote their conferences in various ways. The offered packages containing several kinds of promotions are priced between €300 and €3,000. The fee of individual marketing elements — like featuring a conference on chosen landing pages or in email campaigns — vary between €100 and €1,500.

Cross-sales of travel services

Although directly not mentioned, fees of offered travel services probably contain dividends received by MyMUN. It is a common sales scheme, when a seller offers a service, and a second party offers a large audience in assumed need of the offered service, in exchange for dividend. MyMUN's offerings — flight tickets, accomodation, health insurance and group services — all provide ways for selling the services with considerable financial benefits.

Delegate's Club

The Delegate's Club is a subscription-based membership group within MyMUN, offering three levels of additional services for conference participants. Membership is not a requirement for using the platform for conference applications, but it comes with benefits. The smallest *Bronze* package costs €27 monthly for 6 months: it offers a flatrate for health insurances purchased through MyMUN, and full access to the position paper database.

The *Silver* package costs €65 monthly for 1 year, includes all benefits of the Bronze package, and comes with a free ISIC Card¹⁸ offering discounts for hotels, hostels, restaurants and museums. In addition to all benefits of the Silver package, the *Gold* package offers discounted subscription to diplomatic periodicals, and costs €90 per month for a fixed 1-year term.

2.5 MUNPlanet

MUNPlanet was the largest MUN community in the world in the form of an online knowledge network where MUNers create, curate and share their knowledge and experiences about issues of global importance [16]. It was a potential competitor of Diplomatiq, being a global, domain-specific social network expanding along Model United Nations. Its domain was Model United Nations, its target audience was all participants of MUN conferences, and its specific features were to facilitate knowledge sharing within the domain.

No further information is available of MUNPlanet, since at the time of writing this thesis, its website [15] is not reachable, and its Facebook page [16] having more than 150,000 followers received its last update in June 2019.

¹⁸The ISIC Card is a student ID card, the only one of its kind which is internationally accepted.

Chapter 3

Chosen technologies

This chapter mentions the main technologies, which I built Diplomatiq on.

3.1 Spring Boot

I like Java¹ because of its view of the software engineering world: its meticulous design and its the well-planned typing and interface system shows that Java is a technology which deliberately wants to be mature. Also, it is the world's most popular object-oriented programming language in 2020, according to StackOverflow [42]. Along these aspects, I was sure that I want to build Diplomatiq on a Java-based technology.

I also wanted to build Diplomatiq's back end application on a solid, production-grade solution. Although I used Spring Boot² for previous hobby projects, I wanted to collect more experience with using the framework for developing an API application, for professional purposes. Spring Boot offers that it makes it easy to create stand-alone, production-grade Spring based applications that one can "just run" [43].

Spring Boot offers convincing features for developers who want to build software instead of configuring and plumbing. One of Spring Boot's main characteristics is its opinionated view on starter dependencies, making initial build configuration unnecessary: one can just bootstrap a project, then start developing it without needing to select dependencies, generate code, or configure the project at all. Another important aspect is the capability creating stand-alone, self-contained Spring application with an embedded servlet container, without needing to maintain a Java application server. These two aspects, and the fact that Spring Boot is a mature, production-grade framework, convinced me easily to decide that I implement Diplomatiq's back end application with the Spring Boot framework.

¹<https://www.java.com>

²<https://spring.io>

3.2 Angular

I wanted to implement Diplomatiq's client application as a single-page web application, offering progressive, native-like experience to users, which is ubiquitously available with the help of only a web browser. Since the envisioned application satisfying all requirements of the Model United Nations framework would be huge in terms of code and components, a scalable, mature solution is needed, which makes use of cutting-edge web technologies, such as lazy loading application modules. Angular³ is a scalable, well-maintained, complete framework, and also my primary choice for larger web application projects.

I have been also developing smaller and larger Angular applications in my professional capacity for the last three years. I experienced that Angular offers a mature, complete, but customizable feature set for web applications, which aim to grow over the course of time. It also features a capable development toolset with integrations to several development environments. Since Angular is built on TypeScript⁴, thus offers a competent type system, an Angular application is able to scale rapidly, while it remains maintainable.

3.3 Neo4j

Graphs are mathematically defined data structures being broadly used in several fields of computer science. Recent technologies and implementations made possible for developers to easily embed graph data models into their applications. There are numerous real-world scenarios which can be represented more efficiently as graphs (*nodes* connected to each other by *edges*), than with the traditional, relational approach.

Graph databases are NoSQL databases, which store data in graphs instead of the traditional, table-based approach.⁵ In graph databases, a relationship represented as an edge in the graph is a “first-class” entity, and has the same basic storage characteristics as a node. Relationships are directly linked to entities, and therefore entities are directly linked to each other via relationships. This allows the querying of related entities to be fast, since the process does not involve lookups.

In data models where connections between data represent just as much business value as the actual data itself, it can be sensible to use graph database technologies over SQL databases. Whereas in relational databases, connections between entities rely on foreign keys and pivot tables, in graph databases they come naturally, without additional modeling needs. A (domain-specific) social network is exactly such case, where the connections value the same as data, so using a graph database was a natural decision to make.

³<https://angular.io>

⁴<https://www.typescriptlang.org>

⁵The underlying data storage methods vary. There are graph databases which store graph data in relational tables, introducing another layer of abstraction between the stored physical data and the database.

3.3.1 The property graph data model

It is common to define graphs as a set of objects, in which some object pairs are connected to each other. In this model, an object is called *vertex* or *node* or *point*, and a connection between two *vertices* is called *edge* or *relation*. Connections can be detailed further by specifying their directionality, also they can be *labeled* to define them even more. Similarly labeling vertices leads to the model of *typed graphs*. If we assign properties to the nodes or relations, we get the model of *property graphs*. Properties, as shown in Figure 3.1, are usually key-value pairs in the format of `key = 'value'`. Generally, keys are strings, and values represent common data types like string, integer, float, etc.

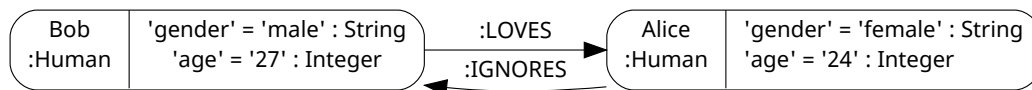


Figure 3.1 Two persons' relationship modeled with a property graph

3.3.2 The Neo4j graph database

The field of graph databases has been gradually expanding in the recent years. Many vendors offer graph database solutions either based on a pure graph model or on multi-model approaches [44, 45, 46]. Databases vary by query languages as well, one of the most popular being GraphQL, an open-source data and manipulation language for APIs [47].

Among a handful of graph database vendors [48], Neo Technology's Neo4j is the most popular one [49]. It features full ACID-compliance upon a pure graph data model, contrary to other vendors' multi-model approaches. Besides Neo Technology, Neo4j is backed by the open-source community as well [50]. There are two variants: *Community Edition* and *Enterprise Edition* with an extended feature set [51]. Neo4j also offers a program tailored to startup companies [52]. With the help of this startup program, companies can build their applications on the Enterprise Edition of the Neo4j platform, free of charge.⁶

During my Bachelor's Thesis, I had the opportunity to experience the depths of Neo4j. I refactored and extended a static code analysis software, which builds Abstract Syntax Trees, and stores them in Neo4j for running queries over the graph structure. In this project, Neo4j proved to be a stable, mature database, even usable for analysing repositories having more than 30,000 lines of code, which got mapped to an Abstract Syntax Tree with millions of nodes and even more millions of relationships. Every since then, I am confident that Diplomatiq will use Neo4j as its database platform.

⁶Certain limitations apply: the company must have at most 50 employees and at most \$3 million annual revenue, among others [52].

3.3.3 Cypher

Cypher is a query language developed especially for graph databases by Neo Technology [53]. Figure 3.2 shows that the language uses a sort of ASCII-art to represent nodes and relationships: nodes are in parentheses, relationships are in brackets surrounded by relationship direction information.

```
(Bob)-[:LOVES]->(Alice)
```

Figure 3.2 A basic Cypher example

Cypher syntax is elegant and expressive, thus very readable. Besides using it to represent nodes and relationships, we can utilize it to access the Neo4j's indexing capabilities and stored procedures as well. Even complex pattern-matching conditions can be expressed easily and intuitively in Cypher. Although today's modern application development frameworks — adopting increasingly capable data mapping solutions — make it less and less necessary to directly interact with databases, complex queries can still involve composing database commands manually. Therefore the expressiveness and ease of use of a database query language still remains essential.

Basic querying capabilities include matching a given pattern. Patterns can include nodes, and directed relationships, and Cypher can also match by nodes' or relationships' properties. Figure 3.3 shows a basic example on how to find every such relationship, in which a male human loves another female human.

```
MATCH (b:Human { gender: "male"})-[:LOVES]->(a:Human { gender: "female"})
```

Figure 3.3 Basic pattern matching capabilities

Inserting nodes and relationships can also be achieved in an easy and semantic way. Figure 3.4 shows a basic example on creating two nodes and a relationship between them.

```
CREATE (b:Human { gender: "M"})-[r:LOVES]->(a:Human { gender: "F"})
```

Figure 3.4 Creating two nodes and a relationship between them

For avoiding inserting duplicate nodes, creation can be replaced with merging. In this case, if a node with the specified identifiers exists, it gets updated with any additional data specified by the query. Merging is similar to the create operation, as shown on Figure 3.5.

```
MERGE (b:Human { gender: "M"})-[r:LOVES]->(a:Human { gender: "F"})
```

Figure 3.5 Merging to avoid creating duplicate data

3.4 Git and GitHub

I have been maintaining altogether 9 projects related to Diplomatiq, all being tracked by a version control system, Git. I chose Git because of its maturity⁷ and popularity, and also because of the fact that this is the versioning tool that I am most experienced with. For open-source software maintenance, I chose GitHub, as it is the most popular software collaboration platform⁸, and offers advanced development and project management tools, security settings, and a full-featured, integrated testing infrastructure. Also, it enables to maintain repositories under a larger unit, called organization, offering sophisticated administrative and security features. GitHub is free of charge for open-source projects [56].

3.5 Microsoft Azure

Having relevant work experience in operating a production cloud server infrastructure, I had a concrete concept on Diplomatiq's demands on the short and on the run as well. The key aspects of choosing the platform were the following:

- It should provide strong cloud-based services with hybrid cloud⁹ capabilities for later expansion into real-world diplomacy.¹⁰
- It should provide strong PaaS¹¹ capabilities, and also IaaS¹² solutions.
- It should provide options on data residency to comply with diplomatic requirements.
- It should offer a flexible pricing model, so I can start with a smaller, cheaper infrastructure, and scale it later as Diplomatiq's production workload grows.
- It should provide mature, integrated security solutions.
- It should provide affordable developer support.

Considering the above, I evaluated service offerings of Microsoft Azure, Google Cloud Platform, and Amazon Web Services. Google seems to lag behind on hybrid solutions, and also seems to be the worst from the financial aspect for Diplomatiq's use cases. Even though Amazon is the oldest cloud service, it seems to lack integrated data residency solutions. Microsoft seems to offer everything Diplomatiq will need in the long run, and it proved to be the best in terms of pricing, therefore I chose Microsoft Azure as Diplomatiq's cloud server infrastructure platform.

⁷Git has been developed since 2005 [54].

⁸GitHub has over 40 million users [55].

⁹A hybrid cloud consists of cloud-based and on-premise infrastructure elements.

¹⁰According to my knowledge, governmental and diplomatic software often require on-premise solutions.

¹¹Platform-as-a-Service

¹²Infrastructure-as-a-Service

Chapter 4

Building and securing a company-level, production-grade infrastructure

This chapter describes my approach of building and securing a production-grade infrastructure supporting the development, testing, and public operation of Diplomatiq.

4.1 Introduction

The underlying infrastructure plays a foundational role in the eventual success or failure of every business. 21st-century companies often build their business operations entirely on information technology solutions, meaning a well-founded IT infrastructure is key to succeed. Even though at the time of writing this thesis, Diplomatiq is a company existing only in the future, the infrastructure I elaborate in the present will be the foundation of its business operation. By setting up a robust and secure infrastructure, I want to establish the future of Diplomatiq. It needs to be done right, so it does not need to be done again.

This involves two principles. The first is that key infrastructure elements need to be established using mature and robust solutions, so they do not need to be rebuilt or replaced later. This excludes trial versions of services, expiring student offers, and generally free solutions as well. Organizational hierarchy needs to be set up properly, allowing later expansion, and the infrastructure with all its access credentials should be documented meticulously. The second principle is that security should be taken into consideration from the very beginning. Authentication and authorization policies should be the as strict as possible, and all access credentials should be stored in a safely encrypted manner.

Throughout this chapter, I introduce the various infrastructure elements I evaluated, purchased and integrated into Diplomatiq's infrastructure. Security-related aspects will appear in most of the sections.

4.2 Naming

A good brand name identifies a company in various ways. Apart from marketing purposes, the name should be sufficiently unique to be usable within the various services and namespaces on the Internet. For this purpose, *Diplomatiq* seemed to be suitable: it is unique, appropriately short for both domain names [57] and the human memory [58] with its 10 characters, and it characterizes its subject well.

I formulated the following — prioritized — guidelines for reserving namespaces for Diplomatiq across services on the Internet:

1. If available, use *Diplomatiq* (capitalized).
2. Else if the service allows lowercase letters only, use *diplomatiq*.
3. Else if use *DiplomatiqOrg* (capitalized).
4. Else if use *diplomatiqorg*.
5. Else use a custom name.

As of now, there has been no need to apply the 5th rule.

4.3 Brand

For visual recognition, a company needs a well-defined image. Diplomatiq's corporate identity was designed by one of my acquaintances, Roland Hidvégi. It includes a logotype, three variants of application icons, ten brand colors, and Eina [59] as the advised font family¹. Figure 4.1 and Figure 4.2 shows the logotype and the application icon variants.

Figure 4.1 The logotype of Diplomatiq



Figure 4.2 The application icon variants of Diplomatiq

¹As Eina is not available as a free web font [60], I temporarily use Helvetica instead.

4.4 Domain name

4.4.1 Overview and purchasing the domain name

A domain name in the Domain Name System (DNS) represents a network domain, or it translates to an Internet Protocol address [61]. Having a domain name is necessary for companies with web-facing services, both for users to easily memorize the address of the service, and for deploying security measures, such as Transport Layer Security (TLS) [62]. As Diplomatiq is not primarily a commercial entity, but rather a diplomatic organization, I decided its domain name to be `diplomatiq.org`. Since I was already registered at the domain name registrar Namecheap², it was straightforward to purchase the domain name from them, under my existing user account. I set up various DNS records for deployed services, these will be detailed later at describing the services themselves.

4.4.2 Security

Even though I have a secure, long, cryptographically random password for my Namecheap account — as well as for every other user account I have — and I store it in an encrypted password manager, I enabled multi-factor authentication, to reduce the possibility of an unauthorized party accessing Diplomatiq's DNS infrastructure. I also turned on all security alerts to get immediately notified about all events regarding the domain. I deployed DNS Security Extensions (DNSSEC) under the Diplomatiq domain. DNSSEC essentially prevents spoofing DNS data by providing a set of methods to DNS clients to cryptographically authenticate the integrity and existence (or non-existence) of DNS records [63]. Although there is a long-standing debate about the usefulness and operational security of DNSSEC [64, 65, 66], I decided that until it does not cause any failures or outages in production, it will be enabled.

4.5 TLS infrastructure

4.5.1 Overview

Transport Layer Security (TLS) is a cryptographic protocol with the goal of providing a secure channel between two communicating parties — usually between a client and a server — over the Internet, offering cryptographic assurances for the following:

- *Authentication* with public-key (asymmetric) cryptography. In TLS, the server is always authenticated, and the client is optionally authenticated.
- *Confidentiality* with secret-key (symmetric) cryptography.
- *Integrity* with cryptographic message-authentication codes [62].

²<https://www.namecheap.com>

For protecting web application users and web API³ consumers on the Internet, a web server should serve its contents over TLS, e.g. over the HTTPS protocol, which is essentially HTTP over TLS. More and more web browser APIs require websites and web applications to be served over HTTPS [67].

Authenticating the server involves a digital certificate, which proves the ownership of the server's *public key*. The public key and its cryptographic key pair, the *private key* are involved in the cryptographic key handshake of TLS, resulting in a symmetric key for encrypting data in transit. Certificates are issued to one or more specific domain names, cryptographically signed by a trusted third party called a Certificate Authority (CA).⁴ Acquiring such a certificate requires proving the ownership of the domain names in question. *Non-wildcard certificates* only certify domain names they were issued to, whereas *wildcard certificates* certify given domains and all their immediate subdomains.

4.5.2 Purchasing a TLS certificate

Having a TLS certificate signed by a trusted CA is a requirement of serving content over HTTPS, thus it is necessary — but not sufficient on its own — for securing web applications and web APIs. Such certificates can be purchased from a multitude of vendors. For `diplomatiq.org`, I purchased a TLS certificate from Sectigo⁵, one of the leading TLS certificate vendors in the world [68]. The certificate is issued to the domain names `diplomatiq.org` and `www.diplomatiq.org`, and since it is a non-wildcard certificate, I will need to acquire additional certificates for other subdomains.

4.5.3 Security

The private key of a certificate is a highly sensitive secret. An attacker obtaining the private key of a server certificate is able to decrypt all traffic sent to and received by the server, or it can even modify the data in transit, tricking the user into surrendering sensitive information, such as passwords. While being stored securely, the private key needs to be always available to the server, as it is constantly involved in the communication. The private key of the certificate issued to `diplomatiq.org` is encrypted with a long, cryptographically random password, and it is stored in a cryptographic Hardware Security Module⁶ by Microsoft Azure's Key Vault⁷ service. This way, the private key is only available in an audited, secure manner, guarded by strict access policies.

³Application Programming Interfaces define interactions between softwares. In this context, a web API means a web-facing service, which serves data for client applications.

⁴For the sake of compactness, I will not go into the endless details of the X.509 certificate infrastructure and the underlying public-key cryptography in this thesis.

⁵<https://sectigo.com>

⁶Hardware Security Modules are separate physical computers designed to keep cryptographic keys safe. They offer tamper resistance making it extremely difficult to extract and steal secret keys [69].

⁷Microsoft Azure and its Key Vault service in particular will be detailed later.

As an additional safety measure, I deployed *Certification Authority Authorization (CAA)* DNS records for `diplomatiq.org`. CAA records specify authorized CAs, which are allowed to issue certificates for the given domains [70]. The records essentially form a CA whitelist, preventing the mis-issue of valid TLS certificates by unauthorized parties. I set up CAA records as granularly as possible: every subdomain has its own authorization. I specified Sectigo for the apex domain `diplomatiq.org` and the subdomain `www.diplomatiq.org`, DigiCert for the subdomains `app.diplomatiq.org` and `api.diplomatiq.org`, and Let's Encrypt for the subdomain `neo4j.diplomatiq.org`. DNSSEC prevents spoofing additional CAA records into the Diplomatiq domain. Diplomatiq currently only makes use of server certificates for supporting TLS on its website, web application and web API. Possible future uses of certificates include signing released software with a code signing certificate, or authenticating API clients with client certificates.

4.6 Email infrastructure

4.6.1 Overview and usages

The ability of sending and receiving emails is a basic business requirement for communicating with business clients and service providers. In the following, I will refer these as *individual emails*, as they are mostly initiated by a human individual. Also, application development often demands sending automated *transactional emails* for customers, as well as *promotional emails* delivering marketing campaigns and promotions.

4.6.2 Sending and receiving individual emails

Namecheap offers basic email forwarding capabilities along with its DNS service I subscribed to. Even though certain business email providers offer more robust — and also more expensive — solutions, I decided that my current requirements are covered by forwarding emails received by any address ending with `@diplomatiq.org` to my personal email address. I also managed to configure my personal email provider to send emails in the name of several `@diplomatiq.org` email addresses through SendGrid⁸, a service for delivering transactional and promotional emails. For achieving this, I needed to set up separate email entities in my personal email provider to send emails through SendGrid's SMTP-over-TLS API, authenticated by a newly created API key. Currently several email addresses are configured with the above method, each used for different purposes in different services. Dedicated *per-service email addresses* are useful for services lacking federated authentication and role-based access control features, as they enable to distribute responsibilities among colleagues by providing access to the email addresses, without irrevocably binding those responsibilities to the colleagues' email addresses. The configured email addresses with their usage are available in the Appendix.

⁸<https://sendgrid.com>

4.6.3 Transactional and marketing emails

For delivering transactional and marketing emails, I subscribed to SendGrid's smallest paid plan, which includes 40,000 sent emails per month. The service allows to create rich-text email templates in its online editor, then send personalized emails to multiple addresses, by substituting template placeholders with per-user customized data. I configured SendGrid to use the `team@diplomatiq.org` email address for all outgoing email communication. Configuring the service to access and use the `diplomatiq.org` domain was straightforward: it only involved adding generated CNAME records to the DNS configuration.

4.6.4 Securing SendGrid access

Since SendGrid offers neither federated nor email-based user management, I created a new account with a username. I enabled two-factor authentication for the service, and I defined various alerts for account access and quota usage. I also introduced IP address-based access controls: my user account is accessible only from my static home IP address and my private VPN^{9,10}, and the SendGrid API authenticated by my API keys is only accessible from the IP range of the production server infrastructure, which I will detail later. The issued API keys have minimal privileges allowing email sending only.

4.6.5 Securing emails sent by Diplomatiq

Sending emails over the Internet is inherently insecure, as the design of the core email protocols do not incorporate any security features for sender authentication and authorization [71]. The infrastructure on its own allows anyone to send emails from any domain, without verifying the authenticity of the sender [72]. There are several additional security measures to mitigate this threat, such as the Sender Policy Framework [73], DomainKeys Identified Mail Signatures [74], and the Domain-based Message Authentication, Reporting, and Conformance [75] protocols. I have applied all three of them for Diplomatiq.

The Sender Policy Framework (SPF) was designed to detect if the sender address of an email was forged by a party outside the sender's domain. The framework's operation is based on a DNS TXT record¹¹ indicating the host or IP address of email servers authorized to send emails originating from the domain. Besides authorized servers, the record also includes instructions for recipient servers and clients on what to do with detected forgeries: such emails can either be forwarded to the recipient's mailbox tagged as spam, or rejected and not delivered. Diplomatiq's SPF policy is configured by Namecheap and SendGrid based on my settings, and it commands to reject emails detected as forgeries.

⁹Virtual Private Network

¹⁰I have a personal VPN hosted by a self-configured virtual machine in a data center. The VPN's IP address is also allowed to access SendGrid, so I can log in to my account even if I am not home.

¹¹The SPF record has a specific format, which is defined in RFC 7208 [73].

The DomainKeys Identified Mail (DKIM) Signatures scheme offers similar email sender forgery detection capabilities as the Sender Policy Framework, but also it provides additional assurances supported by public-key cryptography. For utilizing DKIM, the sender server needs to cryptographically sign outgoing emails with a private key, and the domain's administrators need to publish the server's corresponding public key in a DNS TXT record.¹² Recipient email servers and clients can check the authenticity of an email by looking up the sender domain's public DKIM key then verifying the DKIM signature attached to the email¹³ with the public key. On the one hand, a valid DKIM signature cryptographically guarantees that the email was sent from an authorized party, and on the other hand, it also verifies that the email was not modified in transit. I configured the DKIM records of Diplomatiq to be automatically managed by Namecheap and SendGrid.

The Domain-based Message Authentication, Reporting and Conformance (DMARC) protocol extends SPF and DKIM by allowing domain administrators to explicitly indicate that emails are to be authenticated by SPF or DKIM or both, and to instruct email recipient clients to behave in a specific way when any or all authentications fail, such as rejecting the message, or putting it in quarantine. DMARC also offers a reporting mechanism, which sends reports about authentication failures to a specified email address. Reports can be daily aggregates or real-time “forensic” reports including detailed data about each failures, or both. Diplomatiq's DMARC policy is set to the strictest: emails not passing all checks should not be delivered to the recipient's inbox. For receiving and analyzing aggregate reports, I use an external tool, called Dmarcian¹⁴. DNSSEC prevents the unauthorized modification of Diplomatiq's SPF, DKIM and DMARC policies.

4.7 Source code management

4.7.1 Registering the Diplomatiq GitHub organization

I registered an organization on GitHub under the name *Diplomatiq*. I uploaded all necessary data including logos, descriptions, general and billing email addresses, and website addresses. Considering future employees, I made it mandatory for all members of the organization to use two-factor authentication. I verified my ownership of the `diplomatiq.org` domain to GitHub with a custom DNS TXT record, thus GitHub displays a “Verified” label next to Diplomatiq's website address. For making future collaborative development easier, I added a set of organization-wide issue and pull request labels, making sure all repositories I create have the same issue types.

¹²The DKIM record contains additional information besides the key itself. The format of the record is specified by RFC 6376 [74].

¹³DKIM signatures are usually verified by email clients rather than end-users, thus the signatures are generally not visible as part of the email.

¹⁴<https://dmarcian.com>

4.7.2 Standardized, organization-wide documents and configurations

As I created more projects, I experienced that certain documents, templates and configurations need to be present in all of projects, mainly because of the recommended open-source community standards maintained by GitHub [76]. For being able to instantly create a new project with Diplomatq's standardized project structure containing all such necessary files, I created another project called *project-config*, which will be described in Chapter 5.

License

The most basic project requirement is a license. Within Diplomatq, project licenses are stored in a file called `LICENSE` within the project's root directory. This is in line with GitHub's recommendations, thus the platform can automatically parse and display license information. Currently all Diplomatq projects are licensed under the MIT License [77].

Readme

All projects are initialized with a `README.md` file containing the project's name and description. I put all project documentation into the readme file for most projects, thus these files are not left empty. I usually also include badges into readme files about build status, versioning and license information, as well as code quality and code coverage data.

Code of conduct

Code of conduct documents formulate a set of ethical norms and responsibilities on practices of collaboration. Diplomatq uses the version 1.4 of the Contributor Covenant's Code of Conduct across all its projects, in a file named `CODE_OF_CONDUCT.md`. Even though currently there is no open-source collaboration in Diplomatq's repositories, once we get there, I want to take the Code of Conduct very seriously, in order to create a welcoming and inclusive development environment. I have already configured the *conduct@diplomatq.org* email address for reporting unacceptable behaviour.

Contributing information

The document `CONTRIBUTING.md` — named in line with GitHub's open source recommendations — summarizes how to contribute to Diplomatq projects. It describes means of communication with project maintainers, and methods of requesting features and reporting bugs. It formulates a set of submission guidelines for issues and pull requests. It also presents a style guide for the code itself, and for commit messages. As I adopted the Angular-style Conventional Commits specification [78], the document introduces the concept conventional commit messages in an example-based manner.

Pull request template

In order to discourage future contributors from submitting incomplete work, I prepared a checklist as part of GitHub's pull request template. As the checklist is saved into the file called `.github/PULL_REQUEST_TEMPLATE.md`, GitHub displays it as the initial contents of the to-be-submitted pull request's details field. The checklist requires:

- the pull request to consist of one commit;
- the commit message to follow the commit message guidelines;
- the tests to be updated (if applicable);
- the documentation to be updated (if applicable).

Beyond the checklist, the template instructs the developer to indicate the pull request's type, describe the application behavior the pull request modifies, describe the new behaviour, and indicate if the pull request introduces a breaking change.

Issue templates

In open-source projects, it is common that developers submit issues without describing clear and concise details. To discourage this routine, I created two types of issue templates. When developers want to create a new issue in a repository, they are offered these issue templates to choose from. The *bug report* template encourage developers to detail the perceived failure as much as possible, and to disclose the execution context, expected behaviour and steps of reproduction. The *feature request* template supports the issuer in describing their exact requirements along with considered solutions, if there are any.

.editorconfig file

EditorConfig helps maintain consistent coding styles for multiple developers working on the same project across various editors and IDEs. The EditorConfig project consists of a file format for defining coding styles and a collection of text editor plugins that enable editors to read the file format and adhere to defined styles [79].

Security policy

Since exploiting security vulnerabilities can lead to catastrophic consequences like data breaches or loss of data, security issues are to be handled carefully and discreetly. A security policy establishes rules and methods for reporting a security issue in a development project to the maintainers without causing any harm. Diplomatiq's security policy requires contributors to report found security issues to the `security@diplomatiq.org` email address, possibly encrypted with Diplomatiq's public PGP key.¹⁵

¹⁵Diplomatiq's public PGP key is available on the <https://www.diplomatiq.org/pgp-key.txt> URL.

4.7.3 Development and release model

In most of Diplomatq's repositories, I use the a modified version of the GitFlow workflow [80]. In essence, the workflow operates on four types of branches in a repository:

1. The *master* branch is stable, and should be kept stable at all times. Code gets merged into the master branch only from a *release* branch, immediately before a new version of the software is released into production.¹⁶ Contributors can not push code directly onto this branch, but administrators can.
2. The *develop* branch is part of the repository's regular development flow. When creating feature branches, contributors branch from develop, and the target of contributonal pull requests is the develop branch. Contributors can not push code directly onto this branch, but administrators can.
3. The *feature/** branches are also parts of the regular development flow. Contributors develop their contributions on feature branches, regardless of the contribution's type (feature, bugfix, refactor, etc.). Contributonal pull requests are created from feature branches targeting develop.
4. The *release/** branches are created when a release candidate is being prepared, by branching from develop. Release-related code changes like version bumps, changelog generation and release hotfixes are committed onto the release candidate's release branch, and eventually the finalized release candidate is tagged. After conducting all necessary testing on the release candidate, the release branch is merged into master, then it is immediately released into production by continuous delivery. After the deployment finished, the release branch is merged into develop. Contributors can not push code directly onto release branches, but administrators can.

Figure 4.3 displays the above workflow, which can be well supported by GitHub's continuous integration and continuous delivery capabilities and branch protection rules. I created such rules for the *master*, *develop*, and *release/** branches. All branches require certain status checks to pass before merging code into the mentioned branches, and require the merged branches to be up to date when merging. As a general rule, I also require linear history in Diplomatq's repositories, because I experienced that this way tasks such as automated changelog generation or finding regressions with `git bisect` becomes easier.

I implemented a versioning strategy following the scheme of semantic versioning. This scheme defines the software version as a numeric triplet separated by dots in the form MAJOR.MINOR.PATCH. The MAJOR version is to be incremented in case the version contains incompatible API changes, the MINOR version is to be incremented if the version's added functionality is backwards compatible, and the PATCH version is to be incremented if the version solely consists of backwards compatible bug fixes [81]. Adhering to the above

¹⁶As described later, most Diplomatq projects are configured in a way that merging a tagged commit into the master branch triggers the continuous delivery flow and deploys the version into production.

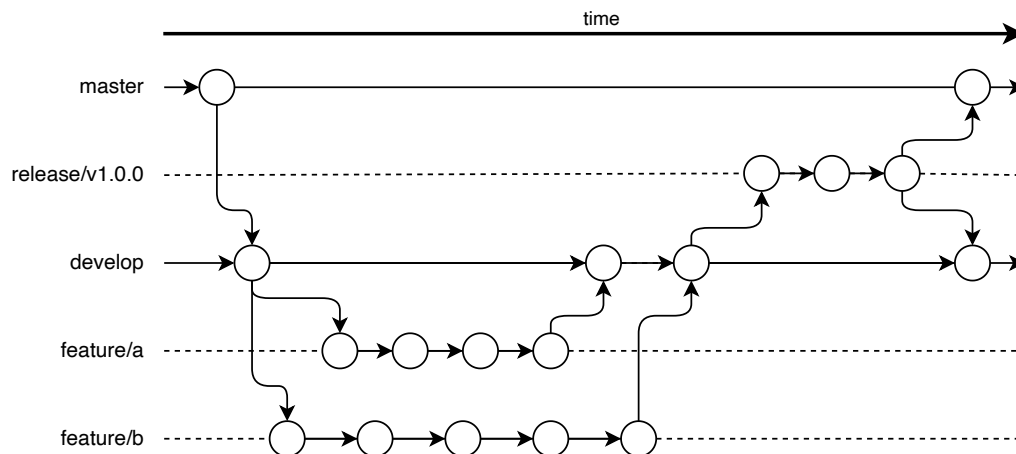


Figure 4.3 Diplomat's modified version of the GitFlow workflow

helps managing software versions and dependencies with a lesser chance to introduce unintended breaking changes.

As Diplomat's development and release model features both conventional commits and semantic versioning, it was straightforward to implement automated changelog generation as well. Using Conventional Changelog Tools [82], I automatically generate each released version's changelog from the commit messages contained by the version.

4.7.4 Automated dependency management

As development and release processes become more and more automated, the time frame required for releasing new software versions reduces. With continuous delivery, pushing new code into the remote repository often means that the code is released to production without further developer interaction, after all necessary automated tests pass. This results in the need of more frequent dependency updates as well. Maintainers can either check and update every dependency manually, or they can apply automated solutions.

I configured Dependabot for the automatic maintenance of dependencies across Diplomat projects. Dependabot is a GitHub-owned, integrated tool for watching package managers like NPM and Maven, continuously checking for new versions published. I set up Dependabot in a way that if it detects one of the dependencies in Diplomat projects having a new version released, it opens a pull request in the related project with the dependency update. This automated workflow frees me from a lot of manual work, and also ensures that the dependencies of Diplomat projects are always up to date. Pull requests containing security-related updates are separately tagged, allowing me to handle them with higher priority. As I do not want to lose the ability of reviewing dependency updates,

merging pull requests opened by Dependabot is not automated. But since continuous integration workflows are applied to all pull requests, all checks and tests defined in the repository are run for Dependabot's updates as well as for any other contribution.

4.7.5 Continuous integration and continuous delivery

Martin Fowler defines continuous integration (CI) as a software development practice where team members integrate their code changes into the repository mainline frequently, at least daily, and each integrations are verified by a set of automated build and test methods in order to detect integration errors as soon as possible. According to Fowler, this leads to less integration problems, and faster development and release cycles [83].

In case of Diplomatq projects, the repository mainline is the develop branch, and the integrations are essentially pull requests opened by repository contributors. The set of automated build and test methods applied to pull requests vary across projects according to the chosen technology, language, and framework, but Diplomatq libraries and production software usually contain automated tests for verifying conformance, functional correctness, and adherence to various code quality metrics.

I configured all projects to use GitHub Actions [84] for continuous integration (and continuous delivery). Previously I used the mature and well-supported Travis CI [85], but encountering constant stability problems with Travis made me to switch to the then freshly announced GitHub Actions. Since the switch, I experienced GitHub Actions to be a well-founded, stable solution. As it is an integral part of the GitHub platform, it furthers the possibilities of integrations by allowing to subscribe to more repository events than external platforms [86]. It also enables to utilize GitHub's REST and GraphQL APIs for extending the base functionality of GitHub Actions with built-in authentication — a feature I make use of in almost all projects.

GitHub Actions essentially allows to create custom, automated *workflows* for various phases of the software development lifecycle. Such workflows are declaratively defined in a YAML¹⁷ files, stored in the repository alongside the project's sourcecode. Workflows are triggered by GitHub's *events*, and consist of *jobs* able to run sequentially on different runners having different operating systems. Jobs encompass *steps*, the smallest work unit within a workflow. Steps are either predefined *actions* stored in public repositories and created by GitHub members, or shell scripts.

Although different projects require different CI workflows, I have established the following baseline for checking contributions across Diplomatq projects:

- If the workflow was triggered by a pull request, check if the pull request consists of one commit. This involves substeps of querying the number of commits for the pull

¹⁷Yet Another Markup Language is a markup language for human-readable configuration and serialization.

request through GitHub's GraphQL API, extracting the number of commits from the JSON response, and checking if the number is equal to 1.

- Check if the commit message conforms to the requirements of Angular-style conventional commits. The commit message is linted with the *commitlint* tool, part of the previously mentioned Conventional Changelog Tools.
- After setting up the project's programming language environment and installing all necessary dependencies, perform code linting in order to verify that the submitted code conforms to the required code styling of Diplomatq. The code is linted with language-dependant tools.
- Verify that the project can be built by creating a distribution deployment artefact.
- Run all unit, integration, and end-to-end tests of the project, if there are any.
- Scan the code with SonarQube, and upload the results to SonarCloud.

If any of the above steps fails, the contribution should be considered as faulty, and it should not be accepted into the mainline. Since all Diplomatq projects are open-source, and GitHub Actions is offered free of charge for open-source projects, I usually run workflows incorporating the above steps against multiple platforms, operating systems, and multiple versions of programming language environments concurrently, without additional costs.

I also utilize GitHub Actions for continuous delivery. Most projects are configured in a way that pushing a tagged commit to the master branch triggers the release process. The process consists of producing and uploading a distribution build artefact to GitHub's permanent artefact storage, running all tests against all environments on the artefact, and if every step is successful, deploying the artefact into the production infrastructure.

4.7.6 Code analysis with SonarCloud

SonarSource's¹⁸ SonarQube¹⁹ along with its cloud platform, SonarCloud²⁰ are tools for inspecting code quality based on static code analysis. Both examine the code using the same configurable — and in case of SonarQube, extensible — analysis ruleset in order to detect bugs and security vulnerabilities. SonarQube is to be used in on-premise, SonarCloud is to be used in cloud-based development environments, and both can be included in continuous integration environments.

As it is free to use for open-source development, I configured SonarCloud for most Diplomatq projects. After creating the Diplomatq organization on the platform, I imported the Diplomatq's GitHub repositories into SonarCloud through an automated flow. I created a token for each project, which authenticates the project against SonarCloud's API when performing analyses and uploading results.

¹⁸<https://www.sonarsource.com>

¹⁹<https://www.sonarqube.org>

²⁰<https://sonarcloud.io>

SonarCloud introduces the concepts of *quality profiles* and *quality gates*. A quality profile is a set of language-dependant quality rules, which get applied against the codebase during an analysis. Profiles are categorized into four parts:

1. *Bugs* are development errors which occasionally result in faulty software.
2. *Vulnerabilities* are pieces of code that causes the software to become vulnerable against attacks.
3. *Code smells* are bad coding practices which make the code harder to maintain.
4. *Security hotspots* are subject to become vulnerabilities if applied incorrectly.

An example of a quality rule categorized as a bug in the Java quality profile is the reversion of the “not equals” operator (incorrectly using `= !` instead of the correct `!=`): the code compiles, but the operator does not produce the intended results. Quality profiles can be customized by including or excluding specific rules. When using SonarQube, new quality rules can also be implemented for several languages, but SonarCloud can not be extended with custom rules [87]. For Diplomatq projects, I use the quality profiles recommended and maintained by SonarSource.

A quality gate is a set of boolean conditions based on measurement values. Such conditions are not dependent on any language or platforms, since it formulates general measurements, such as code coverage should be kept above a specific percentage, the number of bugs should be equal to zero, or security rating should be the highest. A quality gate “lets through” code which complies with all its requirements. For Diplomatq projects, I use a custom quality gate with the highest and strictest possible settings on the platform.

As I configured SonarCloud to be part of the continuous integration workflows of Diplomatq projects, only such pull requests can be merged into a Diplomatq repository, which passes all requirements I set in SonarCloud. Since the CI rules of GitHub projects apply to me as well as any other future contributor, this enforces even me to write code with consistently high quality and 100% test coverage.²¹

4.8 Node Package Manager (NPM)

4.8.1 Overview

The Node Package Manager (NPM)²² — part of an ecosystem built upon the JavaScript runtime environment Node.js — is a package manager for software written in JavaScript. NPM’s online package registry containing public and private packages enables developers to download and re-use shared packages by incorporating them into their own software in

²¹I always strive to produce high quality code and avoid bugs. However, SonarCloud helped me to notice two minor bugs so far, which were not reported by tests, even with 100% test coverage.

²²<https://www.npmjs.com>

a versioned manner, with the help of NPM's command line client. It has become the de-facto standard solution for sharing JavaScript software in the open-source community [88]. Currently, NPM has over 1,300,000 packages [89].

4.8.2 Creating and securing an organization

I implemented several JavaScript libraries to be published as open-source software, but in Diplomatq's name instead of my personal account. For this, I created an organization on NPM, similarly to GitHub. Organizations can publish packages in a scoped way: the name of the organization prefixed with a "@" character — the scope name — becomes the prefix of the package's name. The scope and the package name are separated with a "/" character. For example, the `project-config` package published under the Diplomatq organization gets finally named `@diplomatq/project-config`²³.

NPM offers no security policies for organizations. As a security measure, all I was able to do was to enable two-factor authentication in my account for publishing packages as well, as it was of course already enabled for authentication. This means that if I invite a user into Diplomatq's NPM organization, they will be able to publish packages even if they do not have two-factor authentication enabled. This way, an attacker stealing the password of one of the organization's future accounts can easily compromise Diplomatq packages. I contacted NPM's customer support about mitigating this future threat, and I was told that the functionality of organizational policies are on their development roadmap, expected to be released to production in the fourth quarter of 2020. Until then it is not likely that other developers will join Diplomatq's NPM organization, meaning the lack of security policies means no imminent security threat.

4.9 Communication and social media presence

4.9.1 Gitter

I created a space for Diplomatq on Gitter²⁴, a chat service for GitHub repositories and developers. The space was named Diplomatq, according to the primary naming rule in Section 4.2. Gitter allows to create several chat rooms within Diplomatq's space, corresponding to the repositories within Diplomatq's GitHub organization. As advised in Diplomatq's contributing guide, the Gitter space should be the primary place for discussing questions and problems regarding Diplomatq projects. At the time of writing this thesis, I am the sole member of Diplomatq's Gitter space, and there has been no communication in the chat rooms.

²³NPM scopes can be lowercase only, so according to Diplomatq's naming rules described in Section 4.2, the organization's scope has been `@diplomatq`.

²⁴<https://gitter.im>

4.9.2 Slack

For future internal communication, I registered a workspace for Diplomatiq on Slack²⁵, one of today's most popular business communication platforms. The workspace was named Diplomatiq, but due to the *diplomatiq.slack.com* domain being already taken, its domain name has been *diplomatiqorg.slack.com*, according to the 4th naming rule in Section 4.2. Even though the workspace is unused as I am its sole member, reserving *diplomatiqorg.slack.com* ensures that the domain name will not be taken by another company. As a security measure, I made it mandatory to use two-factor authentication.

4.9.3 Facebook

I created and published a Facebook²⁶ page for Diplomatiq, reserving the Facebook page path <https://www.facebook.com/DiplomatiqOrg>, according to the 3rd naming rule of Section 4.2. The page has no posts and contains no data, but I uploaded one of Diplomatiq's application icon as its profile picture. In the future, I want to heavily build on Diplomatiq's Facebook presence for reaching the younger generation of Model United Nations.

4.10 Procuring the Neo4j database software

4.10.1 Licensing overview

I chose the Neo4j graph database as the main database serving the Diplomatiq social network application. The reasons of my decision are detailed in Section 4.11.10, this section focuses on licensing and the procurement of the database software.

As already mentioned in Section 3.3.2, Neo4j offers two software editions with different feature sets and licensing [51]. The Community Edition is open source, available under the terms of the GPL v3 license. The Enterprise Edition offers four licensing options:

- The commercial license permits using the software in closed source applications as well, and is offered under a paid subscription agreement.
- The developer license permits free local development usage after registration.
- The evaluation license is a time-restricted variant of the commercial license, offering free usage for a fixed-term trial period.
- The startup license permits using the software for free for startups having at most 50 employees and \$3 million of annual revenue, but limits the number of database instances to 3 production, 3 staging, and 6 testing/development machines, each having 256 gigabytes of RAM and 24 processor cores at most [52].

²⁵<https://slack.com>

²⁶<https://www.facebook.com>

Besides the software itself, Neo Technology offers another approach for using their graph database: Neo4j Aura is a managed graph DBaaS²⁷, offering on-demand scaling, a capacity-based pricing construction, and clustering features for high availability.

4.10.2 Acquiring the startup license

After evaluating licensing options and the Neo4j Aura service, I decided that since Aura is too expensive for my current budget, I enrolled in the startup program. This involved filling a registration form with personal data and information about the (prospective) company. Two days after submitting the form, I received the email with a license key for Neo4j Enterprise Edition. This enabled me to download and deploy the software onto Diplomatiq's server infrastructure.

4.11 Building and securing the server infrastructure

4.11.1 Platform overview

In this section, I use the term server infrastructure to denote the various servers, software, networking solutions, and configurations in Microsoft Azure that enables and supports the public operation of the Diplomatiq social network software system on the Internet (excluding DNS and emailing).

4.11.2 Creating a directory for Diplomatiq

Overview

In enterprise infrastructures, the biggest organizational unit is usually called a *directory*. A directory encompasses the entire organization with its users, computing resources, and often includes information about premises and office resources as well. In Microsoft Azure, a directory — also called *directory tenant* — is managed via the service called Azure Active Directory (AD).

Creating a personal directory with my personal Azure account

I could not sign up to Azure by simultaneously creating a directory dedicated to Diplomatiq (I was later told by support that this is only possible by signing up through Microsoft salespersons, which I intentionally avoided). Instead, I needed to create a personal Microsoft account with my personal email address, which initialized my personal directory and set its domain name to *luczsomagmail2559.onmicrosoft.com*, based on my email address.

²⁷ database as a service

Creating a directory dedicated to Diplomatiq

For Diplomatiq's own, dedicated directory, I needed to create a separate directory tenant from my personal directory. For this dedicated directory tenant, I set Diplomatiq as the organization name, and as its initial domain name, I set *diplomatiq.onmicrosoft.com*. As I created the new directory tenant from my personal Microsoft account, the *global administrator* — the highest possible role able to manage all services and everything else in a directory — of the new tenant became my personal account.

Setting diplomatiq.org as the primary domain in Diplomatiq's directory

In order to be able to invite users with *@diplomatiq.org* email addresses, the ownership of the *diplomatiq.org* domain needed to be verified to Microsoft. I registered *diplomatiq.org* as a custom domain name for the directory, and proved the administrative ownership of the domain by adding a Microsoft-provided identifier as a DNS TXT record. After the verification, I set the domain to be the primary one, leaving *diplomatiq.onmicrosoft.com* as a secondary directory domain.

Creating a company Azure account and making it administrator

After Diplomatiq's domain name was verified in Azure, I was able to create a new user in Diplomatiq's dedicated directory with the *soma.lucz@diplomatiq.org* email address. I assigned the global administrator role to this new account, making it administratively equivalent to my personal account.

Deleting the personal account

After I had completely set up Diplomatiq's dedicated directory and had configured its *soma.lucz@diplomatiq.org* user to be a global administrator, there was no need for keeping my personal Azure account, so I deleted it from the directory. Later I deleted the account itself, which deleted its personal directory as well.

Further directory settings and security considerations

I performed further configuration to restrict future users' access within the directory in order to keep responsibilities limited and separated. I disabled user application registration, to disallow arbitrary users to issue application credentials accessing resources in the directory. I restricted the access to the AD's administration portal, so users are not able to modify AD settings. I made it mandatory for all future users in the directory to use two-factor authentication.

4.11.3 Naming conventions

Microsoft Azure has a detailed guide as part of its Cloud Adoption Framework on how to name management groups, subscriptions, resource groups, and resources themselves [90]. Since Azure does not allow to rename a resource after its creation, I wanted to establish a solid system of naming conventions. I downloaded the naming and tagging convention tracking template provided by Azure, and decided to stick to it. As the table does not contain conventions for all resource types, I sometimes needed to extend it with new rules. Henceforth every resource name I introduce was named according to Diplomatiq's naming conventions.

4.11.4 Creating a subscription and a support subscription

Within a directory — the largest organizational unit — there can be several management groups nested into each other, as the second level of resource organization. Management groups contain subscriptions (or other management groups), within subscriptions there are resource groups, and resource groups consists of resources. Since I decided Diplomatiq only needs one subscription to encompass all resources, its directory would make no use of management groups.

I set up the *diplomatiq-prod-001* subscription paid by my credit card as a pay-as-you-go subscription. This allows me to pay for the resources I used in a time-based manner: the more resources I use (or the higher their pricing tiers are²⁸), the more I pay. I additionally subscribed to the *Developer support plan* later to resolve a networking issue with the Key Vault service.

4.11.5 Structuring resources

As part of its global infrastructure, Microsoft Azure provides services in more than 60 *regions* worldwide. A region is a set of datacenters within a *geography* (usually a country), which acts as a data residency boundary [91]. Deploying infrastructure elements across several geographies allows building high-availability, low-latency systems — the closer the infrastructure is to the client, the quicker the content can be delivered over the Internet.

Regions are bound to resource groups. Since the current needs of Diplomatiq do not justify deploying multi-region services, I created only one resource group in the Microsoft's North Europe (Ireland) region, named *rg-diplomatiq-prod-001*. All resources presented later were deployed into this resource group. I chose the North Europe region because it is one of the oldest and best-supported Azure regions, and it is also among the regions having the highest service coverage in the Azure infrastructure. Later this decision proved problematic: North Europe is also one of the most popular regions, and due to the heavy interest in

²⁸Higher pricing tiers offer more features or higher performance.

cloud solutions raised by the COVID-19 pandemic, Microsoft introduced limitations on new infrastructure deployments. Because of this, initially I was not able to deploy any virtual machines into the region, and this prevented me from deploying a machine for the Neo4j graph database as well.

4.11.6 Securely storing secrets and credentials

Maintaining an IT infrastructure involves dealing with several kinds of credentials, keys, and certificates. Most of these are highly sensitive secrets; letting an attacker steal them could cause critical service disruption or even permanent damage.²⁹ Microsoft Azure offers its Key Vault³⁰ service for storing secrets in a secure and audited manner, guarded by configurable access policies.

The Key Vault service is offered in two tiers:

- The *standard* tier provides all security and auditing functionalities of the Key Vault service, except for storing keys in Hardware Security Modules (HSMs).³¹
- The *premium* tier provides the optional usage of Hardware Security Modules for storing keys, in addition to all features offered by the standard tier.

Diplomatiq's infrastructure incorporates several kinds of secrets, which I decided to store in the Key Vault. Since the two tiers' prices differ only minimally, I chose the premium tier, offering the optional usage of HSMs. I created two Key Vault instances. The *kv-sslcerts-prod-001* instance stores TLS certificates necessary for secure communication with clients and across services. The *kv-prodcreds-prod-001* instance stores everything else: database access credentials, database encryption keys, and API keys of integrated services.

Key Vault access policies can be fine-tuned in various ways. Each entity within the directory can have a separate set of access to the actual cryptographic operations performed by Key Vault instances. I configured the access policies of the instances to be as strict as possible: service identities³² required to access entries as part of their normal operation are able to *get* an entry by referencing it by its identifier, but they can not *list* entries, and they have no write access at all. As an additional security measure on top of authenticated and audited access, the Key Vault instances are not reachable from the Internet, only from a specified internal subnet the backend services are deployed to.

²⁹Even if attackers can not directly use the breached access credential, a responsible company assumes that it was used.

³⁰<https://azure.microsoft.com/en-us/services/key-vault>

³¹As mentioned earlier, Hardware Security Modules are separate physical computers designed to keep cryptographic keys safe. They offer tamper resistance making it extremely difficult to extract and steal secret keys [69].

³²Azure provides a built-in way to identify and authenticate a service in a directory, called *service identity*.

4.11.7 Setting up the infrastructure for the website

Overview

Diplomatiq's website can be reached on the address *www.diplomatiq.org*. I created the website for conducting future marketing operations, and to introducing the company. The website is separated from the Diplomatiq application itself, available on *app.diplomatiq.org*.

Serving a website over the Internet requires a public-facing webserver or — if the website consists of only static elements without application logic implemented on the server — a serverless, static web storage. Both approaches have benefits: running a webserver allows a deeper level of configuration, but a static web storage can more easily be replicated around to world to enable serving websites with lower latency. Due to lack of important security capabilities of static web storage technologies in Azure — e.g. adding headers for implementing Content Security Policy³³ —, I decided that the website infrastructure should be implemented on a webserver-based approach.

Microsoft Azure provides a mature Platform-as-a-Service (PaaS) technology called *App Service*. In contrast to Infrastructure-as-a-Service (IaaS), one does not need to maintain the infrastructure itself when using a PaaS: the underlying computers are abstracted, and the operating system and server software are continuously updated. App Services are essentially managed PaaS infrastructures with built-in scaling, load balancing, and security features. As I have relevant work experience on the maintenance of production App Services, I decided to use an App Service for serving Diplomatiq's website.

Creating a suitable App Service Plan

Azure requires having an App Service Plan for deploying App Services. While App Services are in essence instances of web applications (consisting of multiple actual servers and infrastructure, which is abstracted and invisible to the user), App Service Plans can be described as server farms, which can comprise several App Services. App Service Plans are offered in several pricing tiers, each providing different hardware — either in a shared or isolated virtualized environment — with differing performance, and varying features.

I created an App Service Plan named *plan-diplomatiq-prod-001*, and determined its pricing tier based on the following criteria:

- It should support HTTPS. Since all pricing tiers support HTTPS, this requirement does not exclude any tiers.
- It should support adding custom domain names, so the application can be served from *diplomatiq.org* instead of a domain name ending with *azurewebsites.net*. This

³³Content Security Policy is an important security measure for websites and web application, mitigating certain types of attacks, such as Cross-Site Scripting (XSS) [92].

excludes smaller tiers targeted for development or testing usage.

- It should support multiple deployment slots³⁴ for conducting testing in production³⁵, and swap-based, zero-downtime deployment.³⁶

The cheapest tier supporting the required features listed above was the smallest tier targeted for production use, called S1, therefore I chose this for *plan-diplomatiq-prod-001*.

Creating and configuring the App Service

I created an App Service for the website named *app-diplomatiqwebsite-prod-001*, and deployed it into the *plan-diplomatiq-prod-001* App Service Plan. I configured it to be reachable under the *diplomatiq.org* and *www.diplomatiq.org* domain names. This involved adding a CNAME DNS record for the *www* subdomain with a value specified by Azure, and an A DNS record for the apex domain, pointing to the public IP address of the App Service.

I chose the *canonical domain name* of Diplomatiq's website to be the *www.diplomatiq.org*. Due to security, load-balancing and administrative issues, choosing a canonical name and sticking to it is an important aspect of maintaining a web application [93, 94]. I configured the platform in a way that requests targeting the *diplomatiq.org* apex domain also reach the web server, but the server should issue a *permanent redirect* to *www.diplomatiq.org* with the HTTP status code 301 in this case.

After setting up the domain names, I configured the App Service's TLS settings. I configured the webserver to mandate clients to use HTTPS with the 1.2 version of the underlying TLS protocol³⁷. In order to use HTTPS, I needed to provide the previously introduced Sectigo TLS certificate issued to the *diplomatiq.org* and *www.diplomatiq.org* domains. I uploaded the certificate along with its password-protected private key into the *kv-sslcerts-prod-001* Key Vault, and used the automated attachment process to bind them to the App Service.

Zero-downtime deployment

At this point, the infrastructure for securely serving a website was ready. In order to be able to deploy new application versions without downtime, I cloned the default *production* deployment slot with all its settings, and created a *staging* slot. With the swap-based

³⁴A deployment slot is a separated instance of the same web application available under the same domain name. The *production* slot gets served by default, and the other slots can be visited by sending a cookie along the request, which instructs the load balancer in front of the slots to route the request to the targeted one.

³⁵Testing in production involves running the System Under Test (SUT) under the same conditions as the SUT runs normally in the production environment.

³⁶With swap-based deployments, a new version is deployed into another deployment slot, then the slot gets swapped with the production slot to receive all incoming traffic, without downtime.

³⁷Mandating the client to use HTTPS means that requests using the HTTP protocol are redirected to *https://www.diplomatiq.org* with a *permanent redirect* having HTTP status code 301. It also means configuring HSTS properly, described later.

deployment process provided by App Services, new website versions can be deployed onto the staging slot, and — as the users' incoming requests are routed to the production slot by default — the staging slot can be reconfigured then restarted according to the new version's demands, without causing service disruption. After the necessary reconfiguration is finished, and the staging slot's server instances are warmed up for receiving production workload, the staging slot can be swapped into production without downtime. The automated swap operation essentially consists of modifying the configuration of the slots' load balancers to route incoming production traffic to the new slot. This reconfiguration is performed without restarting any infrastructure elements, thus without downtime.

The deployment itself is performed as part of the continuous delivery workflow introduced earlier. Azure provides a set of configurable actions (as in GitHub Actions) to be incorporated into automated deployment procedures. I integrated such an action into the website's deployment workflow, which automatically deploys the built artefact into Azure. The deployment is authenticated with the App Service's publish profile, which is embedded into the repository as a GitHub encrypted secret [95].

HTTP Strict Transport Security (HSTS) preloading

Compatibility reasons justify that web servers also serve entry points for web applications over HTTP as well as HTTPS, so a user visiting e.g. *http://www.diplomatiq.org* (note the usage of HTTP instead of HTTPS) is eventually redirected to the website instead of receiving an error. The primary and only goal of HTTP entry points should be to redirect the user to the HTTPS entry point of the application. The *HTTP Strict Transport Security (HSTS)* browser mechanism furthers this approach by allowing servers to specify a *Strict-Transport-Security* header in their response, instructing the browser to permanently remember that the site should only be visited over HTTPS in the future [96]. If an HSTS entry exists in the browser for a given site, and the user tries to visit the site over HTTP, the browser automatically upgrades the connection to HTTPS, before even making an actual request to the site.

HSTS is based on the *trust on first use* principle, meaning its protection only applies after a user visited the site the at least once. This means that if an attacker hijacks the very first HTTP request a user made to a site, they can impersonate the server and serve malicious content to the user [97]. Major web browsers, such as Google Chrome, Mozilla Firefox, Apple's Safari, Microsoft Internet Explorer 11 and Microsoft Edge solve this problem by packaging a HSTS preload list into their application, specifying sites supporting HSTS [98].

I have enabled HSTS for all Internet-facing Diplomatiq services, and also added the *diplomatiq.org* domain name (and implicitly all its subdomains) to Google Chrome's HSTS preload list.³⁸ Most major web browsers build their own lists upon this list [98].

³⁸The automated submission process can be initiated on <https://hstspreload.org>.

4.11.8 Setting up the infrastructure for the front end application

Overview

In terms of infrastructure requirements, Diplomatq's front end application is very similar to the website. Since the application is a single-page client application, it does not need any back end logic, apart from routing and security capabilities. I created the same infrastructure with the same configuration for the front end application as for the website. It is hosted by an App Service in the same App Service Plan as the website, and its deployment model is also the same as the website's. There are two differences: it is served under a different subdomain, and it has one more deployment slot besides production and staging.

Subdomain settings, acquiring a new TLS certificate

I set up the resource to be served under the *app.diplomatq.org* domain name. Similarly to the website, this involved adding a CNAME DNS record for *app.diplomatq.org* with a value provided by Azure. As the Sectigo TLS certificate is issued to the *diplomatq.org* and the *www.diplomatq.org* domain names, it can not be applied to this resource. For being able to serve the application over HTTPS, I needed to acquire a new TLS certificate for its subdomain. Via a fully automated process, I created a managed TLS certificate offered by the App Service³⁹, saved it into the *kv-sslcerts-prod-001* Key Vault, then attached it to the App Service. As their name suggests, managed certificates are automatically managed and renewed by Azure [100], freeing me from further maintenance tasks related to TLS.

Deployment slots

As detailed earlier, the development model of Diplomatq features a branch named *develop* as its semi-stable mainline branch for day-to-day development. Besides the *production* and *staging* slots, I created a deployment slot named *develop* as well. After a pull request is merged into the *develop* branch in the front end project's repository, the *develop* branch is automatically released to the *develop* deployment slot. This is similar to the concept of nightly builds, only in this case updates can happen more frequently, as more pull requests are merged into the *develop* branch.

4.11.9 Setting up the infrastructure for the back end application

Overview

The back end application of the Diplomatq social network is exposed to clients by an API available over HTTPS. The API and the application itself will be detailed in Chapter 6, but since the application needs a Java runtime environment, it evidently needs a webserver

³⁹Azure managed TLS certificates are issued by DigiCert [99], thus I needed to add another CAA record to Diplomatq's DNS settings to allow DigiCert issuing a certificate to *app.diplomatq.org*.

as well, which forwards the incoming requests to the application. Similarly to the website and the front end, I decided to use an App Service for hosting the back end application.

Subdomain settings and another new TLS certificate

I set up the resource to be served under *api.diplomatiq.org* domain. As the Sectigo TLS certificate can not be applied to this resource either, I created a managed TLS certificate and saved it into the *kv-sslcerts-prod-001* Key Vault via the same process I already mentioned.

Deployment slots

The application features the same three-slots deployment model as the front end application, with the same naming and same semantics.

Securing configuration values and production credentials

The maintenance of a back end server application often involve managing configuration values and secrets, such as encryption and API keys, and database access credentials. The naive and catastrophically insecure way of managing secrets is embedding them in the code, allowing attackers to harvest and exploit them [101, 102]. Besides security issues, embedding secrets or infrastructure configuration in the code leads to a tight coupling between the infrastructure and the application, and it causes that the application needs to be redeployed in case of key rollovers or infrastructural changes. For mitigating security threats and tight coupling, I embedded neither configuration values, nor secrets into the application's code base — apart from the “dummy” secrets, which make local development easier. Instead, I use all such values referenced as environment variables.

In Azure, App Services' environment variables can be set on the service's graphical management interface, or their values can be provided by code-based application configuration⁴⁰. Providing environment variable values in code would be against the point, since that code would still be committed into the repository. Therefore I set all such environment variables on the App Service management interface.

App Services offer another level of secret abstraction by referencing Key Vault entries in environment variables. App Services substitute specially formatted reference strings found in environment variables with the actual value of the referenced Key Vault entry *at runtime*. This way, sensitive secrets are not copy-pasted between services, and stored only one, centralized and audited, secure location.

When configuring the back end service, I placed insensitive configuration values (such as the database's name, URI and username) directly into environment variables, but sensitive

⁴⁰There are more advanced, more scalable solutions, such as the Azure App Configuration service [103], which I did not use and do not detail in this thesis.

secrets (such as the password and data encryption keys of the database, and the API key for SendGrid) are referenced values stored in the *kv-prodcreds-prod-001* Key Vault.

Creating multiple isolated back end environments

I created two separated back end environments in order to be able to conduct safe testing, without compromising user data. The *production* environment encompasses the back end application instances running on the production and staging deployment slots, storing data in the *diplomatiqbackend-production* database. The *development* environment comprises the instance running in the develop slot, and it stores data in the *diplomatiqbackend-develop* database. Even though the two environments are not completely isolated from each other, as the two databases are stored on the same virtual machine, the *development* environment can be regarded as a sandbox, which can not cause any harm to the *production* environment. Storing configuration values separated from application code made supports multiple deployment environments well, since this way the exact same application can be deployed into both environments without needing compile-time reconfiguration.

4.11.10 Setting up the database infrastructure

Overview

Apart from Neo Technologies' own database service provider called Neo4j Aura [104], no cloud service providers offer Neo4j in a database-as-a-service construction. For using Neo4j, I needed to purchase a virtual machine, install, configure and secure Neo4j Enterprise Edition, set up the databases, and perform the required network and access configuration across the back end service and the database machine.

Procuring the virtual machine

As I mentioned earlier, last months' high interest in cloud computing caused by the COVID-19 pandemic put a heavy load onto one of Azure's most popular regions, North Europe, where Diplomatiq's infrastructure is hosted. Microsoft introduced restrictions on new infrastructure deployments in the region, which prevented me from acquiring even a single instance of a small-sized virtual machine. Eventually the support allowed me to procure a virtual machine from one of the smallest pricing tiers, offering limited computing and IO performance only. This proved to be a serious bottleneck — the database answers slowly even to simply queries — but this was still a better solution than migrating Diplomatiq's entire infrastructure into another region.⁴¹ After the restrictions are lifted, I will upscale the database instance into a higher pricing tier offering more performance.

⁴¹Microsoft Azure does not support moving resources across regions for most of its resource types [105]. The migration process would have meant recreating and reconfiguring the entire infrastructure in another region.

Installing, configuring and securing Neo4j Enterprise Edition

The procured virtual machine came with the 18.04 version of Ubuntu operating system, as I requested. Having experience with securing Ubuntu servers, I set up and secured SSH-based authentication and disabled password-based logins, and I installed and configured *ufw* (*Uncomplicated Firewall*) to let through only the ports 22 (for SSH), 7443 (for Neo4j HTTPS administration), and 7687 (for Neo4j binary communication).

I installed Neo4j Enterprise Edition, and configured its HTTPS connector to accept connections on the *neo4j.diplomatiq.org* domain name. I created a public IP address for the virtual machine in Azure, and registered a DNS A record pointing the domain name to the IP address. For HTTPS connections, I also needed to acquire a valid TLS certificate issued to the domain name. For automated certificate management and renewal, I used the Electronic Frontier Foundation's *certbot* [106], which deploys free TLS certificates from Let's Encrypt, an open certificate authority [107]. I also configured its Neo4j's Bolt⁴² connector to accept encrypted database connections only. I disabled the insecure HTTP connector.

On Neo4j's interactive web interface available on *neo4j.diplomatiq.org*, I set up my own administrative user — with a long, cryptographically random password — to access the database, and deleted the default *neo4j* user. Even though at this point both the database machine and Neo4j itself were properly secured, as an additional security measure, I sealed the machine off the internet with the help of Azure networking rules, making it available only within its own subnet, and another dedicated subnet.^{43,44}

Setting up databases with dedicated users

For the *production* and the *development* environments mentioned earlier, I set up two different graph databases within Neo4j, *diplomatiqbackend-production* and *diplomatiqbackend-develop*. I created a dedicated user for both databases, ensuring that the environments are separated along database credentials as well as all other configuration. Since Neo4j is a schemaless database, and indices and constraints can be automatically managed by Neo4j-OGM, the object-graph mapping framework I used at implementing the Diplomatiq application, there was no need for further configuration regarding the databases.

4.11.11 Networking and security

I placed all Diplomatiq server resources in a common virtual network to be able to apply firewalls, logging, and filtering rules to the infrastructure. I created the virtual network

⁴²Bolt is Neo4j's binary communication protocol [108]. The back end application communicates with the database over Bolt protocol.

⁴³The back end service operates within the other dedicated subnet.

⁴⁴Neo4j's subnet can be reached from my personal IP address, for administrative purposes.

vnet-prod-northeurope-001 within the resource group *rg-diplomatiq-prod-001*, and set its address space to `10.0.0.0/16`.

Filtering network traffic can be achieved by security rules defined in *Network Security Groups (NSGs)*. As NSGs are bound to subnets, at first I needed to create two subnets within *vnet-prod-northeurope-001*:

- The *snet-prod-northeurope-001* subnet contains Internet-facing services: the website, and the front end and back end services. Its address space is `10.0.0.0/20`. As this subnet hosts only App Services, I delegated the whole subnet to the service *Microsoft.Web/serverFarms* for more stable networking conditions [109].
- The *snet-prod-northeurope-002* subnet contains resources that should not be available over the Internet: the database machine and the Key Vaults. Its address space is `10.0.16.0/20`.

Then I created two NSGs: *nsg-internetfacing-prod-001* and *nsg-restrictive-prod-001*. The earlier allows any inbound traffic coming from the Internet on ports 80 and 443, and allows any outbound traffic. The latter allows inbound traffic only coming from the address space of *snet-prod-northeurope-001* on ports 443 and 7687 — essentially it allows internal services to access the Key Vaults and Neo4j, but nothing else.

Figure 4.4 presents a summary about Diplomatiq's server infrastructure, displaying the main networking characteristics and rules.

4.11.12 Resource locking

In order to prevent the accidental deletion of resources, I created a resource lock on the *diplomatiq-prod-001* subscription. As long as this lock exists, the subscription and the contained resources are not allowed to be deleted.

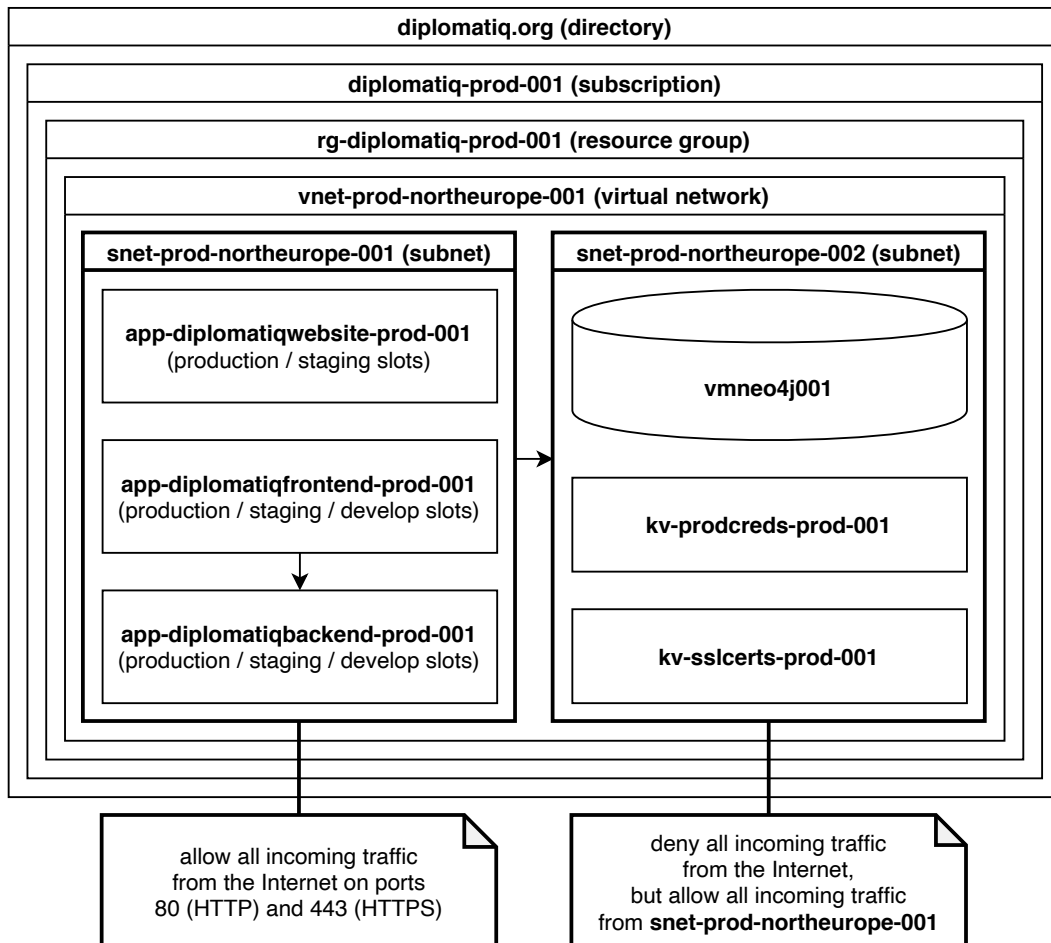


Figure 4.4 Summary of Diplomatq's server infrastructure

Chapter 5

Overview and development of supportive libraries

This chapter gives an overview about the produced supportive libraries. It unfolds the reasons of their existence, as well as their features and implementation details.

5.1 crypto-random

5.1.1 Introduction

Cryptography is very sensitive to misuse. Even if there are no obvious programming errors in a software utilizing cryptographic primitives, and even if the result it produces is the right one, the implementation can still contain catastrophic mistakes, opening several kinds of vulnerabilities to malicious attackers. Cryptographic software development needs to be approached humbly, accepting that one should not incorporate cryptography into their software without experience or expert guidance.

Secure software development often involves randomness and unpredictability. Secure encryption needs a cryptographically random, unpredictable key, and the best passwords are also the unpredictable ones. Even though most platforms offer reliable *cryptographically secure pseudo-random number generators (CSPRNGs)*, there are few ready-to-use solutions for *properly* generating strings from custom character sets, numbers from custom intervals and boolean values. This usually results in developers having little cryptographic experience implement *something which seems to yield random results*, but is totally broken from a cryptographic point of view, opening up the application to a variety of attacks.

Having worked in cryptographic software development for the last three years, I feel qualified to apply well-implemented cryptographic primitives when developing security-related software (but I do not feel qualified at all to implement those cryptographic primitives

by myself). I built the *crypto-random* TypeScript library with the goal to provide a simple, easy-to-use interface for generating cryptographically strong, *uniformly distributed* random integers from custom intervals, strings from custom character sets, and boolean values. The library is based on the Web Crypto API's CSPRNG [110], and contains no custom implementation of low-level cryptographic primitives. I use *crypto-random* extensively in the Diplomatq social network's TypeScript-based client application.

5.1.2 Features and operation

The Web Crypto API's CSPRNG can be invoked via the *window.crypto.getRandomValues()* method in JavaScript, and it is implemented by almost all browsers [111]. The method fills the provided typed array¹ with cryptographically secure random numbers. The API provides no immediate way to get random strings or booleans or other values — this is why I needed *crypto-random*. Essentially, the library maps the random numbers it acquired from the Web Crypto API's CSPRNG to other data types, following the given user constraints.

The library is able to generate the following random values:

- *bytes* of the given number,
- *integers* of the given number, within the given interval, optionally unique within the set of results,
- *strings* of the given length, consisting of characters of the given character set, with optionally unique characters,
- *boolean values*.

The library also offers a set of helper methods to create strings of the given length of various common alphabets, such as lowercase English letters, uppercase English letters, numbers, and the combination of these.

5.1.3 Discrete uniform distribution

In the context of random generation, discrete uniform distribution essentially means that the generator produces its output in a way that each of its *possible* output values gets chosen as the *actual* output value with equal probability [112]. That is, each *possible* output is equally likely to be the *actual* output.² This results that a large number of generated values follow the discrete uniform distribution. For random generators used in security-related applications, uniform distribution is a critical requirement, otherwise the generated output can be more easily predicted. I implemented *crypto-random* so that its output follows the discrete uniform distribution.

¹In JavaScript, a typed array is either an Int8Array, a Uint8Array, an Int16Array, a Uint16Array, an Int32Array, or a Uint32Array.

²For example: heads or tails with a fair coin follows the discrete uniform distribution, as both outputs has $\frac{1}{2}$ probability, meaning both outcomes are equally likely to happen.

5.1.4 Testing

The library's full functionality is tested with unit tests, resulting in 100% code coverage. The code coverage metrics are always checked to remain 100% as part of the library's continuous integration workflow. Besides basic input-output and format tests, the core generation logic is also tested whether it produces its output following a uniform distribution, based on an adaptive approach of Pearson's χ^2 test [113]. The test itself follows Fourmilab's test implementation [114]. The underlying default PRNGs are always considered to be cryptographically secure, so the actual randomness of the output is not tested.

5.2 convertibles

5.2.1 Introduction

The *convertibles* library is a TypeScript utility library to convert values between textual and binary representation. Such conversions can be useful in several scenarios, e.g. transmitting binary data in a JSON structure. The library is extensively used in Diplomatq.

5.2.2 Features

This module is built with the intention to help with the following:

- encode a source value into a better-suited serialization format,
- decode the serialized format and get the source value back.

For example, the source value can be a meaningful Unicode string value. This source value is what the application works with, this is what the business logic is built upon.

If this source value needs to be stored or transmitted in different formats, the source value can be encoded into a less complex and better manageable target value, like:

- a byte array (e.g. for storing it in binary structures),
- a Base64 string (e.g. for transmitting it as the part of a JSON payload),
- or a Base64URL string (e.g. for storing it as a filename or putting it into a URL).

Table 5.1 summarizes the source and target formats supported by *convertibles*.

Source format	Uint8Array	Base64	Base64URL	hex
Unicode string	●	●	●	○
Uint8Array	—	●	●	●

Table 5.1 Source and target formats supported by *convertibles*

5.2.3 Testing

The library's full functionality is tested with unit tests, resulting in 100% code coverage. The code coverage metrics are always checked to remain 100% as part of the library's continuous integration workflow. All conversions are tested with several test vectors. The conversion of Unicode strings are tested in all Unicode normal forms.

5.3 resily

5.3.1 Introduction

The *resily* TypeScript resilience and transient-fault-handling library allows developers to express policies such as Retry, Fallback, Circuit Breaker, Timeout, Bulkhead Isolation, and Cache. I was inspired by App-vNext/Polly [115], and decided to implement a similar toolset with a different approach, in TypeScript.

5.3.2 Features

The *resily* library offers *reactive* and *proactive* policies:

- A *reactive* policy executes the wrapped method, then reacts to the outcome (which in practice is the result of or an exception thrown by the executed method) by acting as specified in the policy itself. Examples for reactive policies include retry, fallback, circuit-breaker.
- A *proactive* policy executes the wrapped method, then acts on its own as specified in the policy itself, regardless of the outcome of the executed code. Examples for proactive policies include timeout, bulkhead isolation, cache.

5.3.3 Reactive policies

Retry policy

The retry policy claims that many faults are transient and will not occur again after a delay. It allows configuring automatic retries on specified conditions. The number of retries can be configured, up to infinity. Before retrying, certain actions can be performed by adding retry hooks. The policy can be instructed to wait a certain number of milliseconds before retrying. The duration of the wait can depend on the number of the current retry. Also, *resily* provides several predefined backoff strategies for retry policies, such as constant backoff, linear backoff, exponential backoff and jittered backoff.³

³The jittered backoff strategy uses the random integer generation logic of *crypto-random*, thus *resily* is dependent on *crypto-random*.

Fallback policy

The fallback policy claims that failures happen, and we can prepare for them. It allows configuring substitute values or automated fallback actions. The fallback chain can consist of an arbitrary number of elements. If there are no more elements on the fallback chain but the last result/exception is still reactive — meaning there are no more fallbacks when needed —, a `FallbackChainExhaustedException` is thrown. Before fallbacks, certain actions can be performed by adding fallback hooks.

Circuit breaker policy

The circuit breaker policy claims that systems faulting under heavy load can recover easier without even more load — in these cases it's better to fail fast than to keep callers on hold for a long time. If there are more consecutive faulty responses than the configured number, it breaks the circuit (blocks the executions) for a specified time period.

The circuit breaker policy has 4 states, and works as follows:

- *Closed* is the initial state. When closed, the circuit allows executions, while measuring reactive results and exceptions. All results (reactive or not) are returned and all exceptions (reactive or not) are rethrown. When encountering altogether *numberOfConsecutiveReactionsBeforeCircuitBreak* reactive results or exceptions consecutively, the circuit transitions to *Open* state, meaning the circuit is broken.
- While the circuit is in *Open* state, no action wrapped into the policy gets executed. Every call will fail fast with a *BrokenCircuitException*. The circuit remains open for the specified duration. After the duration elapses, the subsequent execution call transitions the circuit to *AttemptingClose* state.
- The *AttemptingClose* is a temporary state of an attempt to close the circuit. This state exists only between the subsequent execution call to the circuit after the break duration elapsed in *Open* state, and the actual execution of the wrapped method. The next circuit state is determined by the result or exception produced by the executed method. If the result or exception is reactive to the policy, the circuit transitions back to *Open* state for the specified circuit break duration. If the result or exception is not reactive to the policy, the circuit transitions to *Closed* state.
- The circuit can be broken manually by calling `policy.isolate()` from any state. This transitions the circuit to *Isolated* state. While the circuit is in *Isolated* state, no action wrapped into the policy gets executed. Every call will fail fast with an *IsolatedCircuitException*. The circuit remains in *Isolated* state until `policy.reset()` is called.

The number of consecutive reactions breaking the circuit can be configured, along with the duration of how long the circuit should be broken. It is also possible to subscribe state transitions with a set of hooks.

5.3.4 Proactive policies

Timeout policy

The timeout policy claims that after some time, it is unlikely the call will be successful. It ensures the caller does not have to wait more than the specified timeout. On timeout, the policy's promise is rejected with a *TimeoutException*.

Only asynchronous methods can be executed within a timeout policy, or else no timeout happens. The timeout policy is implemented with `Promise.race()`, racing the promise returned by the executed method with a promise that is rejected after the specified time elapses. If the executed method is not asynchronous (i.e. it does not have at least one point to pause its execution at), no timeout will happen even if the execution takes longer than the specified timeout duration, since there is no point in time for taking the control out from the executed method's hands to reject the timeout's promise.

The executed method is fully executed to its end (unless it throws an exception), regardless of whether a timeout has occurred or not. The timeout policy ensures that the caller does not have to wait more than the specified timeout, but it does neither cancel nor abort⁴ the execution of the method. This means that if the executed method has side effects, these side effects can occur even after the timeout happened.

Bulkhead isolation policy

The bulkhead⁵ isolation policy claims that too many concurrent calls can overload a resource. It limits the number of concurrently executed actions as specified. Method calls executed via the policy are placed into a size-limited bulkhead compartment, limiting the maximum number of concurrent executions. The size of the bulkhead compartment and the queue can be configured.

If the bulkhead compartment is full — meaning the maximum number of concurrent executions is reached —, additional calls can be queued up, ready to be executed whenever a place falls vacant in the bulkhead compartment (i.e. an execution finishes). Queuing up these calls ensures that the resource protected by the policy is always at maximum utilization, while limiting the number of concurrent actions ensures that the resource is not overloaded. The queue is a simple FIFO⁶ buffer.

⁴TypeScript/JavaScript has no generic way of canceling or aborting an executing method, either synchronous or asynchronous. The timeout policy runs arbitrary user-provided code: it cannot be assumed the code is prepared in any way (e.g. it has cancel points). The provided code could be executed in a separate worker thread so it can be aborted instantaneously by terminating the worker, but run-time compiling a worker from user-provided code is ugly and error-prone.

⁵A bulkhead is a wall within a ship which separates one compartment from another, such that damage to one compartment does not cause the whole ship to sink [115].

⁶first in, first out

When the invoked on a method, the policy's operation can be described as follows:

1. If there is an execution slot available in the bulkhead compartment, execute the method immediately.
2. Else if there is still space in the queue, enqueue the execution intent of the method — without actually executing the method —, then wait asynchronously until the method can be executed. An execution intent gets dequeued — and its corresponding method gets executed — each time an execution slot becomes available in the bulkhead compartment.
3. Else throw a *BulkheadCompartmentRejectedException*.

From the caller's point of view, this is all transparent: the promise returned by the policy is:

- either eventually resolved with the return value of the wrapped method,
- or eventually rejected with an exception thrown by the wrapped method,
- or immediately rejected with a *BulkheadCompartmentRejectedException*.

Cache policy

The cache policy claims that within a given time frame, a system may respond with the same answer, thus there is no need to actually perform the query. It retrieves the response from a local cache within the time frame, after storing it on the first query.

The cache policy is implemented as a simple in-memory cache. Its validity can be configured, and it can be invalidated manually as well. It works as follows:

- For the first time (and every further time the cache is invalid), the cache policy executes the wrapped method, and caches its result.
- For subsequent execution calls, the cached result is returned and the wrapped method is not executed — as long as the cache remains valid.
- The cache is valid as long as it is not expired (see time to live settings below) or manually invalidated.

5.3.5 Testing

The library's full functionality is tested with unit tests, resulting in 100% code coverage. The code coverage metrics are always checked to remain 100% as part of the library's continuous integration workflow.

5.4 project-config

Bootstrapping new projects within an organization where projects have many common elements can be tedious. After creating new new project structure, the necessary files need

to be copied from another repository, and the project-specific content needs to be replaced to match the new project. To avoid this manual work, I created the *project-config* tool. This tool essentially bootstraps a new project with the provided name and description, meaning it copies all necessary configuration and description files into the new project, and replaces placeholders with the project name and description. It is implemented with a Bourne Again Shell (bash) script.

5.5 eslint-config-tslib and eslint-config-angular

ESLint is a configurable linter tool for maintaining the code quality of JavaScript and TypeScript repositories. It identifies and reports certain patterns. I have not found any such ESLint rulesets on the Internet, which would have been strict enough for my taste, so I decided to create two own rulesets: one for TypeScript libraries, and one for Angular. Both rulesets are used across Diplomatq's TypeScript repositories.

Chapter 6

Overview and development of the Diplomatiq application

This chapter demonstrates the Diplomatiq social network application¹, its specification, client-server architecture, features and development methods, and implementation details.

6.1 Introduction

6.1.1 Personal goals and outline of this chapter

I want Diplomatiq to be a long-term project. After finishing the university, I want to establish and lead a company responsible for the operation and further development of the social network, extending it with features supporting real-world diplomacy as well. I want to make changes for the better, providing a suitable platform for diplomacy, one of the main driving forces of our globalized world. In this chapter I outline Diplomatiq's vision to the future, as well as its target audience, specification, implemented features so far, and development details.

6.1.2 Vision

With the continuous advancement of today's computing power and data processing capabilities, and mass data sources like social networks, we become capable to detect new, important connections between seemingly unrelated actions and events of the world. Several studies has shown such event detection approaches regarding various topics, such as smart societies [116], critical public infrastructure [117], and security threats in computer systems [118]. Analyzing data in a given domains even opens the possibility of predicting domain-specific events [119].

¹The application is available on <https://app.diplomatiq.org>.

Diplomatiq's long-term vision is to become an embedded, integrated system in diplomacy, providing tools for diplomats, and capable of detecting events of international relations on the global scale, based on its own data. In order to get there, it starts off as a domain-specific social network initially for prospective diplomats of the younger generation, Model United Nations participants. Then it gradually expands to real-world diplomacy as more and more career diplomats are invited into the application through MUN conferences. This creates the social base of its ecosystem. New features driving diplomatic adoption can be added progressively, further expanding the network.² Once it is used by enough people in the simulated and the real diplomatic domains, the system will already hold data of great value. Analyzing this data, along with processing data added into the system in real time, would open possibilities to detect and predict global events.

6.1.3 Target audience

Diplomatiq's initial target audience is the participants of Model United Nations conferences. As previously assumed, the number of students who participate in and/or organize MUN conferences is probably in the order of millions, worldwide. However, supporting MUN conferences is also an entry step into real diplomacy. This means that in the future, Diplomatiq's target audience will expand to career diplomats, government officials, and diplomatic administrators as well.

6.2 Long-term feature goals against competitors

6.2.1 Introduction

As previously detailed, Diplomatiq has a direct competitor regarding MUN conference organization: the MyMUN application provides several kinds of features for MUN conference organizers and participants, and has an already established social base with its more than 100,000 members [17]. Although the long-term vision of Diplomatiq is much broader than the current scope provided by MyMUN, in the short run, Diplomatiq should be able to cope with MyMUN.

This section introduces planned features for a similar, but better social experience, than the one provided by MyMUN. Most of the features detailed below were not planned in detail, and were not implemented as part of the current version of Diplomatiq. However, I want to include them into this thesis to summarize the future scopes of Diplomatiq as well as the current ones. The actually implemented features of the framework is detailed in Section 6.3.

²It is important to emphasize here that right now it does not matter, what kind of features will be implemented for diplomats. The first goal of Diplomatiq is to build a heavy social base with many important diplomats. When Diplomatiq gets there, it can decide what to do with the gathered data and social foundation.

6.2.2 Features for entities organizing conferences, associations, and clubs

Registering and administering MUN conferences

In Diplomatiq, the basic organizational unit should be a *conference*. Such conferences should be able to have multiple sessions, which is an actual manifestation of the conference as a few-day-long event. A conference should be able to have a flexible hierarchy of organizational roles in order to cope with the relatively high fluctuation of the organizational body of a conference due to students graduating and leaving the host institution.

Grouping conferences into an MUN association

An *MUN association* is a group of conferences owned, organized and hosted by the same legal entity. The concept of an MUN association can mostly be applied when the same educational institution organizes multiple conferences: in this case, the institution can be regarded as an MUN association. The system should be able to handle all organizational aspects of creating and maintaining an MUN association, along with the necessary administration of its leadership and ownership. Similarly to conferences, the system should provide a flexible approach for handling organizational hierarchies.

Hosting MUN clubs

MUN clubs are extracurricular activities usually taking place weekly or more often, hosted by an MUN association (educational institution) usually within its own premises, for its own members (students). An MUN club provides an opportunity for junior MUN participants to develop their debating and other skills in order not to take part in a conference completely unprepared. Diplomatiq should be able to handle the administration and documentation of MUN clubs, encourage association members to take part as often as they can, and record students' MUN club performance in order to combine them into their conference recommendations.

Administering the organization of a conference session

Diplomatiq should cover all administrative requirements of organizing an MUN conference. This includes customizable participant application with assignable roles, and handling all financial aspects of the application procedure. Organizers should be able to set the participants' fee regarding their roles, with additional discounts. Fully custom payment classes should also be added for covering additional financial needs. The system should provide a way to refund participant fees in case of cancellations. It should also support creating flexible and complete financial reports covering the requirements of accounting. Regarding finances, the software should offer ultimate transparency, and should record every transaction to be audited later.

Delegates and chairs should be able to choose their country and committee preferences up to an allowed number configured by organizers. The system should provide an automated way of creating preliminary assignments of county-committee pairs and conference participants — based on organizational priority metrics and applicant preferences — even before a delegation is accepted or rejected, and allow the organizers to incorporate these assignment previews into their decisions regarding the delegation's or the delegate's acceptance to the conference. The final assignments should also be automatically created, with optional organizational modifications.

Cross-selling additional services

Diplomatiq should establish business relationships with local businesses, such as hotels, restaurants, and entertainment providers in order to provide delegates additional offers or entertainment packages for conferences, with optional discounts. The system should also be able to handle contractual relationships between such local businesses and the entity organizing the conference, and incorporate this into its financial system when calculating the consummative participation fee for applicants. In essence, Diplomatiq should allow the cross-sales of its own business partners' services, and also the services of conferences' business partners. These services can later include travel insurance, flight tickets, and car rentals, similarly to MyMUN. In the long run, Diplomatiq should provide a concierge-like service for applicant delegations, with personalized travel, flight, and entertainment offers, taking over all paperwork from applicants, so they can concentrate on preparing to the conference instead of the administration.

Conference awards

It is common to award best delegates and best chairpersons of a committee or of the whole conference. For delegates, conference awards can be an important milestone for further conferences or even in their professional lives, therefore Diplomatiq should be able to store these awards. Also, the system should be able to officially certify that a delegate received such awards on a conference.

Additionally, Diplomatiq should provide a way for ordering additional hard copies of awards. This should be fully automated, initiated by the user from their personal profile within Diplomatiq. The platform's payment system could issue an invoice to directly the user.

Personalized conference items

All MUN conferences involve personalized items, such as badges and placards. After the application and assignment procedure finished, the system should be able to produce all such personalized items as one, singular, printable artefact, sorted by the organizers' needs, in order to free organizers from manual batch work.

Ordering conference items from Diplomatiq

Diplomatiq should be able to carry out the printing, packaging and shipping operations of several types of conference items on demand. This includes badges and placards in printed, optionally laminated form, as well as pass holders, and conference awards. This way, conference organizers can outsource all — virtual and paper-based — administrative paperwork to Diplomatiq.

Marketing operations

The system should provide a sophisticated marketing toolset for conference organizers for advertising their conference, with optionally targeting participants based on their previous MUN history. It should offer multiple types of promotions, with multiple pricing tiers.

Logging and auditing

All actions recorded on Diplomatiq should be retrievable in a non-deniable, auditable manner. The platform should offer sophisticated search features for fetching the history of applicants, or organizational actions of a conference. Nevertheless, the principle of least privilege and access should be applied in every case: organizers should only access data they need to conduct their work, within only their conference.

Providing catering, supplies, and infrastructure services for conferences

In the long run, Diplomatiq should provide on-site services to conferences. These services can include catering services for lunch and for soirées, providing supplies for conferences such as mineral water and snacks, and even infrastructure services. Usually educational institutions do not even have wireless internet, let alone microphones, computers, and printers for every committee.

Recording entire MUN conference sessions, producing commercials

Diplomatiq should provide a data recording service for conferences, as part of its on-site services. Conferences produce lots of data: committee proceedings, resolutions, video recordings of ceremonies... On demand, Diplomatiq should be able to send a team to the conference with the tools to record the entire conference including committee work with the contribution and speeches of every delegates in every committees, ceremonies, soirées. Ultimately the “recorded package” should contain video and sound recordings, the exact record of all proceedings in committees, and all related paperwork, and serve as the record of the entire conference session. This recorded package then can be uploaded to Diplomatiq, and retrieved later for additional work on demand. The video and sound recordings of the conference can also be used to produce commercials, if the organizers order any such services, for later advertisement.

6.2.3 Features for entities participating in conferences

Recommending conferences to delegates and chairpersons

Diplomatiq should recommend conferences to delegates and chairpersons, based on their experience, and preferences they set in their profile settings. Recommendations can include sponsored and paid recommendations, as well as additional advertising of conferences. The platform should measure the efficiency of its organic and sponsored recommendations, and improve its targeting algorithms upon them.

Public participant profile

Participants often benefit from having an extensive MUN experience. Diplomatiq should provide audited profiles for delegates certifying their conference participations and listing their awards, allowing conference organizers to see a CV-like summary of applicant delegates. Profiles should also contain organizational experience, if there is any.

Membership programs and discounts

Diplomatiq should provide a discount system for top delegates, delegations and conferences. Discounts can be participant fee discounts, or discounts for attached services, either way encouraging entities to strive for better performance.

6.2.4 Features for non-MUN entities

Similarly to Twitter and LinkedIn, Diplomatiq could also offer data licensing model for external parties interested in globally available, audited and verified data on Model United Nations. Later, simultaneously with the framework's expansion to real-world diplomacy, as the volume of data to be licensed also grows, more kinds of interested parties can be involved. In this model, Diplomatiq should offer anonymized data only.

6.3 Specification and implemented features

6.3.1 Introduction

This section lists Diplomatiq's implemented features extensively, and can be regarded as a specification on how the system available on <https://app.diplomatiq.org> should behave under different conditions. The section presents the features along the user journey of a user who registers the application, applies to a conference, then decides to organize a conference as well. Supplementary user flows, such as a user forgetting a password, are presented separately. At this point, I would like to point out that the goal of this thesis was to implement the framework on strong infrastructural and architectural foundations, but with *a primitive, minimal feature set*.

6.3.2 Registration

The user visiting Diplomatiq arrives to a login screen, where they either logs in with their existing credentials or decides to register a new account. Diplomatiq offers registration and authentication based on email address and password only.³

For registering a new account, the user needs to enter their valid, existing email address, their first and last name, and their password. After initiating the signup, they receive a success message about receiving an email with a link containing a long and secure email verification code. The user needs to click on that link and thus verify their email address before logging in to Diplomatiq, otherwise the system will not let them in.

If there is already a registered in Diplomatiq with the email address provided at registration, the application is obviously not able to register another user with that same email address. However in this case, the user is still notified that the registration was successful, and a verification email was sent to the provided email address — even though the email of course will not arrive. This is one of the many security measures to prevent malicious parties from obtaining the email addresses of registered users. Diplomatiq does not leak user emails in any way.

6.3.3 First and second factor of authentication

After the user verified themselves, they arrive to the login page of the application with their email address prefilled.⁴ After the user entered their password and clicked on *Log in*, they receive an email with a two-factor authentication code.⁵ After the user provided the valid authentication code, they are logged in to Diplomatiq, and is taken to the Dashboard.

6.3.4 The Dashboard

The Dashboard provides a summary about conferences the user takes part in or organizes. The two kinds of conferences are shown in two separate tab. In participant role, the user can see data about the conference (name, codename, dates, and venue address) and their committee seat (represented country in a committee) they applied to. The user can cancel their application in this view. If the user has not applied to any conferences, they are encouraged to apply to one in the empty state view of the participated conferences' list.

³This was a technological decision justified by my security requirements. I will detail this decision's background in Chapter 7.

⁴The lack of automatic login in this case is intentional. On the one hand, password-based cryptographic operations involved by the login process cannot be substituted with a simple link. On the other hand, I have put serious effort into implementing a zero-knowledge authentication protocol which makes it cryptographically impossible that the server impersonates the client. Sending a link to the user which provides them a fully authenticated session would contradict this.

⁵In the future, I will incorporate a dedicated multi-factor authentication service, possibly Authy (<https://authy.com>).

In organizer role, the user's organized conferences are listed with a name, codename, dates and venue address. Also, the number of applied participants is shown in a progress bar view, in the ratio of all announced committee seats of the conference. The organizer user can see the country matrix, which is a list of users occupying committee seats they applied to. The user can also cancel the conference from this view. If the user has not organized any conferences, they are encouraged to organize one in the empty state view of the organized conferences' list.

In the Dashboard's both views' right side, the user is encouraged to apply to a conference, or to organize one. This panel is always visible, regardless of the user current conferences, either in participant or in organizer role.

6.3.5 The navigation bar

The top navigation bar of the application hosts two elements. The application's logo is always visible, and if the user is logged in, navigates to the Dashboard, else it navigates to the login screen. If the user is logged in, the navigation bar displays a user menu as well. The user menu is a drop-down menu displaying the user's email address and having 4 navigation action buttons: *Dashboard* (navigates to the Dashboard), *Explore* (navigates to the view where users can apply to conferences), *Settings* (navigates to the user settings), and *Log out* (logs out the user and navigates to the login page).

6.3.6 Applying to a conference

Users can apply to announced conferences on the Explore page, which can be reached either via the user or from the Dashboard. On the Explore page, the conferences which the user can apply to are presented as a list, with their name and codename, dates, and venue address. Such conferences are conferences which are in the future, and the user has not applied to them yet. With the only button in a conference's row, a user can initiate the application process. This opens a modal showing the vacant committee seats of the conference, which the user can directly select and apply to, or abort the application process by closing the modal. After clicking on a seat's *Apply* button, the application process is complete. The conference organizers are notified of the application via email.

6.3.7 Organizing a conference

Users can initiate organizing a conference from the Dashboard. On the opening modal, users should provide conference data, such as the conference's name and codename, its starting date and ending date, the address of the conference venue, and committees with seats. The dates can be provided as string in *YYYY-MM-DD* form, or by an associated date chooser element. Dates are validated, so the start date can only be one day after the current

date, and the ending date must not precede the starting date.⁶ For the venue address, users should provide the country, city, the address itself, and a postal code. The country is implemented as a search-assisted drop-down from a fixed set of countries according to the ISO 3166 standard.

Committees and their seats can be added on a per-committee basis, on a separate modal, providing the committee name and its codename. Adding a country creates a new form element, where the country's name can be provided. Once all countries in a committee are provided, the committee can be saved, and another committee can be added. After all committees are added, the conference can be saved, which also immediately opens the application to the conference.

6.3.8 Cancelling a conference application

The user can cancel their application to a conference on the Dashboard, on the participant role tab. Cancelling a conference application requires password-level authentication for assuring the identity and intention of the user: they need to provide their password to be able to cancel their application. This action sends an email to the conference organizer about the cancellation.

6.3.9 Cancelling a conference

The user can cancel one of their organized conferences on the Dashboard, on the organizer role tab. Cancelling a whole conference needs two-factor-level authentication for assuring the identity and intention of the user: they need to provide their password and enter the two-factor authentication code they received in email, in order to be able to cancel a whole conference. This action sends an email to all conference applicants about the cancellation.

6.3.10 Password change

Users can change their password on the Settings page, which can be reached from the user menu. Changing password needs password-level authentication: the user needs to provide their current password in order to be able to change it.

6.3.11 Account deletion

In case a user does not want to use Diplomatiq any more, they can delete their account, if they have no such conferences either as organizer or as participant, which ends in the future. Account deletion can be initiated on the Settings page, and requires two-factor-level authentication: they need to provide their password and enter the two-factor authentication code they received in email in order to delete their account.

⁶As there are one-day conferences, the end date can be the same as the start date.

6.3.12 Password reset

If a user forgot their password, they can reset it with an email-based password-reset flow, initiated from a modal on the login page. When the user provides their email address in the modal, and submits the form, they receive an email with a long and secure password reset link. Opening the link takes them to the reset password view, where they can supply a new password. After resetting a password, they are able to log in with the new password.

6.4 Development and implementation details of the back end

6.4.1 Introduction

Having already used Spring Boot in hobby projects, I had an explicit concept on how I want to build and layer the back end application. This section introduces the steps of setting up the Diplomatq back end application — exposing an API over HTTPS, communicating with a database — from scratch until a publicly available application on the internet. Throughout the development, I strived to keep the quality, configurability and reliability of the application as high as possible. Security-related features and implementation details — such as authentication, authorization, additional security features for communicating over a HTTPS API, encrypted database entries, and additional web application security elements — of the application will be detailed in Chapter 7.

6.4.2 Setting up an empty Spring Boot application

Since Spring Boot heavily advertises that since it needs minimal configuration due to its opinionated view of the platform and third-party libraries, it makes easy to start off with the framework. I wanted to make use of this, therefore I initialized an empty Spring Boot project with the help of Spring Initializr, which bootstraps projects with needing minimal configuration.⁷ I set up a Java-based project with Maven, using the latest released version of the framework. The project's group name has been *org.diplomatq*, its artifact name has been *diplomatq-backend*, and thus its full package name has been *org.diplomatq.diplomatq-backend*. I chose JAR-packaging due to reasons detailed later, and set the required Java SDK version to Java 11.

6.4.3 API concepts

Introduction

Since Diplomatq's API is primarily for internal use⁸, I was able to make design decisions without worrying about adoption aspects. Since Diplomatq's own client is the only one

⁷<https://start.spring.io>

⁸In the future, Diplomatq will provide a public API as well, but on different design principles.

who consumes this API, I was able to build non-conventional architectural and security solutions⁹ into the API without concern for its usability for external developers. Nevertheless, this approach ultimately lead to an API with transparent, well-documented, easy-to-follow conventions, fitting well into Diplomatq's security framework.

Flat, RPC-like construction

REST¹⁰ APIs work well those kinds of semantics when the server exposes *resources* over an API, and clients use those resources in certain ways. However, this semantics carries several inconvenient abstractions of and assumptions about the data model and the operation of the underlying application, which can make the API inflexible. During the development of Diplomatq's back end API, I intentionally avoided the REST semantics in favor of a flat, RPC-like¹¹ interface, exposing *methods* instead of *resources*. Every method is exposed via its own HTTP endpoint with a one-depth fixed path, relative to the base URI of the API, e.g. <https://api.diplomatq.org/password-authentication-init-v1>.

JSON or binary communication

I decided to use JSON as the primary format of communication between the client and the server due to its transparency and ubiquitous support. However, if the content sent or received is represented in a purely binary format — e.g. an image file or a cryptographic key —, it can be easier to send it in its binary format rather than serializing then deserializing, therefore these resources can be transferred as binary.

In Diplomatq's API, I do not use content negotiation between the client and the server.¹² Every exposed endpoint has a fixed semantics for receiving and sending data.

GET, POST, and PUT methods only

According to well-defined semantics below, I decided that the API endpoints should be callable either with the *GET*, the *POST* or the *PUT* HTTP method.

- *GET* methods are allowed to have query parameters, but no request body.¹³ Semantically, they should be used for getting data, and must never modify data.
- *POST* methods are allowed to have a request body, but no query parameters. Semantically, they should be used for creating or modifying remote objects.

⁹The implemented security measures will be detailed in Chapter 7.

¹⁰Representational State Transfer

¹¹Remote Procedure Call

¹²Content negotiation allows the server to expose data in different representations on the same URI, by the demands of the client.

¹³A GET request can not have a request body anyway.

- *PUT* methods are allowed to have both a request body and query parameters. Semantically, they should be used for uploading large amounts of binary data, with attached metadata as query parameters.

Endpoint-level versioning

The versioning of the API is implemented on a per-endpoint basis, in order to provide maximum flexibility. Every exposed endpoint's path ends with the endpoint's actual version — such as */login-v1* and */login-v2* —, meaning they are essentially different endpoints, and can have completely different semantics and differing behaviour. Since the API is meticulously documented¹⁴ — and also only used by Diplomatq clients — this should not cause any problems, instead it allows to gradually modify the API instead of needing to publish a new version of the entire API with every breaking change.

With endpoint-level versioning, endpoint deprecation can also be easier. After a new version of an endpoint was published — without client change necessary —, a new client using the new endpoint can be released, then the old version of the endpoint can be unpublished from the API. This workflow allows to think of API changes as step-by-step modifications instead of a new version of the entire interface. This results in the lack of versioning the interface as a whole.

6.4.4 Implementation details of the API

I used Spring Boot's built-in capabilities to implement the API itself. Since the API methods do not carry any resource-like semantics, they have been compartmentalized into controllers along authentication schemes and levels¹⁵. This means every method requiring the same authentication scheme and authentication level is placed in the same controller.

Endpoints are automatically exposed by Spring Boot, according to the controller methods' annotations. These annotations define each endpoints' path (e.g. */login-v1*), HTTP request method (e.g. *POST*), and its consumed and produced value (e.g. *application/json*).

Every input and output parameter of every endpoint is implemented as a separate class, unless the parameter is a simple value. The request and response objects are named unequivocally, based on the endpoint's name and version, i.e. the request object of the */login-v1* endpoint is *LoginV1Request*, and its response object is *LoginV1Response*. Marshalling and unmarshalling¹⁶ is automatically performed by the Jackson library¹⁷, which is automatically configured by Spring Boot.

¹⁴API documentation will be detailed later.

¹⁵The concept of authentication schemes and authentication levels is detailed in Chapter 7.

¹⁶Marshalling (serialization) means transforming an object's memory representation to a serialization format. Unmarshalling (deserialization) means transforming serialized object into memory representation.

¹⁷<https://github.com/FasterXML/jackson>

6.4.5 Documenting the API

I introduced and configured the *springdoc-openapi* library¹⁸ for the automated documentation of the API according to the 3rd version of the OpenAPI Specification [120]. For human processing, the documentation is publicly available.¹⁹

The OpenAPI documentation also has a format processable by machines, meaning I was able to automatically generate a fully functioning API client for the front end project, based on the exposed API documentation for machines.²⁰

6.4.6 Domain data model

The domain data model of the application consists of the following entities:

- `AuthenticationSession`
- `AuthenticationSessionMultiFactorElevationRequest`
- `Committee`
- `CommitteeSeat`
- `Conference`
- `Session`
- `SessionMultiFactorElevationRequest`
- `UserAuthentication`
- `UserAuthenticationResetRequest`
- `UserDevice`
- `UserIdentity`
- `UserTemporarySRPData`

Figure 6.1 displays a summary about Diplomatiq's entire data model with the entities' labels, labeled entity connections, and the connections' cardinalities.

For mapping objects and object relations to graphs, Diplomatiq uses the Neo4j-OGM library, built into the Spring Data Neo4j (SDN) project. SDN provides a fully-fledged data access layer handling database sessions and transactions transparently. Classes annotated with the `@NodeEntity` annotation are automatically regarded as domain entities, thus incorporated into SDN's scope, and can be handled via *repositories* providing convenient, automatically configured CRUD²¹ methods. For more convenient entity modeling, I defined a hierarchy of abstract classes for entities. `AbstractIdentifiableNodeEntity` generates 32-character-long random alphanumeric identifiers; `AbstractCreationRecordedNodeEntity` and `AbstractExpiringNodeEntity` handles storing creation and expiration uniformly.

¹⁸<https://springdoc.org>

¹⁹<https://api.diplomatiq.org/openapi-documentation/v3/ui>

²⁰<https://api.diplomatiq.org/openapi-documentation/v3/api>

²¹Create, Read, Update, Delete

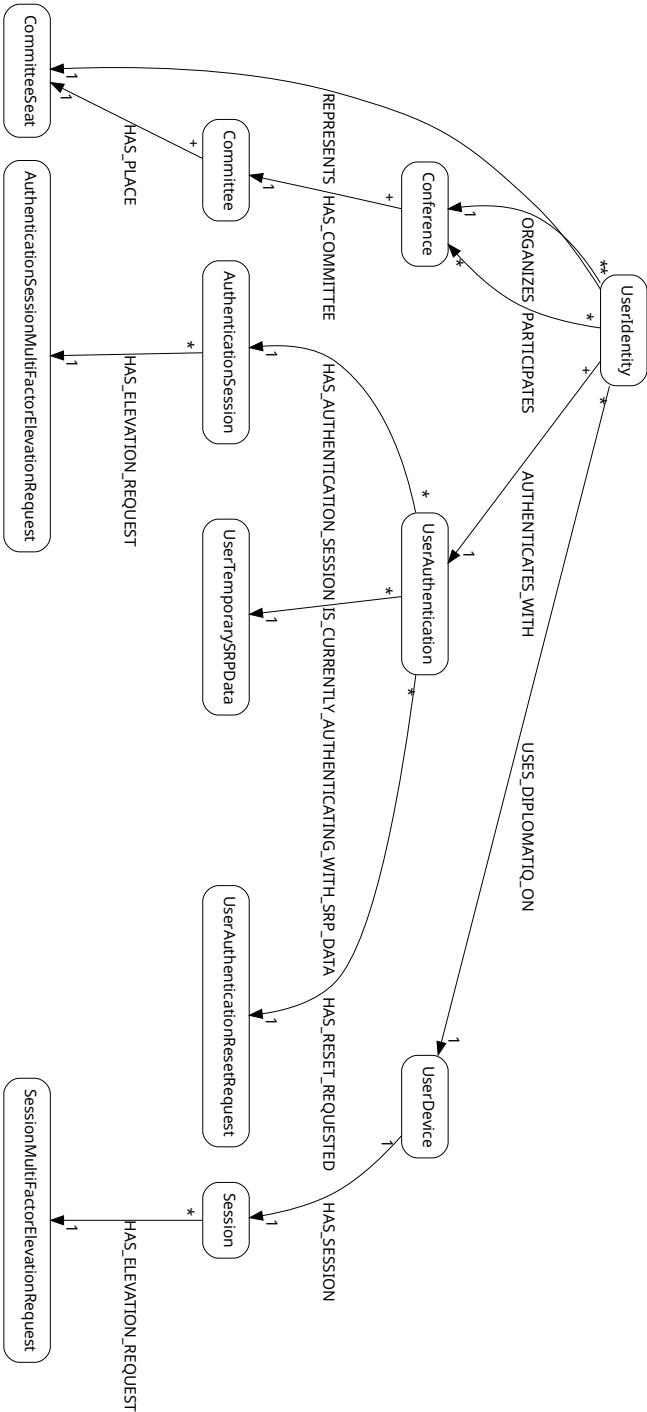


Figure 6.1 The data model of the Diplomatq application

6.4.7 Layering the application

Maintaining a transparent implementation hierarchy is key to efficient software maintenance. Therefore I defined the following application structure from the API layer to the database layer²²:

- *Filters* forward or reject an incoming HTTP request, and play a key part in Diplomatq's security architecture, which is detailed in Chapter 7.
- *Controllers* handle HTTP requests and return HTTP responses. Thanks to the automated marshalling, the controller layer can be regarded responsible for converting JSON input to a POJO²³, and forward it to the service layer (and vice versa).
- *Services* are the highest-level actors of the business logic layer (BLL). They act as transaction boundaries: if an exception is thrown within the service, the related database transaction is rolled back.
- *Engines* contain lower-level logic for performing specific computations.
- *Utilities* and *helpers* represent the lowest levels of the BLL, and abstract frequently used functionalities, such as creating an entity in a uniform way.
- *Repositories* are lowest-level actors of the application hierarchy, representing the data access layer. They are responsible for reading from and writing to the database in a managed way.

6.4.8 Error handling

In the following, the term *business error* denotes all errors that the client's and the server's joint business logic is built upon, essentially meaning all those errors which the client needs to be distinguish from each other. In Diplomatq, errors which are lower level than business errors do not need to be exposed — and should not need to be exposed²⁴ — by the server, thus they are mapped to a suitable business error.

Business errors are handled in a semantically uniform way. One of my design principles was that the API either returns a success or an error, and I mapped this principle to the use two HTTP error codes only, where 200 means success, and 400 means error. In every case, the error response is an object consisting of a distinctive error code string, and optional retry information. Every business error has its own error code, thus the client can distinguish errors based on this error code. In case of an error which does not carry any specific semantics or its origin is unknown, the error code is `InternalServerError`. Error codes are documented in the exposed API documentation.

²²This structure excludes everything which are handled by the application server, or the Spring Boot framework or any related libraries, such as reading and writing HTTP requests and responses, and actually interacting with the database.

²³Plain Old Java Object

²⁴Too detailed errors exposed by the API can reveal sensitive security properties of a system to an attacker.

6.4.9 Configuration

Most of the back end application's configuration has been externalized into configuration files or environment variables. Spring Boot supports externalized configuration well [43]. Database credentials and API keys are always stored in environment variables only, or they get a default, “dummy” value if their environment variables are not found. This makes development easier, since the local development environment can be set up with those dummy values, instead of needing to set up environment variables every time.

6.4.10 Packaging and deployment

Spring Boot provides two way of packaging an application: either as a traditional WAR²⁵ file, or a stand-alone, self-contained JAR²⁶ file. The WAR file format is more suitable for deploying the software into a monolith application server hosting multiple web applications. The JAR file format is self-contained, meaning it encapsulates an embedded servlet container²⁷, and runs the actual Spring Boot application on this embedded application servlet container.

The back end application is packaged as a stand-alone JAR file, as this approach fits more into Diplomatiq's infrastructure: there is no need to deploy and maintain a fully-fledged Java application server. Instead, only a self-contained JAR file needs to be uploaded onto the web server, and the web server needs to be configured to start the application defined by the JAR, and forward all incoming requests to the application. The deployment of the back end application is performed as part of the project's continuous delivery flow. Whenever the master branch of the repository is updated, the automated workflow defined in GitHub Actions eventually creates a distribution artefact. The final artefact is uploaded onto the *staging* slot of the project's App Service in Microsoft Azure. Then the *staging* slot's webserver instances are warmed up in order to receive production workload, meaning that the new version of the application is started on both instances. As soon as both instances reply to a continuous polling with a non-error response — meaning they are fully initialized and ready to work —, the *staging* slot is automatically swapped into production, resuting in no downtime during deployments.

6.5 Development and implementation details of the front end

6.5.1 Introduction

Having used the Angular framework throughout my career so far, I had a defined concept on how I want to build Diplomatiq's client application. Although the implemented application

²⁵web application resource or web application archive

²⁶Java archive

²⁷The default servlet container for Spring Boot is Tomcat, but Jetty or Undertow are available as well [43].

is relatively small and provides few features, I wanted to build the software on a high-quality foundation, so later it can be easily extended on demand.

6.5.2 Creating an empty Angular application

I bootstrapped an empty Angular application with Angular CLI.²⁸ The project has been named *diplomatiq-frontend*. As part of the guided bootstrapping process, I added several modules to the application, such as routing, and set the project's stylesheet format to SCSS. Angular CLI created the folder structure of the project, and installed all necessary dependencies via NPM.

6.5.3 Structuring the application

Components

A routed Angular web application loads *components* on configured paths: such components encapsulate a view displayed in the web browser and a logic of how the component works and reacts to user interactions [121]. Components can be nested into each other, allowing developers to create complex application layouts. I tried to keep the component layer as thin as possible, in order to abstract all actual application logic into *services*.

Modules

Angular applications are modular. *Modules* are a set of components and configuration, which belong to one coherent application unit, a workflow, or closely related set of capabilities [121]. Modularization can be beneficial in terms of compartmentalized application loading or application bundle size. However, the front end application of Diplomatiq consists of only one *root* module, making no use of those advantages.

Services

Services are application classes containing logic. Besides components, most of Diplomatiq's front end application code is in services, which are responsible for data manipulation, and transitively — through a dedicated API service — for calling the back end API as well. Services are usually injected into components or other services with dependency injection.

6.5.4 API access and error handling

I implemented a dedicated service for performing API calls. This service exposes the endpoints of the back end's API, which is compartmentalized by authentication schemes and levels. As the API client is automatically generated using the *openapi-generator*²⁹ tool,

²⁸<https://cli.angular.io>

²⁹<https://github.com/OpenAPITools/openapi-generator>

this compartmentalization is visible in the client as well, meaning the API service actually exposes additional “services” instead of the endpoints themselves. The exposed services are the partial APIs compartments — each with endpoints with the same authentication scheme and level —, and they are the ones providing the actual endpoint stubs.

The *openapi-generator* tool can be configured to generate code for several libraries and frameworks, including Angular. Although, the server stubs generated for Angular did not offer any way for attaching dynamically computed headers to the server request, as it did not provide any way to add before-request or after-request hooks to the generated code. This is a requirement for Diplomatq, because the request’s cryptographic signature — required by the back end API as a header — is not a static, configurable value, but needs to be computed from the request dynamically. Therefore I configured the generator to generate code utilizing the Fetch API [122], a modern, native browser API for performing asynchronous, cancellable network requests.

API errors are handled in an after-request hook of the generated API client. If the API response has a different HTTP response code than 200, a JavaScript error is thrown with its error message set to be the received error code. If the received error response has no error code, or is not an error which conforms to the Diplomatq error scheme mentioned earlier — meaning the back end application is incapable even to produce a business error for some reason —, the `InternalServerError` error code is set.

6.5.5 Packaging and deployment

Angular provides sophisticated build tools, capable of generating all source files of the application into one bundle with *webpack*³⁰, one of today’s most popular JavaScript bundler. Therefore, building the distribution artefact essentially means that only an Angular-specific command needs to be invoked, which builds the application bundle, and copies it with additional assets, such as images and icons, into a folder, usually called *dist*. The contents of the *dist* folder contains the entire application, and can be uploaded to a web server.

Similarly to the back end project, the deployment of the front end application is performed as part of the project’s continuous delivery flow. Whenever the master branch of the repository is updated, the automated workflow defined in GitHub Actions eventually creates a distribution artefact. The final artefact is uploaded onto the staging slot of the project’s App Service in Microsoft Azure. After the files were placed into the webserver’s public folder, and ready to be served to users, the staging slot is automatically swapped into production, resulting in no downtime during deployments.³¹

³⁰<https://webpack.js.org>

³¹In case of static web applications, there is no need to warm up the instances, as there is no back end application to start on the web server. However, Azure copies static files into its webserver’s public folder by taking the webserver offline, and I wanted to avoid the downtime caused by that.

Chapter 7

Security of the Diplomatiq application

7.1 Introduction

7.1.1 Motivation

Web applications in general are notoriously insecure. According to an article by the prominent security firm Positive Technologies, statistics of 2019 showed that hackers were able to hack users in 9 out of 10 web applications, exploiting vulnerabilities in the applications themselves [123]. The same article claims that breaches of sensitive data were a threat in 68% of web applications, with the most breachable data being personal information and access credentials.

Having worked in cryptographic software development, I share the opinion that there is no such thing as too much software security. As on the one hand, Diplomatiq will store sensitive personal information from the beginning of its operation, and on the other hand, future diplomatic applications of the software will require strong security guarantees, I decided to put serious efforts to securing the Diplomatiq application. In this chapter, I present how I designed and implemented an application security framework addressing authentication, authorization, and data protection, providing cryptographic assurances regarding access control and the confidentiality and integrity of sensitive data.

7.1.2 Scope and approach

The implemented application security measures address all application layers across the front end and the back end of Diplomatiq. The front end needs to provide secure local data storage in the browser, as well as a variety of other security measures so the user can not be tricked into surrendering their password or other sensitive data. On the back end, the whole application needs to be secured, from the API's authentication and authorization mechanisms to the encrypted database records.

For securing the front end and the back end, I started off from my professional experience of securing a web application, and I also consulted the Open Web Application Security Project's web application security testing guide [124]. I implemented zero-knowledge user authentication using a suitable protocol, and I built a signing and authentication mechanism for HTTPS requests based on a symmetric cryptographic key hierarchy, created upon the resulting shared key of the user authentication protocol. For authorizing sensitive API operations, I introduce the concept of session assurance level in order to make sure that the logged user is indeed themselves, and not an attacker who hijacked a user's session.

7.2 Implemented cryptographic structures

7.2.1 DiplomatiqAEAD

DiplomatiqAEAD is an *Authenticated Encryption with Associated Data* structure, providing binary serialization and deserialization capabilities. It essentially consists of an encrypted secret value and a public value attached to it, while the whole structure is authenticated, meaning it can not be modified without detection.

For the authenticated encryption, I chose today's prevalent Advanced Encryption Standard (AES) algorithm with 256-byte-long keys, in the more and more popular Galois/Counter Mode (GCM). AES-GCM provides strong confidentiality and ensures integrity of both the ciphertext and the additional authenticated data.

Although it does not contain self-implemented low-level cryptographic elements, DiplomatiqAEAD can be regarded as a cryptographic primitive within Diplomatiq applications, since it is a basic cryptographic building block of the system's authentication and database encryption infrastructure. The structure is used for transmitting encrypted and authenticated data across the front end and the back end application. This requires that the structure can be serialized in a self-contained way — meaning it must contain all necessary metadata needed for decryption as well —, and also that the two implementations on two different platforms are compatible with each other.

Similarly to other block cipher modes of operation, GCM also needs an initialization vector for a properly randomized output. According to the recommendation of the National Institute of Standards and Technology — a prevalent actor in cryptographic standards as well —, the initialization vector should be unique and 12 bytes long for efficient calculations [125]. The initialization vector is also needed for decryption, so it needs to be included in the serialized packet. The output of AES-GCM is the ciphertext and an authentication tag, which is essentially a digest of the ciphertext, the initialization vector, and the additional authenticated data, verifying the integrity of all three. Most cryptography APIs append the authentication tag to the end of the ciphertext. As the authentication tag is needed for performing decryption, it needs to be included in the serialized packet.

For encoding the initialization vector, the ciphertext, the additional authenticated data and the authentication tag as a binary structure, I created a lightweight binary serialization scheme for DiplomatqAEAD.¹ The scheme consists of a header and a body. The body consists of the initialization vector, the additional authentication data, the ciphertext, and the authentication tag, in this order. Since each part of the body can essentially be of arbitrary length, the header contains each body part's length encoded on a fixed length: the length of the initialization vector on 1 byte, the length of the ciphertext on 4 bytes, the length of the additional authenticated data on 4 bytes, and the length of the authentication tag on 1 byte. All lengths are encoded in big-endian byte order, in unsigned magnitude representation. Encoding the length of the ciphertext and the additional authenticated data on 4 bytes introduces a planned constraint that DiplomatqAEAD can be used only for encrypting and authenticating maximum 4 gigabytes of data at once. Figure 7.1 presents the binary serialization format of DiplomatqAEAD.

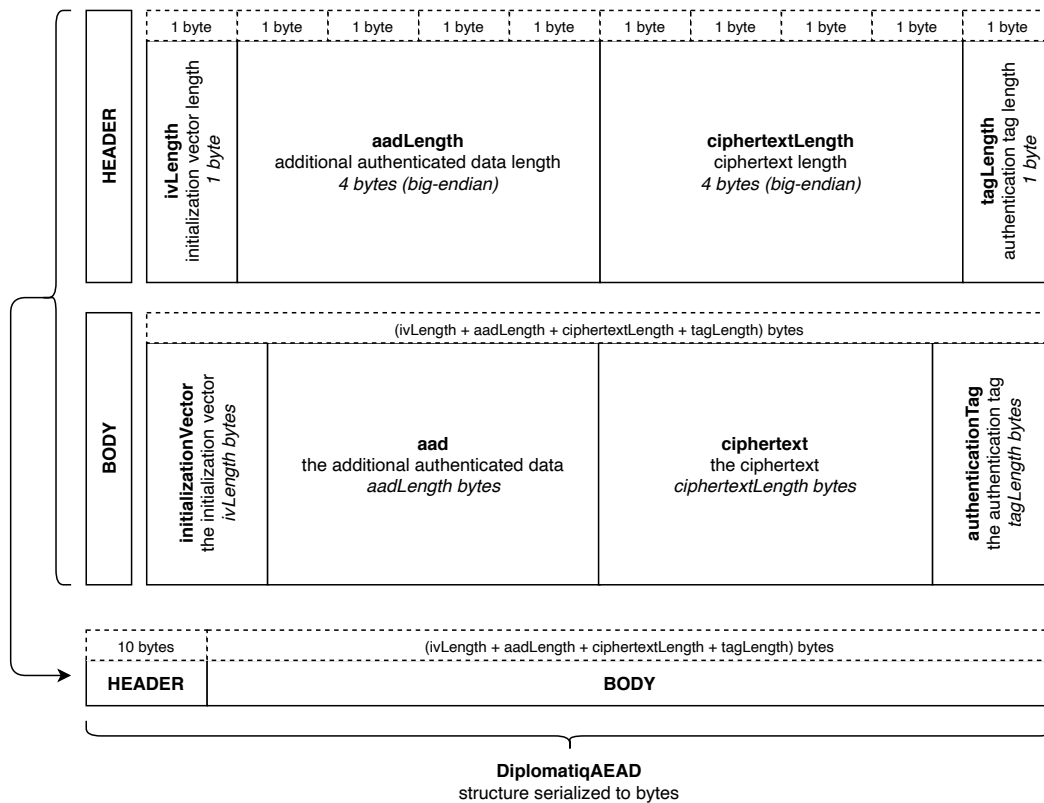


Figure 7.1 The structure of DiplomatqAEAD serialized to bytes

¹Although there are established standards like ASN.1, I decided that using them for this simple purpose only would cause more pain than gain.

7.3 Applied web platform security measures

7.3.1 Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) attacks are a type of code injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to an end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it [126]. Although in most modern browsers, a strong Content Security Policy ruleset disabling inline JavaScript can mitigate XSS, in older browsers the X-XSS-Protection header can be used to instruct the browser to prevent rendering the page if an XSS attack is detected. Diplomatq includes the X-XSS-Protection header into its responses. The Angular framework also implements strong XSS mitigation measures: as long as their built-in filtering is used to display dynamic data in the browser, the application is protected. Diplomatq uses Angular's built-in filtering.

7.3.2 Content Security Policy (CSP)

The Content Security Policy browser mechanism allows websites and web applications to specify what kind of resources (e.g. images, stylesheet files, script files, font files) can be loaded within the context of the application, and what kind of domains can the application connect to via network calls. This can protect against XSS attacks, and can also protect against performing unauthorized API calls. Diplomatq applies a very strict Content Security Policy ruleset, which restricts everything by default, and only allows loading images, script files, stylesheet files, and font files from the application's very own domain name. Inline scripts are not allowed, meaning the application is safe against XSS attacks. Diplomatq's policy allows connecting the front end application to *api.diplomatq.org* only.

7.3.3 Cross-Origin Resource Sharing (CORS)

By default, web browsers enforce the same-origin policy, meaning a web application running on a given domain can not initiate network requests to another domain. The Cross-Origin Resource Sharing mechanism instructs the browser to allow an application running on a certain origin to connect to a domain different than its origin. CORS settings are communicated to browsers via additional HTTP headers, specifying the origins which are allowed to connect to the domain which issued the headers. The Diplomatq back end application issues CORS headers, which authorizes the application running under *app.diplomatq.org* to connect to the back end. No other application is authorized other than Diplomatq's front end application.

7.3.4 Feature Policy

The Feature Policy experimental mechanism enables to deny the usage of browser features for a web application, such as the browser's geolocation API, camera API, vibrate API, full screen API, etc. Feature Policy settings are communicated to the browser via a Feature-Policy header. Their Feature Policies being the strictest, Diplomatiq's applications are not authorized to use such browser APIs at all.

7.3.5 Referrer Policy

The Referrer Policy mechanism can mitigate threats associated with the `Referer`² header. The `Referer` header contains the URL of the page which the user visited before navigated to the current page. Referrer Policy settings are communicated to the browser via the Referrer-Policy header. Their Referrer Policies being the strictest, Diplomatiq's applications omit the `Referer` header entirely.

7.3.6 Preventing click-jacking

Click-jacking is a type of attacks where a page is embedded into another page via an `<frame>` or `<iframe>` element, and the user's clicks are hijacked to perform additional interactions on other web sites. Click-jacking can be prevented by instructing browsers not to allow embedding an application. Although Diplomatiq's restrictive Content Security Policy protects against this in modern browsers, in older browsers the `X-Frame-Options` header should be set to `DENY`. Diplomatiq's applications set this header accordingly as well.

7.4 Authentication and authorization

7.4.1 Introduction, motivation, and terminology

Broken and insecure authentication schemes are a constant problem of web applications [123]. On the list summarizing the top then security vulnerabilities of the Open Web Application Security Project, broken authentication is located in the second place [127].

Besides an authentication scheme being technically correct — it uses suitable protocols and hides sensitive information by techniques like encryption —, it also needs to provide additional guarantees on ensuring a user's identity. Password-only authentication is a contributing factor in most authentication attacks [123]. Times are changing: yesterday's recommendations on authentication systems based on strong, frequently changing passwords are replaced with more modern approaches based on multiple factors of authentication. NIST explicitly advises to organizations that instead of focusing on password policies, incorporate multi-factor authentication solutions into their systems [128].

²The `Referer` header is a misspelling of word "referrer".

I wanted to get authentication and authorization right in Diplomatiq's systems. For certain sensitive interactions, I wanted to provide additional guarantees on the user's identity — beyond the standard approach that “authenticate the user *only once* based on their password and an additional second factor, then later in any point in time believe without verification that the user sitting in front of the computer is still the same” — to avoid stealing session credentials. I also wanted to be able to further encrypt sensitive between the client and the server. Even though all communications are already done encrypted over Transport Layer Security, I still believe there is no such thing as too much security — as long as it does not mean a worse user experience.

In the following, I use the term authentication in two different contexts. *User authentication* means that the user cryptographically proves to the server, that they are in possession of a valid password, which corresponds to the given email address, then performs a second step of email-based authentication. In essence, *user authentication* means a user logs in to Diplomatiq. The process of user authentication results in a set of *login credentials*, which the client stores locally, encrypted. *Request authentication* means that the client cryptographically proves to the server that they are in possession of valid login credentials, without sending the actual credentials over the network. In essence, request authentication happens on a per-API-request basis, and essentially means that when the logged in user's Diplomatiq client application performs a request against Diplomatiq's API, the API checks if the client provided a legitimate cryptographic proof about the possession of the necessary credentials. The goal of user and request authentication is to get a strong proof about the user's identity.

In the following, the term authorization means that an already authenticated user is checked by the server whether they have the necessary level of access and level of *session assurance* to perform a given API operation. Basically this is the same as the original meaning of authorization, but Diplomatiq's authorization scheme incorporates the elapsed *time* into the authorization. A session's assurance level can be elevated with re-authenticating, meaning an elevated session assurance level provides a stronger proof about the user's identity sitting in front of the computer at a given point in time.³ Similarly to request authentication, authorization also happens on a per-request basis, and the request is rejected if the user does not have the required level of access.

7.4.2 Why not OpenID Connect or OAuth 2.0?

There are several already established, mature authentication and authorization standards on the market, like OpenID Connect⁴ [129] for authentication, and OAuth 2.0⁵ [130] for

³After the specified time, the assurance level expires, and a user needs to re-authenticate again for sensitive operations.

⁴<https://openid.net/connect>

⁵<https://oauth.net/2>

authorization. Having implemented several applications upon both methodologies, I have some, but not exhaustive experience on using these standards. After further inspections, I concluded that neither solutions would satisfy all my below specified requirements.

Not implementing OAuth 2.0 introduces a restriction of federated authentication, meaning users will not be able to authenticate themselves to Diplomatiq with e.g. their social media accounts. This question does not get addressed in this thesis.

7.4.3 Requirements

In the following, I summarize the requirements I formulated regarding Diplomatiq's authentication and authorization framework, and I also describe the threats they mitigate.

Zero-knowledge password proof

User authentication should be implemented upon a *zero-knowledge password proof* scheme. The client should be able to verify to the server, that the user knows the password, without the password or its any derivative form ever leaving the client or being transmitted over the network. The server should store neither the user's password, nor any password-equivalent data. This prevents attackers from stealing user passwords either over the network, or by breaking into the back end application's database.

Mandatory multi-factor authentication

Beyond password, a mandatory second factor of authentication should be used for user authentication. This prevents attackers to perform password-based authentication and impersonate the user.

No personal data for request authentication

No personally identifiable data should be used for authenticating a network request. This implies that network traffic between the client and the server should not contain personally identifiable information about the authenticated user by default, unless of course the request's actual payload contains such information to be stored on the server. This is a further layer of protection over Transport Layer Security.

Shared symmetric key between the client and the server for encryption and signing

User authentication should result in a long-term shared symmetric key between the client and the server for performing further encryption of very sensitive data and credentials transmitted over the network. This prevents attackers to obtain plaintext credentials if they somehow manage to steal and decrypt an encrypted network request over TLS.

Signed network requests

Authenticated network requests should be signed, and the signature should incorporate all characteristics of the request, including its timestamp. The request should be rejected by the server if the signature is invalid, or if the timestamp is outside a given interval to the current time. This prevents attackers to replay a network request⁶ if they somehow manage to steal one over TLS.

Time-based session assurance levels

The scheme should provide a way to ascertain the user identity before performing sensitive operations. That is, the scheme should incorporate assurance levels for sessions in the meaning of a higher assurance level means a higher certainty regarding the identity of the user sitting in front of the computer at a given point in time. Sensitive API operations should require a higher session assurance level. This prevents attackers to perform sensitive API operations impersonating the user, who left their computer unlocked.

Immediate logout

The scheme should not rely solely on expiring entities. Request credentials should be authenticated and authorized for every request, and this should always incorporate checking the validity of the provided credentials. This implies that access tokens should be stored on the server, along with their expiry. This prevents attackers from stealing abandoned access credentials, which have not yet expired.

7.4.4 Protecting endpoints

Diplomatiq endpoints can be split into two parts:

- *Unauthenticated* (i.e. “not logged in”) endpoints are publicly available for every user, and they do not need to be able to identify the user. They do not require requests to be signed either.
- *Authenticated* (i.e. “logged in”) endpoints are available after performing user authentication only. These endpoints require the user to be identifiable, and requests must be signed in order to prevent tampering and replay attacks.⁷

Endpoints are protected using filters implemented in Spring Boot. Filters are ordered, and executed sequentially. They can either forward, modify or even reject the request with an error. As described later, both authenticated and unauthenticated are protected by filters.

⁶A replay attack is performed by replaying or delaying an otherwise valid network data transmission. Replay attacks can be carried out by eavesdropping network traffic, then re-transmitting it later.

⁷Request signing serves as another layer of protection over TLS.

7.4.5 User authentication

Proving the knowledge of the password without ever revealing the password

For performing zero-knowledge password-based authentication, I decided to use the Secure Remote Password (SRP) protocol's latest, 6a version [131]. SRP is an augmented password-authenticated key agreement (PAKE) protocol, which specifies a set of cryptographic methods for performing password-based authentication between two parties over an insecure channel, without sending the password or its any derivative over the channel. Similarly to other PAKE protocols, an attacker obtaining the network traffic⁸ of the authentication would not be able to perform an exhaustive search for the password, based solely on the obtained network data. Since the server does not store the password, even if an attacker steals the server's database, they would not be able to impersonate the user, unless they perform an exhaustive key search⁹ for the password first, based on the stolen data.

The authentication results in a shared, symmetric session key between the parties. The protocol provides strong cryptographic proof to both parties that they successfully authenticated against each other, i.e. the server can cryptographically verify the password.

Logging in

The user authentication flow consists of the following steps:

1. The client initiates SRP's password authentication process by providing their email address to the server, and receives a response with the server's cryptographic challenge value.
2. The client performs cryptographic calculations on the challenge value, mixing the password's one-way hash¹⁰ into the calculations, eventually resulting in a value called a *password proof*. The password proof does not have a direct relation to the plaintext password itself, but it is based on the server challenge value and the password, and allows the server to verify that the client is indeed in possession of the valid password. With this step, the client has completed the password authentication, and the client and the server has a shared cryptographic key, which I named *authentication session key*, and the client is in possession of an authentication session.
3. To perform the login operation, which results in a *client device* authorized to acquire a *session*, the user needs to complete multi-factor authentication. This involves receiving an email with a code expiring in 5 minutes, and providing the code to the

⁸Since Diplomatiq uses TLS, it is unlikely that an attacker would be able to eavesdrop client-server communication.

⁹i.e. brute-forcing

¹⁰For hashing, Diplomatiq uses the *scrypt* password-based key derivation function to slow down attackers, who want to perform an exhaustive search for the password [132].

- server. Performing multi-factor authentication elevates the user's authentication session from password-elevated level to multi-factor elevated level.
4. With a multi-factor elevated authentication session, the user is authorized to call the login operation. This issues a *client device* with a device identifier, and returns a *device key* and a long-term *session token*. The device identifier, device key and session token are both encrypted values, wrapped into a DiplomatiqAEAD structure. The device key is encrypted with the authentication session key, and the device identifier and the session token is encrypted with the device key. To acquire the device identifier and the session token, the client first decrypts the device key with the authentication session key, then decrypts the device identifier and the session token. The device identifier is required to identify, which device of which user calls the API, and the session token is a long-term secret credential authorizing the issued device to acquire a session after completing the login flow.
 5. With a logged-in device, the client device is authorized to acquire a session, by providing a session token to the server. The session token is a sensitive credential, therefore it is sent encrypted with the device key.
 6. After acquiring a session, the client can perform calls against the API, providing valid request authentication with the session credentials.

Figure 7.2 displays a summary of the above user authentication scheme. The main security characteristics of the scheme is provided by the cryptographic key hierarchy built upon the authentication session key, which was actually never transmitted over the network. Instead, it was calculated independently on the server and on the client, as a result of the authentication procedure described by the SRP protocol. Essentially all further credentials and cryptographic keys are (transitively) encrypted with this authentication session key, which is impossible to be eavesdropped by an attacker, as it does not leave machines.

Securely storing login session credentials locally

After performing the user authentication, the client needs to persist the credentials of the issued device locally in a secure way. For this, I introduce a device container, which is an encrypted container for storing sensitive credentials in the client browser's local storage. The device container is essentially a DiplomatiqAEAD structure, encrypted with the *device container key*. The device container is never persisted in unencrypted form.

For automatically logging in the user in a given web browser by data stored in a device container as a result of a previous login, the device container key needs to be queried from the server on a public endpoint.¹¹ This key can be regarded as a not-so-sensitive data, since even if an attacker obtains a device container key, they would still need the actual device container, which is stored only in the browser's local storage.

¹¹At this point, the client does not access its credentials, as they are in the encrypted device container.

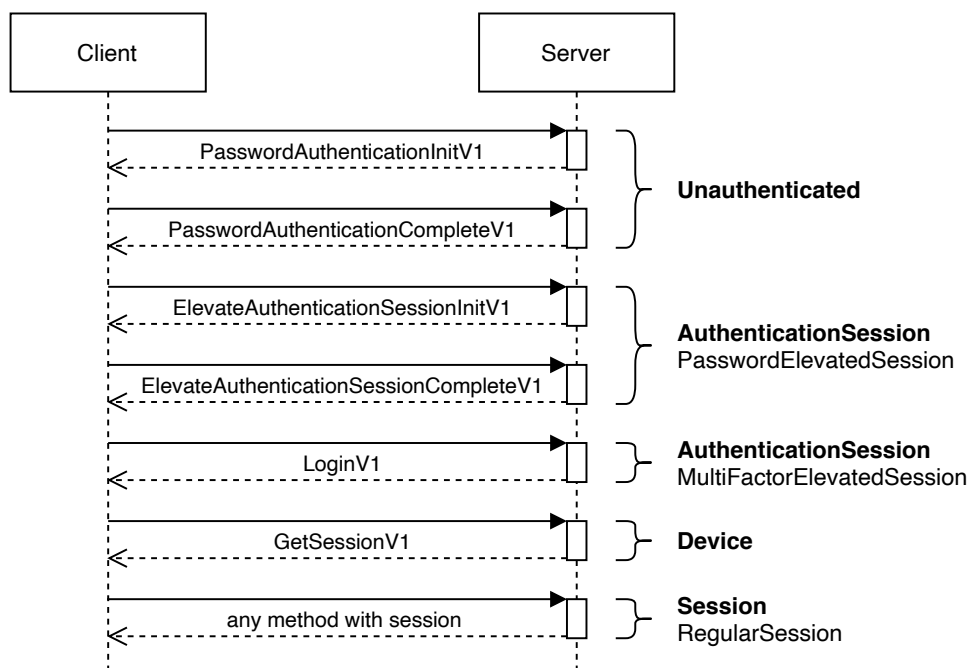


Figure 7.2 The login flow

7.4.6 Request authentication

Overview

After the user has authenticated themselves by logging in into Diplomatiq, each API request initiated by the user is also authenticated and authorized one-by-one. In this section I present different authentication schemes authenticating with different credentials over the user's authentication lifecycle, and I introduce the concept of session assurance levels.

Authentication schemes

From the point of a user entering their password to the first successful “logged-in” API call, there are several steps. These steps were introduced previously, and now I present the underlying authentication schemes I implemented. Schemes are essentially the definitions of how the request should be authenticated, and how the request's signature needs to be checked. All schemes are versioned in order to be changed on demand.

The *Unauthenticated* “authentication scheme” does not require the user to be authenticated, as its name suggests. It does not require signed requests, but it requires that the client includes the client identifier header, and also requires the client clock will be in sync with the server's clock. The client identifier and the clock synchronization is detailed later.

The *AuthenticationSessionSignatureV1* scheme is used during the password and multi-factor authentication flow of the login process. The authentication session is identified by the authentication session id, and the request is signed with the authentication session key the client and the server mutually agreed on.

The *DeviceSignatureV1* scheme is used after the user performed the password and multi-factor authentication, and logged in, meaning performed a call to the login endpoint, which issued a client device to them. As previously mentioned, this long-term device has a device key, and a session token issued to them. Requests performed with the this scheme is signed with the device key, and identified by the attached device id as a header. Actually, this scheme is used for only two purposes: getting a session with the session token and logging out.

The *SessionSignatureV1* scheme is used after the user acquired a session. This scheme is used for every regular API call, and its requests are signed with the device key, and identified by the device key and the session key attached as headers.

Figure 7.3 displays a detailed summary about authentication session schemes.

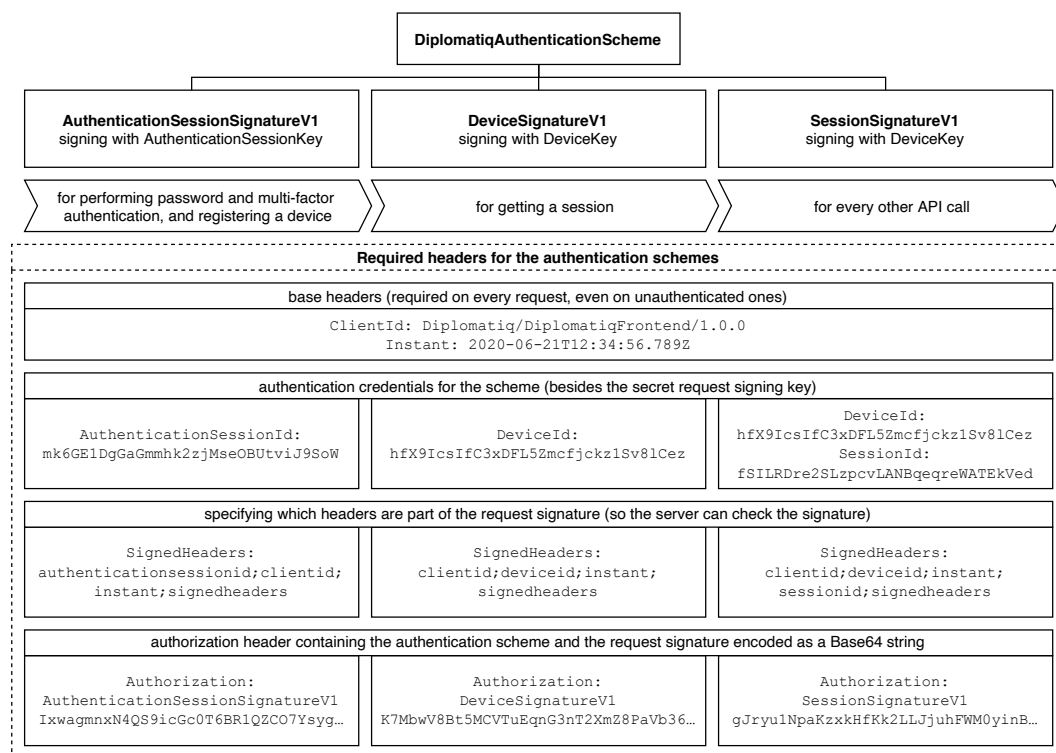


Figure 7.3 A summary of Diplomatiq's authentication schemes

Session assurance levels

There are certain very sensitive API methods and operations, which can require to re-authenticate the user. Such methods implemented in Diplomatq are password change or cancelling a conference application, which require the user's current password. More sensitive methods, such as cancelling a whole conference or deleting a user account require both the user's password and performing multi-factor authentication. These safeguards are implemented to make sure that the user really is who she says she is, and not an attacker with malicious intents. For handling these kinds of safeguards in a uniform way instead of ad-hoc re-authentication solutions, I introduce the concept of *session assurance levels*. A higher session assurance level provides a higher assurance about the current user's identity, as a result of a higher level of re-authentication.

In the extensible, hierarchical system I defined the following assurance levels:

- *RegularSession* is the lowest assurance level. It assures that a user is logged in, and can be renewed without user interaction following its 1-hour expiry.
- *PasswordElevatedSession* is the medium assurance level. It is valid for 15 minutes from point the user provided a valid password, thus it essentially proves that the user provided correct password in the last 15 minutes.
- *MultiFactorElevatedSession* is the highest assurance level. It is valid for 5 minutes from the point the user completed a multi-factor authentication flow. *MultiFactorElevatedSession* assurance level can only be acquired with a *PasswordElevatedSession* assurance level. This assurance level therefore proves that in the last 15 minutes, the user provided correct password, and in the last 5 minutes, they completed a multi-factor authentication flow through their email address, and Diplomatq can be pretty sure that the user is who she says she is.

Figure 7.4 displays a summary of the above assurance levels. Sequence diagrams presenting the process of a session elevation can be found in the Appendix.

An authentication session level is bound to a set of API methods implemented in separate controllers via Spring's controller proxy authorization. This means that if I want to create a method with a password-level session assurance, I just need to insert the method's definition in the controller defining password-level methods. As I already implemented the underlying infrastructure, there is no further configuration needed.

The session assurance levels are stored and handled on the server. The client does not even know about its credentials' current assurance level, it just tries to call the API with its current credentials, and retries after requesting password or multi-factor authentication whenever the server throws a *SessionElevationRequired* error. Whenever a request arrives to the server, it checks if the targeted controller's required assurance level is met by the provided authentication credentials, and throws the previously mentioned error, if not.

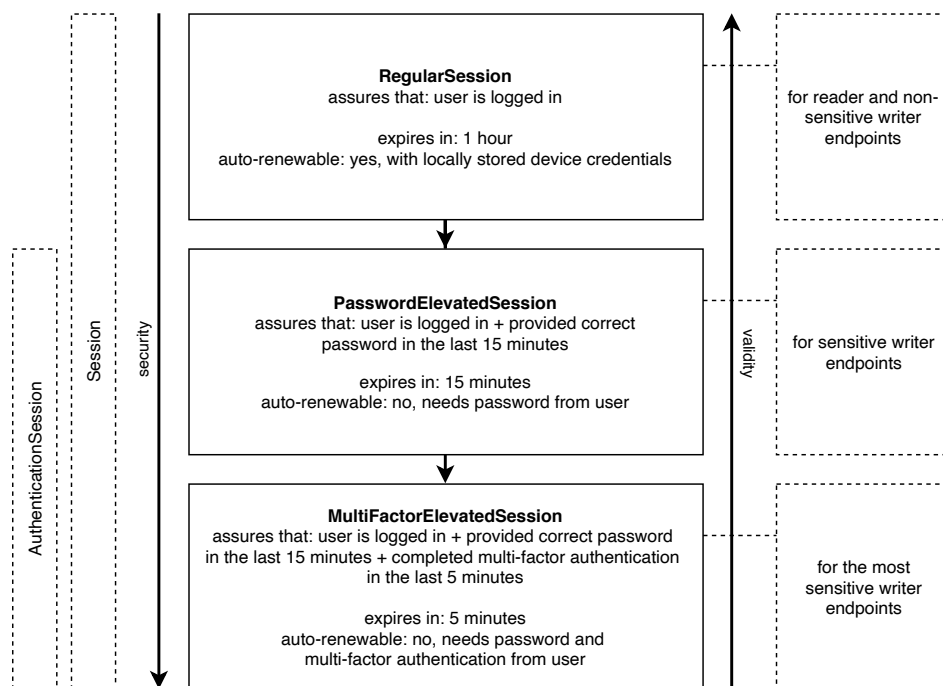


Figure 7.4 Diplomatq's session assurance levels

7.4.7 Client version identifier

For logging purposes, every request against Diplomatq's API must contain a `ClientId` header, defining which Diplomatq client's which version initiated the request to the API. In the future, when Diplomatq has clients for desktop computers and smartphones, this header will also identify the platform. The current value of the `ClientId` header is *Diplomatq/DiplomatqFrontend/X.Y.Z*, where X, Y and Z are the client's version numbers according to the semantic versioning scheme. Requests lacking the `ClientId` header are rejected.

7.4.8 Client clock discrepancy

It is unfortunate if the client's clock differs from the server's clock, because the client can display invalid data. Differing clocks can also break along the safeguards against replay attacks, since a client signing a request with wrong clock setting will not be able to create a valid signature. Therefore, every request against Diplomatq's API must contain an `Instant` header, which contains the client's current instant¹² in the ISO 8601 [133] simplified string format, including date and time.¹³

¹²Instant is a universal, instantaneous point on the time-line in the UTC time zone.

¹³Such as: 2020-05-31T23:59:59Z

7.4.9 Signing requests

In order to make cryptographically sure that requests originate from an authorized client, all authenticated requests against Diplomatiq's API must be signed with a key, which is determined by the endpoint's required authentication scheme. This allows the server to retrieve the client's signing key (such as a device key) by the attached request metadata (such as a device id provided in a request header), then verify that the attached signature was indeed created with the key.

On the one hand, this solution provides an additional layer of protection against request tampering on top of TLS, and on the other hand, since the request should contain its timestamp in the `Instant` header, it also prevents attackers to perform replay attacks after somehow obtaining a plaintext request.

To be an effective security measure, a request signature must incorporate every possible characteristics of the request. Following the request signature scheme of Amazon Web Services¹⁴ [134], I created a signature scheme which works as follows:

1. The client creates the *CanonicalRequest* string, creates its cryptographic digest with SHA-256, then encodes the digest bytes with Base64 encoding, resulting in *RequestSha256Base64*.
2. The client prefixes *RequestSha256Base64* with the authentication scheme used for authenticating the request, resulting in *StringToSign*.
3. The client creates the cryptographic signature of *StringToSign* with using the HMAC-SHA-256 keyed-hash message authentication code algorithm, using the authentication scheme's corresponding key as the algorithm's key, then encodes the signature bytes with Base64 encoding, resulting in *RequestSignatureBase64*.
4. The client sets the *Authorization* header of the request to *RequestSignatureBase64* prefixed by the authentication scheme.

The *CanonicalRequest* is a newline-separated string containing the request's HTTP method, its relative path the the server's base URI, its final query string, the *CanonicalHeaders*, and the *PayloadSha256Base64*.

The *CanonicalHeaders* is also a newline-separated string, containing all request headers in `key:value` format, where the key is converted to its lowercase invariant. As the server needs to know, which headers were incorporated into the signature¹⁵, an additional *Signed-Headers* header is also attached to the request containing the the signed headers' lowercase key separated by semicolons. The *SignedHeaders* header also needs to be included into *CanonicalHeaders*.

¹⁴<https://aws.amazon.com>

¹⁵A web browser application does access *all* HTTP headers of a sent request, therefore the signature algorithm only includes a subset of the headers.

The *PayloadSha256Base64* is the cryptographic digest of the request's final payload, created with the SHA-256 algorithm. If the request has no payload, the *PayloadSha256Base64* still needs to be included.

Figure 7.5 presents a summary about creating a signature for Diplomatq's API.

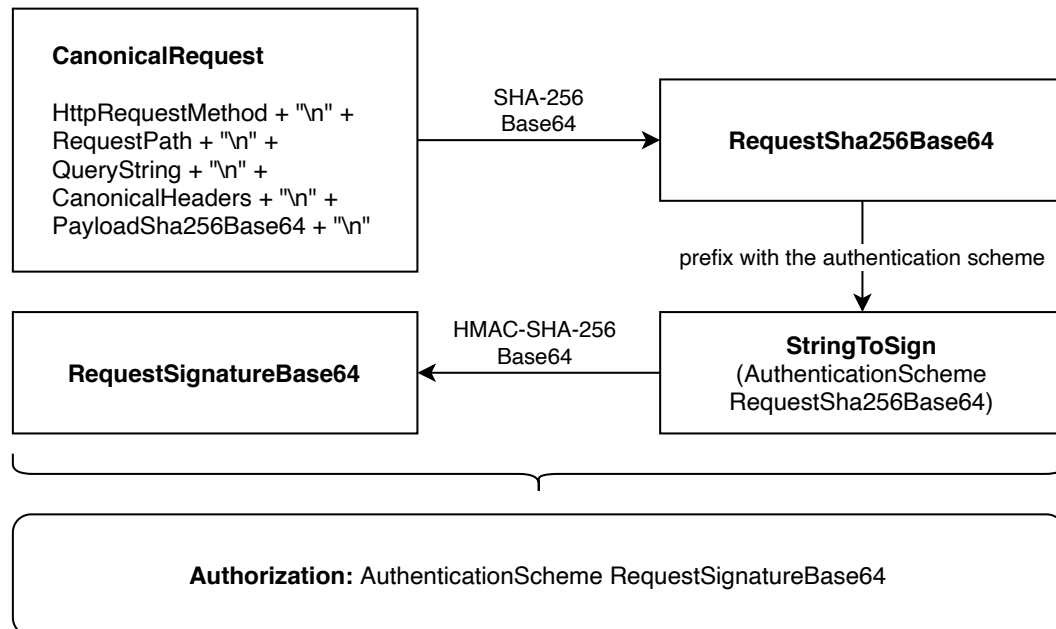


Figure 7.5 The process of signing a request

7.4.10 Implementing request authentication with Spring filters

Requests arriving to the back end application are authenticated with the filtering capabilities provided by Spring [43]. The back end application is implemented as a fully stateless application relying only on persisted database data instead of in-memory storage. Thus Spring's built-in session handling capabilities are not used.¹⁶

In Spring, filters are executed sequentially, with all filters being able to modify, forward, or reject the incoming HTTP request. Filters can be inserted into several parts of the *filter chain*, but I inserted Diplomatq's request authentication filters to the beginning of the *security filter chain*, implying that the server should properly authenticate the request before doing anything else. Diplomatq's request authentication filters do not modify the incoming request: they either forward it to the next filter, or reject it by returning a specific API error, usually *BadRequest*, *Unauthorized* or *SessionElevationRequired*.

¹⁶This also implies that Diplomatq does not use any cookie-based authentication mechanisms.

I implemented the following filters, listed in the order of invocation:

1. The *base headers checker filter* checks whether a non-empty `ClientId` and a non-empty `Instant` header was provided with the request. If not, rejects the request with *BadRequest* error code.
2. The *Clock discrepancy filter* checks whether the UTC instant provided in the `Instant` header is in sync with the server time. If not, rejects the request with *ClockDiscrepancy* error code. The discrepancy tolerance is 1 minute.
3. The *signature verification filter* verifies the request's signature. It parses the provided `Authorization` header to get the authentication scheme and the signature in Base64 encoding, then it creates the signature of the request by the received request data based on the authentication scheme. The signing key is retrieved from the database by the provided device id or authentication session id as a header. The filter creates the request according to the scheme detailed in Section 7.4.9, then compares the provided request signature with the calculated one.¹⁷ If the two do not match, the filter rejects the request with *Unauthorized* error code.
4. After the above formal checks, the *authentication filter* is responsible for actually authenticating the request by the provided authentication session scheme, and the provided credentials. The filter parses the `Authorization` header to get the authentication scheme, then it looks for the authentication scheme's corresponding credentials among the headers. If the credentials are found, and valid, the filter pronounces the request as *authenticated* by setting the user as the authenticated principal in Spring's security context¹⁸, along with the applied authentication scheme.

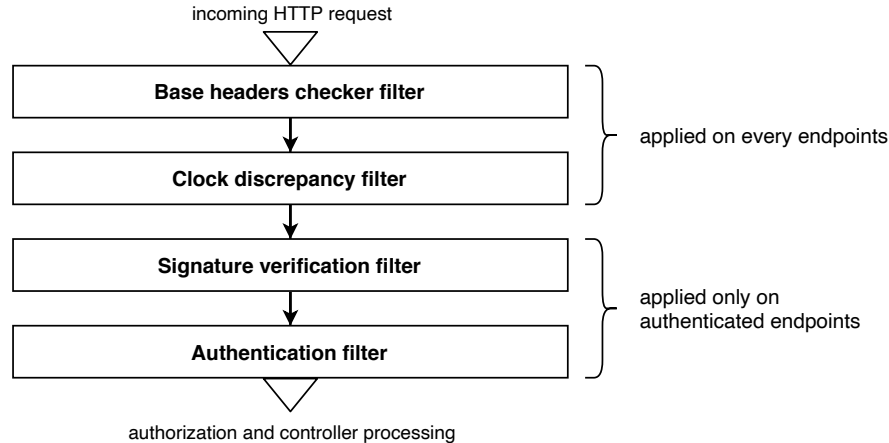


Figure 7.6 A summary of the back end application's request filters

¹⁷The comparison happens with a constant-time algorithm to avoid time-based side-channel attacks.

¹⁸The security context is configured to have a per-request lifecycle: it is flushed after the request was served.

7.4.11 Implementing controller authorization

The filters' job is to *authenticate* the request — meaning they should verify that the request comes from a registered user, who has the necessary credentials to perform a properly signed request against the API —, and store this authentication information — including the user's identity and the used authentication scheme — in Spring's security context. The user and the request can be authenticated without context-dependent information: the filters and the authentication processes do not need to know, what are the access level requirements of the actual target controller. For *authorizing* the request, the system needs to receive authentication data and contextual data as well, since different controllers, methods, resources can demand different authorization levels.

The above essentially means that the *authentication* filter put an information into the security context about *who* (which user) authenticated and *how* (with which authentication scheme) did they authenticate, and this information is processed by the target controller's *authorization* to determine if the credentials are suitable for interacting with the controller.

I implemented controller authorization with the goal of being able to determine on a per-controller basis, that:

1. which authentication schemes can be used for calling the controller's methods,
2. which is the lowest session assurance level, that is needed for interacting with the controller's methods.

The controller authorization was implemented with Spring's *method security* features, using the `@PreAuthorize` annotation. This essentially enables the developer to specify pre-defined or custom methods, which are wrapped around controllers using proxies [43], and need to evaluate to true in order to the request be forwarded to the controller. Evaluating to false causes the request to be rejected with an *AccessDenied* error.¹⁹

I implemented a set of custom *method security expressions* in order to be used with controller authorization. Each authentication scheme required a different method security expression. Method security expressions are provided the required level of assurance as parameters. This means that a controller, which requires all requests to be authenticated with a session having password-elevated assurance level, needs to be only annotated with the `authenticatedBySessionWithAssuranceLevel('PasswordElevatedSession')` within a `@PreAuthorize()` expression, and no further configuration is required.

Figure 7.7 displays the internal logic I built into the controller proxy authorization. If a request passes controller proxy authorization as well, it is forwarded to the controller.

¹⁹Due to the only application of controller authorization is to determine if the user has a sufficient session assurance level, I mapped the *AccessDenied* error directly to *SessionElevationRequired*. If the request is rejected because of an authentication scheme mismatch at this point, that can only mean a faulty or outdated client.

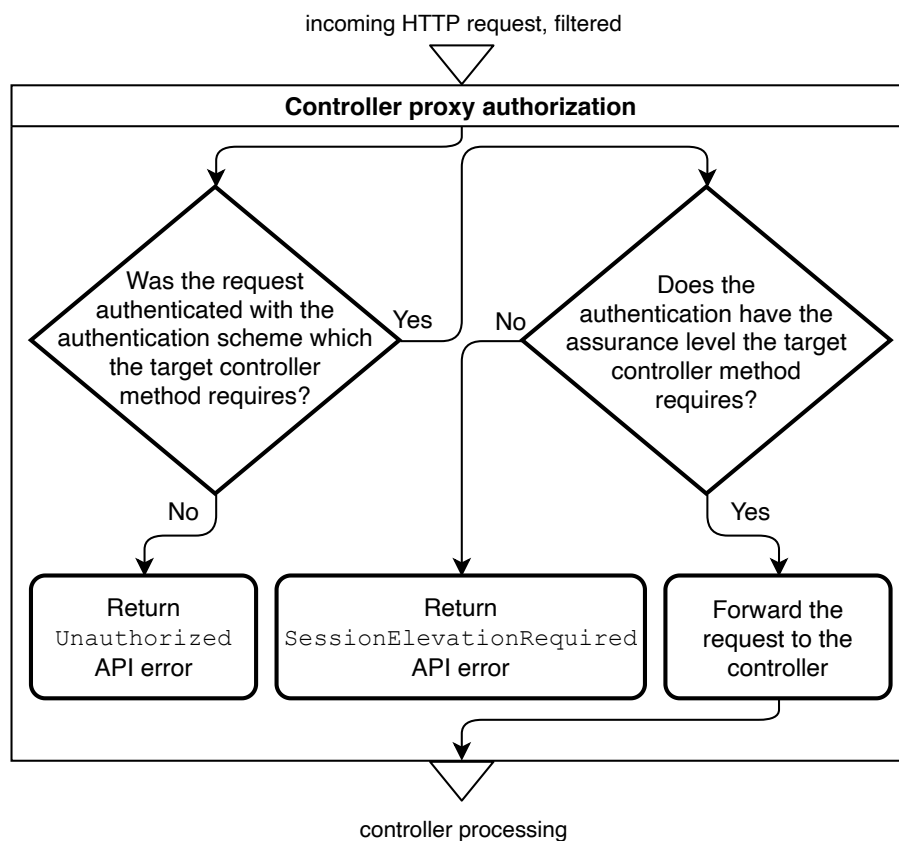


Figure 7.7 A summary of controller authorization

7.5 Encrypting sensitive database data

Although the database is only accessible with securely stored credentials with audited access, and the database machine itself is also sealed off the Internet, I wanted to take extra measures for database security. Although Neo4j offers at-rest encryption [135], it is not integrated into the database, and it still does not prevent data breaches when the database machine itself gets compromised.

My primary requirement for database data encryption is that it should be completely transparent in terms of application logic, meaning it should not require additional code to fetch or decrypt an encrypted database value. My second requirement is that the encryption should happen on a per-property basis, enabling plain and encrypted values to be stored in the database as well. My final requirement is that the encryption key(s) should not be stored in the database, and should not even relate to the database. The database in itself should not even know if a given property of an entity is encrypted or not.

For these requirements, I set up a *database encryption key* key in the *kv-prodcreds-prod-001* Key Vault, and attached this key to the back end application's configuration as a referenced environment variables, the same way as I did for other access credentials detailed in Chapter 4. Then I configured attribute converters²⁰ provided for Neo4j for handling those types of data, which I wanted to encrypt. In the attribute converter, I used the *DiplomatiqAEAD* primitive to securely encrypt, or decrypt and authenticate the data, using the database encryption key mentioned above.

As attribute converters are fully transparent and require no additional code, the first requirement was met. Encryption happens on a per-property basis — individual properties are encrypted instead of whole entities or the whole database —, so my second requirement is met. The encryption key is stored outside the database, outside the database machine, in a secure, audited Key Vault service, so my third requirement is also met.

This scheme protects the data in a different way than traditional at-rest database encryption. Even if attackers manage to steal the entire database, or even if they somehow obtain root-level access to the database machine, they will not be able to acquire sensitive data stored in the database, since all sensitive data is encrypted with an encryption key stored outside the database machine. In this hypothetical scenario, the attackers would additionally need to steal the database encryption key as well, either from the Key Vault service, or from the back end application's servers. Both scenarios are highly unlikely. This provides a layered security model to data stored in Diplomatiq's database.

As encryption and decryption happens on the back end application's servers, the database machine's performance constraints do not apply to encrypting or decrypting database data. I performed several measurements to see if it is any slower to save or retrieve encrypted data than plain data, but the difference proved not to be measurable.

²⁰Attribute converters for Neo4j are responsible for serializing unsupported data types to other data types which Neo4j can natively store. Attribute converters can host arbitrary logic.

Chapter 8

Conclusion and future work

My primary object was to create the Diplomatiq social network software system. To build and publish the system, this involved creating several other supportive libraries, and a company-level, production-grade server infrastructure as well, with several kinds of supportive infrastructure.

The initial version of the Diplomatiq software was released as a publicly available web application, and is suitable for organizing MUN conferences. The application was implemented upon a modern, layered security architecture, which provides cryptographic assurances in terms of application and data security.

8.1 Summary of contributions

I have achieved the following engineering contributions:

- I designed, built, secured and paid a company-level production infrastructure for the development, testing, production operation and maintenance of Diplomatiq, including several kinds of supportive infrastructure.
- I designed an application security framework addressing authentication, authorization, and data protection, which provides cryptographic assurances regarding access control and the confidentiality and integrity of sensitive data.
- I designed, implemented and published an elementary version of the Diplomatiq social network software system as a client-server application, using graph database technologies and the aforementioned security framework.
- I developed several supportive libraries outside the Diplomatiq software along the way. I published the built artefacts of these libraries with detailed documentation, for free use in the open-source community.
- I published the source code of all my contributions as separate open-source projects, centralized under one project organization, called Diplomatiq.

8.2 Future work

The goal of the work described in this thesis is to build a solid foundation for Diplomatiq, the social network software system, and also Diplomatiq, the prospective company, developing and maintaining the application in the future. The foundation was created along with a minimal feature set. However, much more features need to be implemented into Diplomatiq, so it can be a truly powerful tool for organizing Model United Nations conferences, and for enabling delegates to prepare to conferences better.

Acknowledgements

I would like to thank my supervisor Márk Asztalos for his advice and enthusiasm, and for letting me work in my own pace.

I would like to thank Roland Hidvégi for creating the corporate identity of Diplomatiq free of charge. I would also wish to express my gratitude to my colleagues, Péter Bartha and István Hartung, for pushing and encouraging me. I would also like to thank Kristóf Marussy for creating this \LaTeX template.

Last but not least, I am deeply grateful to my girlfriend, my mother, my father, my grandmother and my grandfather for their continuous support and understanding.

References

- [1] Merriam-Webster Dictionary. *Diplomacy*. URL: <https://www.merriam-webster.com/dictionary/diplomacy> (visited on 05/31/2020).
- [2] Encyclopædia Britannica Dictionary. *Foreign policy*. URL: <https://www.britannica.com/topic/foreign-policy> (visited on 05/31/2020).
- [3] United Nations. *Member States*. URL: <https://www.un.org/en/member-states/index.html> (visited on 05/31/2020).
- [4] United States Department of State. *Foreign Service Officer Qualifications*. URL: <https://careers.state.gov/work/foreign-service/officer/13-dimensions> (visited on 05/31/2020).
- [5] United States Department of State. *Key Topics – Foreign Service Institute*. URL: <https://www.state.gov/key-topics-foreign-service-institute> (visited on 05/31/2020).
- [6] mymun GmbH. *Full MUN List*. URL: <https://mymun.com/muns/list> (visited on 05/31/2020).
- [7] The Hague International Model United Nations. *About THIMUN*. URL: <https://thehague.thimun.org/about> (visited on 05/31/2020).
- [8] United Nations. *Rules of Procedure*. URL: <https://www.un.org/en/model-united-nations/rules-procedure> (visited on 05/31/2020).
- [9] United Nations. *Charter of the United Nations*. URL: <https://www.un.org/en/sections/un-charter/un-charter-full-text> (visited on 05/31/2020).
- [10] United Nations. *Subsidiary organs of the General Assembly*. URL: <https://www.un.org/en/ga/about/subsidiary/index.shtml> (visited on 05/31/2020).
- [11] United Nations. *Main Organs*. URL: <https://www.un.org/en/sections/about-un/main-organs> (visited on 05/31/2020).
- [12] WiseMee. *The History of the First MUN*. URL: <https://www.wisemee.com/history-of-the-first-mun> (visited on 05/31/2020).

- [13] Encyclopædia Britannica Dictionary. *League of Nations*. URL: <https://www.britannica.com/topic/League-of-Nations> (visited on 05/31/2020).
- [14] Best Delegate. *10 Top Events in the History of Model United Nations*. URL: <https://bestdelegate.com/10-top-events-in-the-history-of-model-united-nations> (visited on 05/31/2020).
- [15] MUNPlanet. *Website*. URL: <http://munplanet.com> (visited on 05/31/2020).
- [16] Facebook. *MUNPlanet*. URL: <https://www.facebook.com/MUNPlanet> (visited on 05/31/2020).
- [17] MyMUN. *Website*. URL: <https://mymun.com> (visited on 05/31/2020).
- [18] Best Delegate. *How Big is Model United Nations?* URL: <https://bestdelegate.com/how-big-is-model-united-nations> (visited on 05/31/2020).
- [19] Best Delegate. *About Best Delegate*. URL: <https://bestdelegate.com/about> (visited on 05/31/2020).
- [20] Tibor Jordán, András Recski, and Dávid Szeszler. *System optimization*. Typotex Kiadó, 2004. ISBN: 978-963-9548-39-8.
- [21] Soma Lucz. *BIMUN, nostalgically*. URL: <https://www.bimun.hu/history> (visited on 05/31/2020).
- [22] Stephen P. Borgatti, Ajay Mehra, Daniel J. Brass, and Giuseppe Labianca. "Network Analysis in the Social Sciences". In: *Science* 323.5916 (2009), pp. 892–895. DOI: 10.1126/science.1165821.
- [23] Jamie F. Metzl. "Network Diplomacy". In: *Georgetown Journal of International Affairs* 2.1 (2001), pp. 77–87. ISSN: 15260054. URL: <http://www.jstor.org/stable/43133985>.
- [24] Linton C. Freeman. "Visualizing Social Networks". In: *Journal of Social Structure* 1 (). URL: <https://www.cmu.edu/joss/content/articles/volume1/Freeman.html>.
- [25] Evelien Otte and Ronald Rousseau. "Social network analysis: a powerful strategy, also for the information sciences". In: *Journal of Information Science* 28.6 (2002), pp. 441–453. DOI: 10.1177/016555150202800601.
- [26] Mike Thelwall. "Chapter 2 Social Network Sites: Users and Uses". In: *Social Networking and The Web*. Vol. 76. Advances in Computers. Elsevier, 2009, pp. 19–73. DOI: 10.1016/S0065-2458(09)01002-X.
- [27] Facebook. *About Facebook*. URL: <https://about.fb.com> (visited on 05/31/2020).
- [28] Twitter. *About Twitter*. URL: <https://about.twitter.com> (visited on 05/31/2020).
- [29] Instagram. *About Instagram*. URL: <https://about.instagram.com/about-us> (visited on 05/31/2020).

- [30] Facebook Investor Relations. *Facebook Reports First Quarter 2020 Results*. URL: <https://investor.fb.com/investor-news/press-release-details/2020/Facebook-Reports-First-Quarter-2020-Results/default.aspx> (visited on 05/31/2020).
- [31] Facebook. *Understanding Facebook's Business Model*. URL: <https://about.fb.com/news/2019/01/understanding-facebook-business-model> (visited on 05/31/2020).
- [32] TechCrunch. *Twitter officially expands its character count to 280 starting today*. URL: <https://techcrunch.com/2017/11/07/twitter-officially-expands-its-character-count-to-280-starting-today> (visited on 05/31/2020).
- [33] Twitter Investor Relations. *Quarterly results*. URL: <https://investor.twitterinc.com/financial-information/quarterly-results/default.aspx> (visited on 05/31/2020).
- [34] Innovation Tactics. *Twitter Business Model Canvas*. URL: <https://innovationtactics.com/twitter-business-model-canvas> (visited on 05/31/2020).
- [35] TechCrunch. *Facebook Buys Instagram For \$1 Billion, Turns Budding Rival Into Its Standalone Photo App*. URL: <https://techcrunch.com/2012/04/09/facebook-to-acquire-instagram-for-1-billion> (visited on 05/31/2020).
- [36] LinkedIn. *About LinkedIn*. URL: <https://about.linkedin.com> (visited on 05/31/2020).
- [37] DeviantArt. *About DeviantArt*. URL: <https://about.deviantart.com> (visited on 05/31/2020).
- [38] Business Strategy Hub. *How does LinkedIn make money?* URL: <https://bstrategyhub.com/linkedin-business-model-how-does-linkedin-make-money> (visited on 05/31/2020).
- [39] Josh Lim on Quora. *How does DeviantArt make money?* URL: <https://www.quora.com/How-does-DeviantArt-make-money/answer/Josh-Lim-8> (visited on 05/31/2020).
- [40] Facebook. *MyMun*. URL: <https://www.facebook.com/mymun> (visited on 05/31/2020).
- [41] ZoomInfo. *MyMun*. URL: <https://www.zoominfo.com/c/mymun/461056660> (visited on 05/31/2020).
- [42] Stack Overflow. *Programming, Scripting, and Markup Languages, 2020 Developer Survey*. URL: <https://insights.stackoverflow.com/survey/2020#technology-programming-scripting-and-markup-languages> (visited on 05/31/2020).

- [43] Phillip Webb, Dave Syer, Josh Long, Stéphane Nicoll, Rob Winch, Andy Wilkinson, Marcel Overdijk, Christian Dupuis, Sébastien Deleuze, Michael Simons, Vedran Pavić, Jay Bryant, Madhura Bhawe, Eddú Meléndez, and Scott Frederick. *Spring Boot Reference Documentation*. URL: <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle> (visited on 05/31/2020).
- [44] ArangoDB GmbH. *Multi-model highly available NoSQL database - ArangoDB*. URL: <https://www.arangodb.com> (visited on 05/31/2020).
- [45] DataStax. *DataStax | NoSQL Database Built on Apache Cassandra*. URL: <https://www.datastax.com> (visited on 05/31/2020).
- [46] Callidus Software Inc. *Graph Database | Multi-Model Database | OrientDB*. URL: <http://orientdb.com> (visited on 05/31/2020).
- [47] The GraphQL Foundation. *A query language for your API*. URL: <https://graphql.org> (visited on 05/31/2020).
- [48] DB-Engines. *Graph DBMS*. URL: <https://db-engines.com/en/article/Graph+DBMS> (visited on 05/31/2020).
- [49] DB-Engines. *DB-Engines Ranking of Graph DBMS*. URL: <https://db-engines.com/en/ranking/graph+dbms> (visited on 05/31/2020).
- [50] Neo Technology. *Neo4j*. URL: <https://github.com/neo4j> (visited on 05/31/2020).
- [51] Neo Technology. *Neo4j Licensing*. URL: <https://neo4j.com/licensing> (visited on 05/31/2020).
- [52] Neo Technology. *Neo4j Startup Program*. URL: <https://neo4j.com/startup-program> (visited on 05/31/2020).
- [53] Neo Technology. *Intro to Cypher*. URL: <https://neo4j.com/developer/cypher-query-language> (visited on 05/31/2020).
- [54] GitHub. *Initial revision of "git", the information manager from hell*. URL: <https://github.com/git/git/commit/e83c5163316f89bfbde7d9ab23ca2e25604af290> (visited on 05/31/2020).
- [55] GitHub. *Showing 44,103,710 available users*. URL: <https://github.com/search?q=type%3Auser&type=Users> (visited on 05/31/2020).
- [56] GitHub. *Pricing*. URL: <https://github.com/pricing> (visited on 05/31/2020).
- [57] WebsiteSetup. *How to Choose a Domain Name*. URL: <https://websitesetup.org/choose-domain-name> (visited on 05/31/2020).
- [58] Nelson Cowan. "The magical number 4 in short-term memory: a reconsideration of mental storage capacity". In: 24.1 (2001), pp. 87–185. DOI: 10.1017/S0140525X01003922.

- [59] fonts.com. *Eina By Extratype*. URL: <https://www.fonts.com/font/extratype/eina> (visited on 05/31/2020).
- [60] fonts.com. *Eina — Available Font Licenses For This Family*. URL: <https://www.fonts.com/font/extratype/eina/licenses> (visited on 05/31/2020).
- [61] P. Mockapetris. *Domain Names - Implementation and Specification*. RFC 1035. 1987. URL: <https://tools.ietf.org/html/rfc1035>.
- [62] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. 2018. URL: <https://tools.ietf.org/html/rfc8446>.
- [63] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. *DNS Security Introduction and Requirements*. RFC 4033. 2005. URL: <https://tools.ietf.org/html/rfc4033>.
- [64] S. Ariyapperuma and C. J. Mitchell. “Security vulnerabilities in DNS and DNSSEC”. In: *The Second International Conference on Availability, Reliability and Security (ARES’07)*. 2007, pp. 335–342.
- [65] Thomas Ptacek. *Against DNSSEC*. URL: <https://news.ycombinator.com/item?id=8894902> (visited on 05/31/2020).
- [66] Thomas Ptacek. *14 DNS Nerds Don’t Control The Internet*. URL: <https://sockpuppet.org/blog/2016/10/27/14-dns-nerds-dont-control-the-internet> (visited on 05/31/2020).
- [67] Mozilla Developer Network. *Features restricted to secure contexts*. URL: https://developer.mozilla.org/en-US/docs/Web/Security/Secure_Contexts/features_restricted_to_secure_contexts (visited on 05/31/2020).
- [68] Sectigo. *Website*. URL: <https://sectigo.com> (visited on 05/31/2020).
- [69] National Institute of Standards and Technology. “Security Requirements for Cryptographic Modules”. In: (2019). DOI: 10.6028/NIST.FIPS.140-3.
- [70] P. Hallam-Baker, R. Stradling, and J. Hoffman-Andrews. *DNS Certification Authority Authorization (CAA) Resource Record*. RFC 8659. 2019. URL: <https://tools.ietf.org/html/rfc8659>.
- [71] Ian D Foster, Jon Larson, Max Masich, Alex C Snoeren, Stefan Savage, and Kirill Levchenko. “Security by any other name: On the effectiveness of provider based email security”. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 2015, pp. 450–464.
- [72] J. Klensin. *Simple Mail Transfer Protocol*. RFC 5321. 2008. URL: <https://tools.ietf.org/html/rfc5321>.
- [73] S. Kitterman. *Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1*. RFC 7208. 2014. URL: <https://tools.ietf.org/html/rfc7208>.

- [74] D. Crocker, T. Hansen, and M. Kucherawy. *DomainKeys Identified Mail (DKIM) Signatures*. RFC 6376. 2011. URL: <https://tools.ietf.org/html/rfc6376>.
- [75] M. Kucherawy and E. Zwicky. *Domain-based Message Authentication, Reporting, and Conformance (DMARC)*. RFC 7489. 2015. URL: <https://tools.ietf.org/html/rfc7489>.
- [76] GitHub. *Open Source Guides*. URL: <https://opensource.guide> (visited on 05/31/2020).
- [77] Massachusetts Institute of Technology. *MIT License*. URL: <https://choosealicense.com/licenses/mit> (visited on 05/31/2020).
- [78] Massachusetts Institute of Technology. *Conventional Commits*. URL: <https://choosealicense.com/licenses/mit> (visited on 05/31/2020).
- [79] EditorConfig. *Website*. URL: <https://editorconfig.org> (visited on 05/31/2020).
- [80] Vincent Driessen. *A successful Git branching model*. URL: <https://nvie.com/posts/a-successful-git-branching-model> (visited on 05/31/2020).
- [81] Tom Preston-Werner. *Semantic Versioning 2.0.0*. URL: <https://semver.org> (visited on 05/31/2020).
- [82] Conventional Changelog. *Conventional Changelog Tools*. URL: <https://github.com/conventional-changelog> (visited on 05/31/2020).
- [83] Martin Fowler. *Continuous Integration*. URL: <https://martinfowler.com/articles/continuousIntegration.html> (visited on 05/31/2020).
- [84] GitHub. *GitHub Actions*. URL: <https://github.com/features/actions> (visited on 05/31/2020).
- [85] Travis CI. *Website*. URL: <https://travis-ci.org> (visited on 05/31/2020).
- [86] GitHub. *Events that trigger workflows*. URL: <https://help.github.com/en/actions/reference/events-that-trigger-workflows> (visited on 05/31/2020).
- [87] SonarSource. *Adding Coding Rules*. URL: <https://docs.sonarqube.org/latest/extend/adding-coding-rules> (visited on 05/31/2020).
- [88] D. Herron. *Node.js Web Development*. Packt Publishing, 2016. ISBN: 9781785885419. URL: <https://books.google.com/books?id=CwBwDQAAQBAJ>.
- [89] Node Package Manager. *Website*. URL: <https://npmjs.com> (visited on 05/31/2020).
- [90] Microsoft. *Recommended naming and tagging conventions*. URL: <https://docs.microsoft.com/en-us/azure/cloud-adoption-framework/ready/azure-best-practices/naming-and-tagging> (visited on 05/31/2020).
- [91] Microsoft. *Azure global infrastructure*. URL: <https://azure.microsoft.com/en-us/global-infrastructure> (visited on 05/31/2020).

- [92] Mozilla Developer Network. *Content Security Policy (CSP)*. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP> (visited on 05/31/2020).
- [93] Mozilla Developer Network. *Choosing between www and non-www URLs*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/BasicsofHTTP/Choosing_between_www_and_non-www_URLs (visited on 05/31/2020).
- [94] Bjørn Johansen. *To www or not to www – Should you use www or not in your domain?* URL: <https://www.bjornjohansen.com/www-or-not> (visited on 05/31/2020).
- [95] GitHub. *Creating and storing encrypted secrets*. URL: <https://help.github.com/en/actions/configuring-and-managing-workflows/creating-and-storing-encrypted-secrets> (visited on 05/31/2020).
- [96] Lucas Garron, Andrew Bortz, and Dan Boneh. *The state of HSTS deployment: a survey and common pitfalls*. 2013.
- [97] J. Hodges, C. Jackson, and A. Barth. *HTTP Strict Transport Security (HSTS)*. RFC 6977. 2012. URL: <https://tools.ietf.org/html/rfc6977>.
- [98] Google. *HSTS Preload List Submission*. URL: <https://hstspreload.org> (visited on 05/31/2020).
- [99] DigiCert. *Website*. URL: <https://www.digicert.com> (visited on 05/31/2020).
- [100] Microsoft. *Add a TLS/SSL certificate in Azure App Service*. URL: <https://docs.microsoft.com/en-us/azure/app-service/configure-ssl-certificate> (visited on 05/31/2020).
- [101] Aakanksha Saha, Tamara Denning, Vivek Srikumar, and Sneha Kasera. “Secrets in Source Code: Reducing False Positives using Machine Learning”. In: 2020, pp. 168–175. DOI: 10.1109/COMSNETS48256.2020.9027350.
- [102] Michael Meli, Matthew R. McNiece, and Bradley Reaves. “How Bad Can It Get? Characterizing Secret Leakage in Public GitHub Repositories”. In: *Network and Distributed System Security Symposium*. 2019.
- [103] Microsoft. *Azure App Configuration documentation*. URL: <https://docs.microsoft.com/en-us/azure/azure-app-configuration> (visited on 05/31/2020).
- [104] Neo4j. *Neo4j Aura*. URL: <https://neo4j.com/aura> (visited on 05/31/2020).
- [105] Microsoft. *Support for moving Azure resources across regions*. URL: <https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/region-move-support> (visited on 05/31/2020).
- [106] certbot. *Website*. URL: <https://certbot.eff.org> (visited on 05/31/2020).
- [107] Let’s Encrypt. *Website*. URL: <https://letsencrypt.org> (visited on 05/31/2020).
- [108] Bolt Protocol. *Website*. URL: <https://boltprotocol.org> (visited on 05/31/2020).

- [109] Microsoft. *What is subnet delegation?* URL: <https://docs.microsoft.com/en-us/azure/virtual-network/subnet-delegation-overview> (visited on 05/31/2020).
- [110] Mozilla Developer Network. *Crypto.getRandomValues()*. URL: <https://developer.mozilla.org/en-US/docs/Web/API/Crypto/getRandomValues> (visited on 05/31/2020).
- [111] Alexis Deveria. *Can I use getRandomValues?* URL: <https://caniuse.com/#search=getRandomValues> (visited on 05/31/2020).
- [112] Dr. Mads Haahr. *Introduction to Randomness and Random Numbers*. URL: <https://www.random.org/randomness> (visited on 05/31/2020).
- [113] B Ya Ryabko, VS Stognienko, and Yu I Shokin. "A new test for randomness and its application to some cryptographic problems". In: *Journal of statistical planning and inference* 123.2 (2004), pp. 365–376.
- [114] John Walker. *A Pseudorandom Number Sequence Test Program*. URL: <https://www.fourmilab.ch/random> (visited on 05/31/2020).
- [115] GitHub. *Polly*. URL: <https://github.com/App-vNext/Polly> (visited on 05/31/2020).
- [116] Sugimiyanto Suma, Rashid Mehmood, and Aiiad Albeshri. "Automatic Event Detection in Smart Cities Using Big Data Analytics". In: *Smart Societies, Infrastructure, Technologies and Applications*. Springer International Publishing, 2018, pp. 111–122. ISBN: 978-3-319-94180-6.
- [117] Iris Tien, Aibek Musaev, David Benas, Ameya Ghadi, Seymour Goodman, and Calton Pu. "Detection of Damage and Failure Events of Critical Public Infrastructure using Social Sensor Big Data". In: *IoTBD*. 2016, pp. 435–440.
- [118] Munawar Monzy Merza, John Coates, James Hansen, Lucas Murphey, David Hazekamp, Michael Kinsley, and Alexander Raitz. *Investigative and dynamic detection of potential security-threat indicators from events in big data*. US Patent 9,215,240. 2015.
- [119] Y. Lv, Y. Duan, W. Kang, Z. Li, and F. Wang. "Traffic Flow Prediction With Big Data: A Deep Learning Approach". In: *IEEE Transactions on Intelligent Transportation Systems* 16.2 (2015), pp. 865–873.
- [120] SmartBear Software. *OpenAPI Specification, Version 3.0.3*. URL: <https://swagger.io/specification> (visited on 05/31/2020).
- [121] Google. *Angular Reference Documentation*. URL: <https://angular.io/docs> (visited on 05/31/2020).
- [122] Mozilla Developer Network. *Fetch API*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (visited on 05/31/2020).

- [123] Positive Technologies. *Web Applications vulnerabilities and threats: statistics for 2019*. URL: <https://www.ptsecurity.com/ww-en/analytics/web-vulnerabilities-2020> (visited on 05/31/2020).
- [124] Open Web Application Security Project. *OWASP Web Security Testing Guide*. URL: <https://owasp.org/www-project-web-security-testing-guide> (visited on 05/31/2020).
- [125] Morris J Dworkin. *SP 800-38D. Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC*. National Institute of Standards & Technology, 2007.
- [126] Open Web Application Security Project. *Cross Site Scripting (XSS)*. URL: <https://owasp.org/www-community/attacks/xss> (visited on 05/31/2020).
- [127] Open Web Application Security Project. *OWASP Top Ten*. URL: <https://owasp.org/www-project-top-ten> (visited on 05/31/2020).
- [128] NIST. "Special Publication 800-63-3". In: *Digital Identity Guidelines* (2017). DOI: 10.6028/NIST.SP.800-63-3.
- [129] Nat Sakimura, John Bradley, Michael B. Jones, Breno de Medeiros, and Chuck Mortimore. *OpenID Connect Core 1.0*. URL: <https://owasp.org/www-community/attacks/xss> (visited on 05/31/2020).
- [130] T. Lodderstedt, M. McGloin, and P. Hunt. *The OAuth 2.0 Authorization Framework*. RFC 6749. 2012. URL: <https://tools.ietf.org/html/rfc6749>.
- [131] T. Wu. *The SRP Authentication and Key Exchange System*. RFC 2945. 2000. URL: <https://tools.ietf.org/html/rfc2945>.
- [132] C. Percival and S. Josefsson. *The scrypt Password-Based Key Derivation Function*. RFC 7914. 2016. URL: <https://tools.ietf.org/html/rfc7914>.
- [133] International Organization for Standardization. *ISO 8601 Date and Time Format*. URL: <https://www.iso.org/iso-8601-date-and-time-format.html> (visited on 05/31/2020).
- [134] Amazon Web Services. *Signature Version 4 signing process*. URL: <https://docs.aws.amazon.com/general/latest/gr/signature-version-4.html> (visited on 05/31/2020).
- [135] Neo4j. *Neo4j & Thales Team Up to Offer Graph Data-at-Rest Encryption*. URL: <https://neo4j.com/news/neo4j-thales-graph-data-at-rest-encryption> (visited on 05/31/2020).

Appendix

A Lists

A.1 List of implemented projects

- diplomatiq-frontend: <https://github.com/Diplomatiq/diplomatiq-frontend>
- diplomatiq-backend: <https://github.com/Diplomatiq/diplomatiq-backend>
- website: <https://github.com/Diplomatiq/website>
- crypto-random: <https://github.com/Diplomatiq/crypto-random>
- resily: <https://github.com/Diplomatiq/resily>
- convertibles: <https://github.com/Diplomatiq/convertibles>
- eslint-config-tslib: <https://github.com/Diplomatiq/eslint-config-tslib>
- eslint-config-angular: <https://github.com/Diplomatiq/eslint-config-angular>
- project-config: <https://github.com/Diplomatiq/project-config>

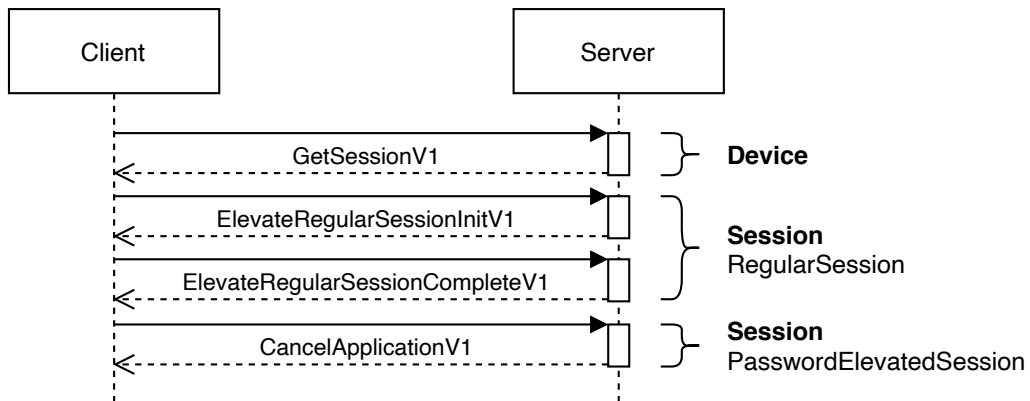
B Tables

B.1 Email addresses configured for diplomatiq.org

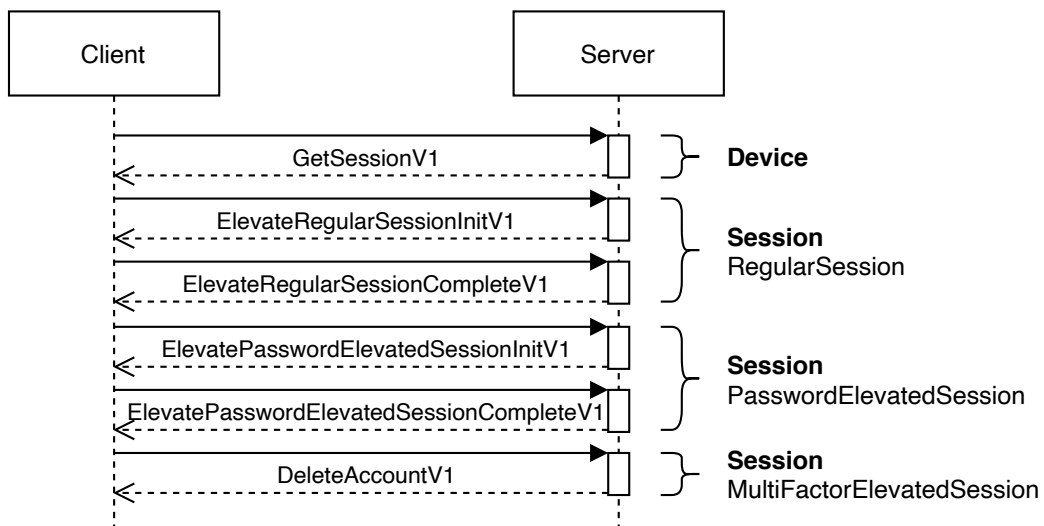
Email address	Usage	Notes
billing@diplomatiq.org	sending & receiving	for billing-related notifications
ceo@diplomatiq.org	sending & receiving	for communicating as the CEO
conduct@diplomatiq.org	receiving	for Code of Conduct violations
github@diplomatiq.org	sending & receiving	for the GitHub service
info@diplomatiq.org	receiving	for personal inquiries
npm@diplomatiq.org	sending & receiving	for Microsoft services
npm@diplomatiq.org	sending & receiving	for the NPM service
security@diplomatiq.org	receiving	for security inquiries
soma.lucz@diplomatiq.org	sending & receiving	for communicating as myself
team@diplomatiq.org	sending	for sending transactional email

C Figures

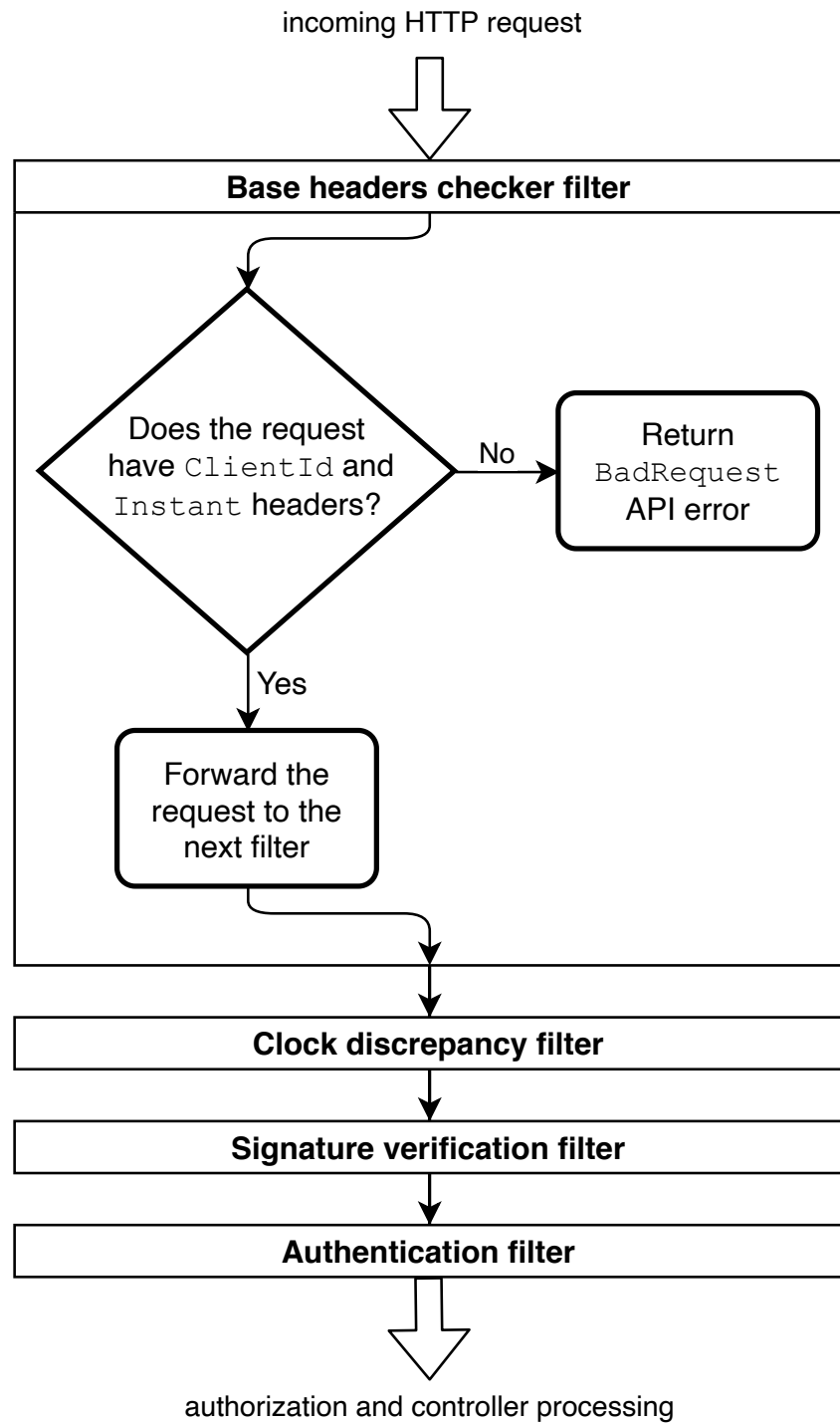
C.1 Server calls for elevating a session to password assurance level



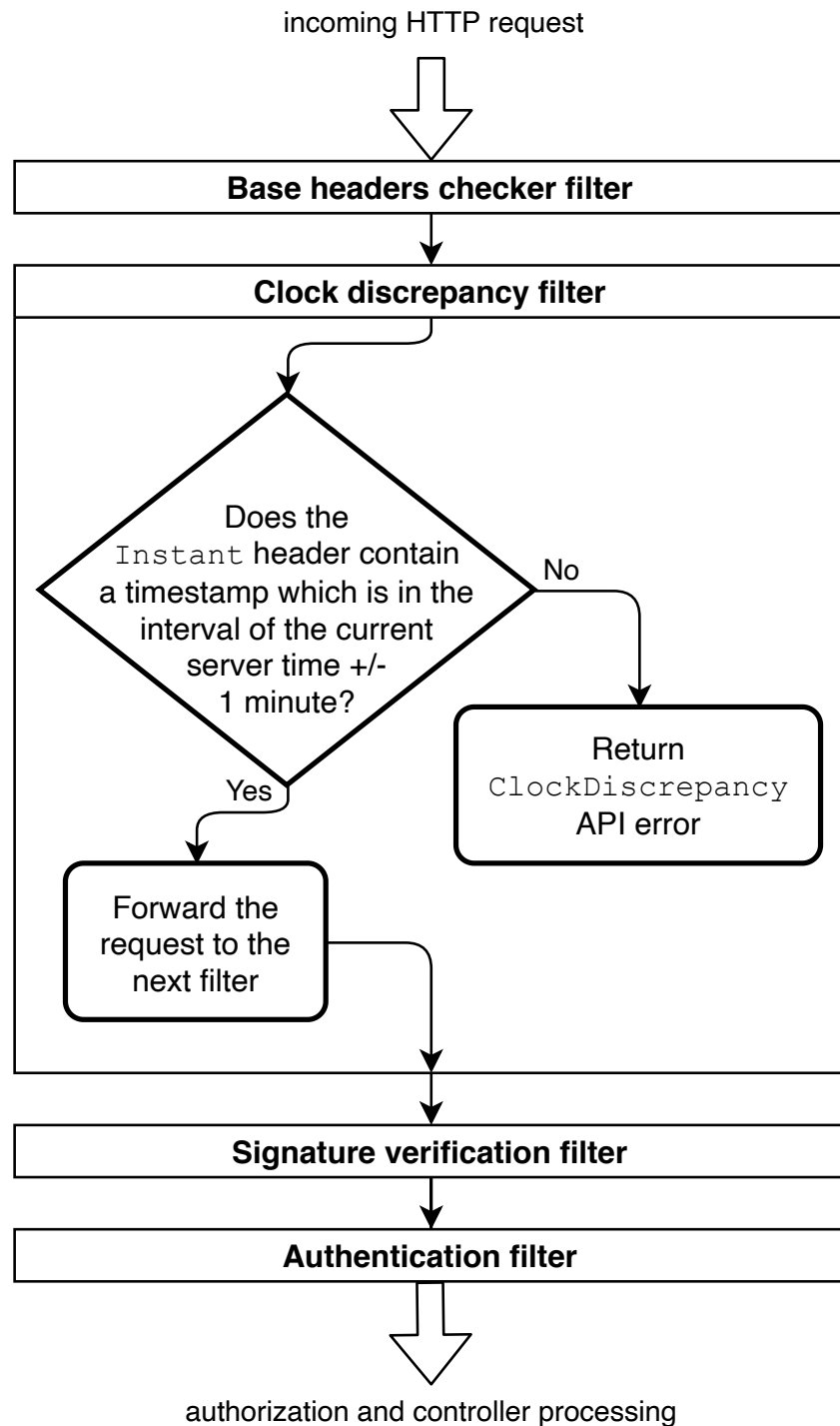
C.2 Server calls for elevating a session to multi-factor assurance level



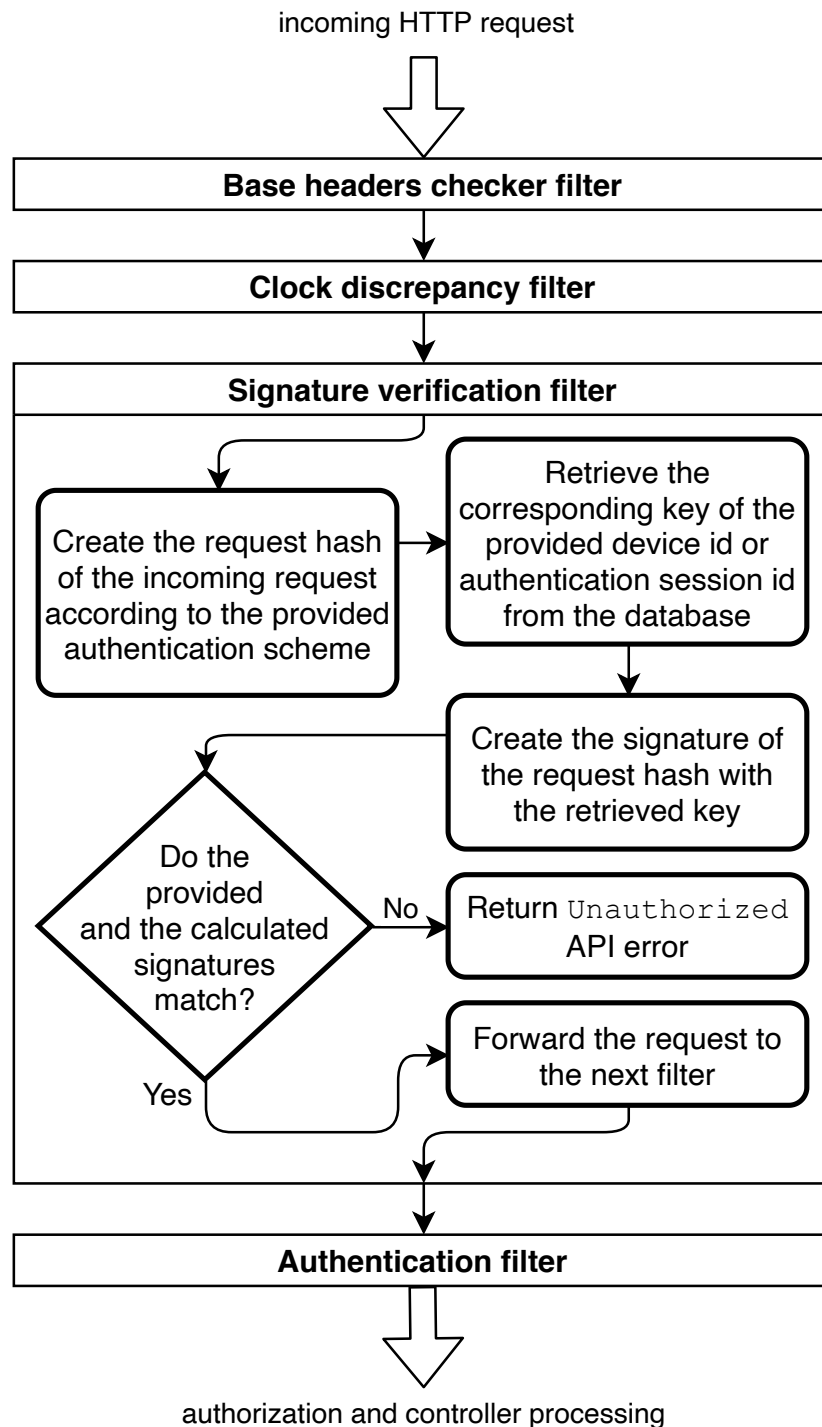
C.3 Operation of the base headers checker filter



C.4 Operation of the clock discrepancy filter



C.5 Operation of the signature verification filter



C.6 Operation of the authentication filter

