



**André
Fidalgo Ventura**

**Sistema colaborativo para a identificação de
cultivares de camélia**

**Crowdsourcing information system for camellia
cultivar identification**



**André
Fidalgo Ventura**

**Sistema colaborativo para a identificação de
cultivares de camélia**

**Crowdsourcing information system for camellia
cultivar identification**

“I have not failed. I’ve just found 10,000 ways that won’t work.”

— Thomas A. Edison



**André
Fidalgo Ventura**

**Sistema colaborativo para a identificação de
cultivares de camélia**

**Crowdsourcing information system for camellia
cultivar identification**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica dos doutores António Guilherme Rocha Campos e Hélder Troca Zagalo, professores auxiliares do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

This dissertation is dedicated to my parents, who have given me the opportunity of an education and support throughout my life.

o júri / the jury

presidente / president

Prof. Doutor Joaquim Arnaldo Carvalho Martins
professor catedrático da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Fernando Joaquim Lopes Moreira
professor associado do departamento de Inovação, Ciência e Tecnologia
da Universidade Portucalense

Prof. Doutor Hélder Troca Zagalo
professor auxiliar da Universidade de Aveiro

**agradecimentos /
acknowledgements**

This dissertation would never have been possible without the ideas, friendship, support and assistance of numerous people. Foremost among them are of course ANTÓNIO GUILHERME CAMPOS and HÉLDER TROCA ZAGALO, who are a constant source of motivation, ideas and inspiration, and I feel enormously privileged to have them as my supervisors.

I thank all of my friends for the great times we've had together and for pushing me forward when needed.

Most importantly, I thank my mother ISABEL GUIMARÃES and father ÉLIO VENTURA who were always supportive and where I needed them to be. I'm grateful for the trust, support and opportunity they gave me to be who I am now.

Palavras Chave

base de dados de espécimes, registo de cultivares, sistema colaborativo, reputação, desenvolvimento web.

Resumo

A identificação de cultivares de camélia é difícil devido à falta de informação completa e sistemática sobre os mais de 20 000 cultivares registados, e não pode ser alcançada através de esforços individuais desconexos. Esta dissertação descreve uma plataforma colaborativa para a identificação de cultivares de camélia, aproveitando as capacidades das tecnologias de informação e comunicação actuais. A estratégia proposta é baseada no conceito de crowd-sourcing. Os registos neste sistema têm origem em pedidos de identificação de cultivares enviados online. O sistema está equipado com um registo de nomes de cultivares. Através de questionários de identificação apresentados online, o sistema recolhe, gradualmente, as respostas aos pedidos de identificação. Simultaneamente, com base no desempenho dos participantes dos questionários, o sistema aplica métricas para estabelecer a sua reputação e pesar as suas respostas no cálculo da probabilidade de um espécime pertencer a um determinado cultivar. Os pedidos de identificação podem ser respondidos com um grau quantificável de certeza, de acordo com o número de respostas, a sua concordância e a reputação de cada participante.

Keywords

specimen database, cultivar register, crowdsourcing system, reputation, web development.

Abstract

Camellia cultivar identification is difficult due to the lack of complete, systematic information about the more than 20,000 registered cultivars, and cannot be achieved through disjointed individual efforts. This dissertation describes a collaborative platform for camellia cultivar identification harnessing the capabilities of modern information and communication technologies. The strategy proposed is based on the crowdsourcing concept. The entries on this system originate in cultivar identification requests submitted online. The system is equipped with a register of cultivar names. Through cultivar identification quizzes presented online, it gradually collects responses to the requests. Simultaneously, based on the performance of quiz respondents, the system applies metrics to establish their reputation and weigh their answers accordingly to calculate the probability of a specimen belonging to a given cultivar. User requests can be answered with a quantifiable degree of certainty according to the number of answers, their agreement and the reputation of each respondent.

CONTENTS

CONTENTS	xvii
LIST OF FIGURES	xxi
LIST OF TABLES	xxiii
ACRONYMS	xxv
1 INTRODUCTION	27
1.1 Motivation	28
1.1.1 The Camellia	29
1.1.2 Historical significance	29
1.1.3 Economical significance	29
2 STATE OF THE ART	31
2.1 Existing efforts	31
2.2 Crowdsourcing	33
2.2.1 Crowdsourcing applied to camellias	33
2.3 Web reputation	34
2.4 Technologies	34
2.4.1 Open source	34
2.4.2 Web framework	34
2.4.3 Database	35
3 SYSTEM REQUIREMENTS AND MODELING	37
3.1 Actors	39
3.1.1 Guest	40
3.1.2 Registered user	40
3.1.3 Translator	40
3.1.4 Moderator	41
3.1.5 Administrator	41
3.2 Use cases	41
3.2.1 Create account	44
3.2.2 Login	45
3.2.3 Submit identification request	45
3.2.4 Answer quiz	45
3.2.5 View cultivar	45

3.2.6	Search cultivar by name	46
3.2.7	View specimen	46
3.2.8	View specimens nearby	46
3.2.9	View profile	47
3.2.10	Edit profile	47
3.2.11	Receive notification by email	47
3.2.12	Manage website languages/translations	47
3.2.13	Manage cultivar register	49
3.2.14	View user profiles and statistics	50
3.2.15	Approve identification request	51
3.2.16	Set member auto-approval of identification requests	51
3.2.17	Manage users	51
3.2.18	Define quiz parameters	53
3.2.19	Suggest new cultivar	53
3.2.20	Vote cultivar for a specimen	53
3.3	Supplementary specifications	54
3.3.1	Specimen database	54
3.3.2	Standard Specimens	57
3.3.3	User reputation and cultivar probability metrics	57
3.4	Domain Model	57
3.5	Activity Diagrams	60
3.5.1	Create account	60
3.5.2	Login	61
3.5.3	Submit identification request	62
3.5.4	Answer quiz	63
3.5.5	View cultivar	64
3.5.6	Search cultivar by name	65
3.5.7	View specimen	66
3.5.8	View specimens nearby	66
3.5.9	View profile	67
3.5.10	Edit profile	68
3.5.11	Receive notification by email	69
3.5.12	Manage website languages/translations	70
3.5.13	Manage cultivar register	74
3.5.14	View user profiles and statistics	77
3.5.15	Approve identification request	78
3.5.16	Set member auto-approval of identification requests	79
3.5.17	Manage users	80
3.5.18	Define quiz parameters	84
3.5.19	Suggest new cultivar	85
3.5.20	Vote cultivar for a specimen	86
4	IMPLEMENTATION	87
4.1	Python	87
4.1.1	Python Packages	88
4.1.2	Virtual environment tools	88
4.2	Django	89
4.2.1	Model-View-Controller	89
4.2.2	Reusable Apps	90
4.2.3	Models	91

4.2.4	Views	92
4.2.5	Templates	93
4.3	Django Project Structure	95
4.3.1	Specimens Application	96
4.3.2	Cultivars Application	99
4.3.3	Quizzes Application	100
4.3.4	Userprofiles Application	102
4.3.5	Mainpages Application	103
4.4	PostgreSQL	104
4.5	Bootstrap	107
4.5.1	Django integration	107
4.6	Web Interface	109
4.6.1	Main page	109
4.6.2	Identification request	110
4.6.3	Quizzes	113
4.6.4	Specimens	116
4.6.5	Cultivars	120
4.6.6	User profile	121
4.6.7	Manage users	123
4.6.8	About, FAQ, and Contact	123
4.7	Internationalization and localization	126
4.7.1	Column approach	126
4.7.2	Multi-row approach	126
4.7.3	Single translation table approach	127
4.7.4	Additional translation table approach	128
4.8	User reputation and cultivar probabilities	129
4.8.1	User reputation	130
4.8.2	Cultivar probabilities	130
4.9	Initial cultivar data	131
4.10	Revision Control	135
4.11	Domain name and web hosting	137
4.11.1	Domain name	137
4.11.2	Web hosting	137
5	SUMMARY & FURTHER WORK	139
5.1	Tests and validation	139
5.2	Summary	139
5.3	Further Work	139
5.3.1	Forum	140
5.3.2	Mobile application	140
5.3.3	Auto-identification of cultivars	140
	BIBLIOGRAPHY	141

LIST OF FIGURES

3.1	Prototype of a cultivar identification quiz	38
3.2	System actors	40
3.3	High-level system use case diagram	42
3.4	Manage users use case diagram	43
3.5	Manage cultivar register use case diagram	43
3.6	Manage website languages and translations use case diagram	44
3.7	Example 1 of morphological features (UPOV guidelines[25])	55
3.8	Example 2 of morphological features (UPOV guidelines[25])	56
3.9	High-level system class diagram	58
3.10	System class diagram	59
3.11	Activity diagram for the use case create account	60
3.12	Activity diagram for the use case login	61
3.13	Activity diagram for the use case submit identification request	62
3.14	Activity diagram for the use case answer quiz	63
3.15	Activity diagram for the use case view cultivar	64
3.16	Activity diagram for the use case search cultivar by name	65
3.17	Activity diagram for the use case view specimen	66
3.18	Activity diagram for the use case view specimens nearby	66
3.19	Activity diagram for the use case view profile	67
3.20	Activity diagram for the use case edit profile	68
3.21	Activity diagram for the use case receive notification by email	69
3.22	Activity diagram for the use case add language	70
3.23	Activity diagram for the use case update language	71
3.24	Activity diagram for the use case delete language	71
3.25	Activity diagram for the use case add translation	72
3.26	Activity diagram for the use case update translation	73
3.27	Activity diagram for the use case delete translation	73
3.28	Activity diagram for the use case add cultivar	74
3.29	Activity diagram for the use case update cultivar	75
3.30	Activity diagram for the use case accept suggested cultivar	76
3.31	Activity diagram for the use case delete cultivar	77
3.32	Activity diagram for the use case view user profiles and statistics	77
3.33	Activity diagram for the use case approve identification request	78
3.34	Activity diagram for the use case set member auto-approval of identification requests	79
3.35	Activity diagram for the use case create user	80
3.36	Activity diagram for the use case update user profile	81

3.37	Activity diagram for the use case update user role	82
3.38	Activity diagram for the use case delete user	83
3.39	Activity diagram for the use case define quiz parameters	84
3.40	Activity diagram for the use case suggest new cultivar	85
3.41	Activity diagram for the use case vote cultivar for a specimen	86
4.1	The Zen of Python, by Tim Peters	87
4.2	Dependency diagram for the Django project applications	95
4.3	Form fields images example for the identification request form	97
4.4	Bootstrap template example	107
4.5	Website: main page	109
4.6	Website: identification request form (location and photo)	111
4.7	Website: identification request form (some characteristics)	112
4.8	Website: quiz example	114
4.9	Website: selecting a cultivar for a quiz answer	115
4.10	Website: specimens list	116
4.11	Website: specimen example (photos, map and voted cultivars)	117
4.12	Website: specimen example (characteristics)	118
4.13	Website: specimen vote example	119
4.14	Website: cultivars list	120
4.15	Website: cultivar example	121
4.16	Website: user profile	122
4.17	Website: edit profile	122
4.18	Website: edit user profile through Django admin site	123
4.19	Website: about page	124
4.20	Website: contact page	125
4.21	Example of a Camellia Web Register cultivar page	131
4.22	Example of Git commits	136

LIST OF TABLES

4.1	Multi-language database: column approach	126
4.2	Multi-language database: multi-row approach	127
4.3	Multi-language database: single translation table approach	128
4.4	Multi-language database: additional translation table approach	129

ACRONYMS

ACID	Atomicity, Consistency, Isolation, Durability 35	ICT	Information and communication technology 28
API	Application Programming Interface 91	IP address	Internet Protocol address 137
DBMS	Database Management System 35, 87, 104	ISP	Internet service provider 137
DDNS	Dynamic Domain Name System (DNS) 137	JSON	JavaScript Object Notation 98, 132
DNA	Deoxyribonucleic Acid 28	MVC	Model-View-Controller 34, 89, 90, 93
DNS	Domain Name System xxv, 137	ORM	Object-Relational Mapper 91
DRY	Don't Repeat Yourself 88	PyPI	Python Package Index 88
FCCN	Fundação para a Computação Científica Nacional 137	SQL	Structured Query Language 35
FTP	File Transfer Protocol 138	TLD	Top Level Domain 137
GOE	Garden of Excellence 27	UPOV	International Union for the Protection of New Varieties of Plants 54, 110
GPS	Global Positioning System 54, 140	URL	Uniform Resource Locator 133
ICJ	International Camellia Journal 28, 29, 31	VPS	Virtual Private Server 137, 138
ICR	International Camellia Register 27–29, 31, 37, 99	WCR	Web Camellia Register 27, 29, 31, 131, 132, 134
ICS	International Camellia Society 27, 29, 31		

INTRODUCTION

THE breeding and naming of new camellia cultivars—a plant or grouping of plants selected for desirable characteristics that can be maintained by propagation—has been going on for centuries. The efforts put into camellia cultivar identification stem from the natural curiosity of owners and aficionados about the origin of different specimens. Cultivar identification can also add economic value, because there is growing interest in plants with historical provenance and, as pointed out in [1] with regard to Cornwall—where tourism is the single most lucrative industry—, garden visitors are prepared to pay a premium for complete, reliable information about them. The same paper also deplores the fact that camellias are often propagated and sold unidentified or—even worse—wrongly identified. The importance attributed to this subject is reflected in the criteria set by the [International Camellia Society \(ICS\)](#) for the recognition of International Camellia [Gardens of Excellence \(GOEs\)](#). Some of the criteria for being a camellia [GOE](#) are having a minimum collection of 200 cultivars or species, all with identifying labels, and maintaining a register of every specimen and their location [2]. Obviously, the [GOE](#) accolade itself is a valuable visitor attraction factor.

However, even without considering variations due to growing conditions (e.g. soil, climate) and sporting (i.e. when a part of a plant shows morphological differences from the rest of the plant, such as differences on foliage shape or color, flowers, or branch structure, often propagated to form new cultivars that retain the characteristics of the new morphology), camellia cultivar identification is a very demanding challenge. The first obstacle is the huge number of cultivars developed and recorded over the centuries. The [ICS's International Camellia Register \(ICR\)](#), a result of 50 years of painstaking data collection [3], publicly available online since 2008 as the [Web Camellia Register \(WCR\)](#) [4] lists over 20,000 entries. But the main difficulty is the lack of complete, systematic information about those registered cultivars. The temporal span covered by the [ICR](#) is

enormous; its entries are essentially text descriptions taken from catalogues which, barring occasional references to paintings or photographs, are normally rather incomplete, with disparate levels of detail. With no formal structure support for this information, there is no systematic way of searching for matches based on given specimen characteristics, making identification, even tentative, virtually impossible.

A crucial step in the massive task of filling in the missing information is to establish standard specimens of the cultivars (i.e. specimens which are known for sure to be from a certain cultivar). This is another tricky problem; recent research case-studies using **Deoxyribonucleic Acid (DNA)** testing have found specimens deemed to be of the same cultivar, producing seemingly identical flowers, that actually belonged to different camellia varieties [5]. Many **ICR** entries, especially the older ones, may be affected by this problem.

Countless identification studies have been reported in the literature; examples can be easily found in recent issues of the **International Camellia Journal (ICJ)**, such as [1], [6] and [7], to name only a few. The difficulty of the subject is invariably confirmed. Also, camellia-related websites tend to be poorly maintained and offer very disparate and incomplete identification data. Creating an efficient, easy-to-use and reliable cultivar identification system is the aim of this project. Considering all the difficulties summarised above and previous projects addressing this problem, it is clear that:

1. It cannot be achieved through disjointed individual efforts. It must be able to draw contributions from the whole camellia community, integrating the collected pieces of information in a single repository shared at global level.
2. It must be acknowledged as a long-term aim, because it can only be implemented gradually.
3. **Information and communication technology (ICT)** is a key tool for this purpose.

ICT is sometimes regarded as a panacea for all sorts of problems, but the strategy proposed here, although it does involve an information system of camellia specimens fed online, is by no means based on that misconception. Cultivar identification relies on the collaborative work of camellia aficionados and, essential as they may be, computers and Internet communication are just tools to promote and assist that collective effort. The key concept here is crowdsourcing, as explained in section 2.2 on page 33.

1.1 MOTIVATION

The motivation for this work arises from both the significance of camellias and the current lack of collaboration mechanisms in the efforts for specimen identification, as discussed in section 2.1 on page 31.

1.1.1 THE CAMELLIA

Camellia is a genus of flowering plants in the family Theaceae. The name Camellia was given by Carl Linnaeus (Carl von Linné) (1707–1778)—a Swedish botanist, physician, and zoologist, known as the father of modern taxonomy[8]—in honour of Jiří Josef Kamel (Georg Joseph Camel) (1661–1706), a Moravian (Czech) lay brother who travelled in Asia and wrote an account of the plants of Luzon—the largest island in the Philippines—which is included in the third volume of John Ray’s *Historia Plantarum* (1704)[9].

Camellias are important due to their historical and economical significance. The [International Camellia Society](#), a non-profit organisation devoted to the genus *Camellia*, founded April 1962 and with over two thousand members worldwide, is the official registration authority for the genus *Camellia*, maintaining the [International Camellia Register](#), available online since 2008 as the [Web Camellia Register](#). The ICS has an annual publication, the [International Camellia Journal](#), besides being an important centre for research in the genus *Camellia*, through its Otomo Fund[10]. The Society also holds regular congresses and meetings.

1.1.2 HISTORICAL SIGNIFICANCE

Camellias had been cultivated in the gardens of China and Japan for centuries before they were even known in Europe. There are indications that camellias may have been first introduced in Europe by the Portuguese during the 16th century, but active commercial spreading in Europe started in the early 18th century when the English prospected and brought back varied plants from their travels[11]. Portugal’s most famous horticulturist, José Marques Loureiro, wrote in 1882 that the first camellias arrived in Porto between 1808 and 1810, ordered by some well-known local amateurs[12].

Although there are thirty-two thousand entries of *Camellia* cultivars in the International Register, some of these are invalid names—synonyms, duplications—and many represent cultivars found in books and catalogues produced in Japan and China hundreds of years ago, as well as in Europe in the 19th century, which have since become extinct. The number of valid entries is possibly around fifteen thousand[13].

1.1.3 ECONOMICAL SIGNIFICANCE

The leaves of *Camellia sinensis*—the tea plant—have huge economic importance in Asia, and especially in the Indian subcontinent, but there’s also a significant trade of ornamental species like *Camellia japonica*, *Camellia oleifera* and *Camellia sasanqua* and their hybrids, of

which a large number of cultivars have been developed.

As an example, Camellias represent an important part of the nursery trade in New Zealand. The Camellia industry generates about \$2–4 million worth of domestic sales and \$0.4 million in export sales annually with most of the plants being used for amenity or ornamental purposes[13].

Beside the nursery trading, there is the economic importance associated with tourism. In this context, cultivar identification efforts can provide added value. Citing the example of Cornwall, England, where tourism is the single most lucrative industry, [1] underlines that garden visitors are prepared to pay a premium for complete, reliable historic information about ancient plant specimens.

STATE OF THE ART

THIS chapter will discuss the existing camellia identification efforts and examples of some related websites/projects, in section 2.1. The crowdsourcing technique—the concept behind this project—, how it works, its key principles, and some of the most known examples, are discussed in section 2.2 on page 33. The Web reputation—used both as the crowdsourcing reward principle and as a weight in the calculation of probabilities—is discussed in section 2.3 on page 34. Finally, there is a section about the Web development technologies, discussed in section 2.4 on page 34.

2.1 EXISTING EFFORTS

Countless identification studies have been reported in the literature; examples can be easily found in recent issues of the ICJ, such as naming of 19th century camellias[1], the identification, history, cultivation, and conservation of heritage camellias in Hawaii[6], and a personal search for pre-1900 camellia cultivars and their preservation[7], to name only a few. The difficulty of the subject is invariably confirmed. Also, camellia-related websites tend to be poorly maintained and offer very disparate and incomplete identification data. The official effort, the WCR (the online version of the ICR), lacks a coherent and easy reading information structure that would also be easy to update and allow contributions to be made.

There are some websites that have a list of specimens and photos, but they are not a collaboration effort in cultivar identification. Here are some examples:

Web Camellia Register (<http://camellia.unipv.it/camelliadb2>)

This website, developed by the University of Pavia, Italy, has a list of cultivars from the ICS and has documents, with the registry of plants, that can be downloaded. There's

a file for each alphabet letter and each file contains all the plants—which name begins with that letter—registered. There is a search tool that allows searching by cultivar name, species or description, and there is a list of all registered cultivars, totalling around twenty-two thousand, but there are almost no photos, just textual descriptions, and it is not possible to contribute directly.

As Camélias das Japoneiras (<http://ascameliasdasjaponeiras.com>)

This is the website from the Portuguese Association of Camellias (Associação Portuguesa das Camélias) and the last update was made in 2009. It contains a list of gardens with information and photographs, and a list of camellia specimens with some information but almost without any photographs. There's a photo album of nurseries, exhibitions and cities but only with photographs and without any other information.

Camélias Flavius Viveiros (<http://cameliasflavius.com>)

António J. S. Assunção's website with a photo album without information about the specimens.

There are also more generic websites including camellias, like The International Plant Names Index or the Plants For A Future:

The International Plant Names Index (IPNI) (<http://ipni.org>)

This is the website resulting from the collaboration between the Royal Botanic Gardens (Kew), the Harvard University Herbaria, and the Australian National Herbarium. It is possible to search, nevertheless there are no photographs and the website is very academic or specialized, containing only technical information on plants. It is possible to send a comment but it is not a system of active collaboration.

Plants For A Future (PFAF) (<http://pfaf.org>)

Existing since 1996 and belonging to “Plants For A Future, Registered Charity No.1057719”, this website contains a lot of information and news articles, has a database of plants and it is possible to search on it. Most of the records have photographs and useful and well presented information such as if the plant is edible, if it has a medicinal use, their physical characteristics, details of cultivation, propagation, etc. However, there is no collaboration system.

Outside the plant realm there are other areas like the register and conservation of butterflies.

Butterfly Conservation (<http://butterfly-conservation.org>)

The website of the English company Butterfly Conservation, having a searchable list of butterfly and moth species, with factsheets and plenty of detailed information. There is no collaboration system.

2.2 CROWDSOURCING

The term crowdsourcing, used by the first time in 2006 by the journalist Jeff Howe [14], refers to the practice of obtaining services, ideas or data content through the contribution of a large number of people, especially in online communities. The most remarkable and well-known example is undoubtedly the Wikipedia, the online encyclopedia whose content can be created, reviewed and improved by anyone. The success of this revolutionary concept is the best demonstration that information and communication technologies have the power to involve communities in tasks that would otherwise be extremely hard or even impossible to accomplish.

Crowdsourcing is proving useful in increasingly diverse areas of application. An interesting example along similar lines to those proposed here is the Treezilla project [15], based on the OpenTreeMap engine [16], whose purpose is to identify and map every tree in the UK, in an effort to raise awareness on the benefits of trees to the local environment.

2.2.1 CROWDSOURCING APPLIED TO CAMELLIAS

In general there are few websites that use a collaborative model to grow and enrich their content. Nevertheless, there are some exceptions like the Treezilla, a collaborative website for the record of Britain's trees, using a crowdsourcing approach, in a similar manner to the goal of this work on camellias.

Treezila (<http://treezilla.org>)

A platform that allows to record Britain's trees, resorting to crowdsourcing, wherein anyone can and is encouraged to help and register the trees nearby. It allows searching by species or location, and for each record there are several and well structured pieces of information.

For this to yield good results, the system must follow some key principles: it must offer stimulus or rewards, be simple and easy to use by anyone, and there must be some filtering of bad contributions so we can have only the good and clean results.

In this specific subject the stimulus will be given by a personal ranking or reputation that will build up as the user interacts with the system and answers to quizzes, thus helping with the identification requests; the simplicity will be given by an easy to use website; the filtering of bad contributions will be possible through the reputation that works as a stimulus at the same time, in a way that the contributions from people with a lower reputation will have a lower weight on the metrics that are used to calculate the probabilities of a given specimen belonging to a given cultivar.

2.3 WEB REPUTATION

As put by Randy Farmer and Bryce Glass, “Today’s Web is the product of over a billion hands and minds. Around the clock and around the globe, a world full of people are pumping out contributions small and large: full-length features on Vimeo; video shorts on YouTube; comments on Blogger; discussions on Yahoo! Groups; and tagged-and-titled Del.icio.us bookmarks.”[17], and there are many ways of defining, calculating and representing (e.g. percentages, five-star systems) a reputation.

Despite all the myriad of contexts in which reputation is being used today, in this project it is defined as the information used to make a value judgement about the users knowledge on camellias, and that information is then used as weight when calculating the probabilities of a specimen being of a given cultivar. It is discussed in more detail in section 4.8 on page 129.

2.4 TECHNOLOGIES

2.4.1 OPEN SOURCE

To implement the system, free and open source tools were preferred. These can deliver superior security and quality[18], as users are able to actually examine the code being used, which is not possible with propriety software. They also allow more flexibility, interoperability and better support options, as most have online communities with excellent documentation, forums, mailing lists, and wikis.[19].

2.4.2 WEB FRAMEWORK

Considering the existing Web application frameworks and narrowing them down to Java, Python and PHP, the chosen one was Django¹—written in Python² and based on the **Model-View-Controller (MVC)** pattern—as it encourages rapid development and clean, pragmatic design. “The Web framework for perfectionists with deadlines”[20]. Python and Django are discussed in more detail in section 4.1 on page 87 and in section 4.2 on page 89, respectively.

Two of most popular websites powered by the Django Web framework are Instagram[21], an online mobile photo-sharing, video-sharing and social networking service, and Pinterest[22], a Web and mobile application company that offers a visual discovery, collection, sharing, and storage tool.

¹<https://www.djangoproject.com/>

²<https://www.python.org/>

2.4.3 DATABASE

Regarding the database, the chosen object-relational database management system was PostgreSQL³, also open source.

PostgreSQL is a powerful, open source object-relational database system, with more than 15 years of active development[23] and offers many advantages over other database systems. There are no licensing costs, it runs on all major operating systems, it is fully **Atomicity, Consistency, Isolation, Durability (ACID)** compliant, it supports storage of binary large objects and its **Structured Query Language (SQL)** implementation strongly conforms to the ANSI-SQL:2008 standard. It has won merit from its users and industry recognition, having received a Linux New Media Award for Best Database System and being a five time winner of the The Linux Journal Editors' Choice Award for best **Database Management System (DBMS)**[23].

PostgreSQL is used by hundreds of companies (e.g. IMDB.com — The Internet Movie Database, Cisco, Skype)[24].

³<http://www.postgresql.org/>

SYSTEM REQUIREMENTS AND MODELING

THIS chapter presents the software requirements specification, describing—from the user’s perspective—how this product must work, and the system modeling.

The scope of the system is to give users an online platform where they can both get help and contribute to help other users with the identification of the cultivar of a camellia specimen.

The proposed solution to the problem presented in chapter 1 on page 27 is based on an online information system serving a community of registered users, built around and providing support to the following key elements:

- Specimen register (with both already identified and to be identified specimens);
- Cultivar identification requests;
- Cultivar identification quizzes;
- Cultivar register based on the **ICR**.

The specimen database will be fed primarily by identification requests submitted by the users. The community will be challenged to try and answer those requests through cultivar identification quizzes generated automatically. Respondents must choose from the system’s cultivar register. The **ICR** will be loaded beforehand but the register will remain dynamic and will allow respondents to suggest new cultivar names, which will only become permanent after approval by system moderators.

The quizzes are presented as identification expertise tests for entertainment, displaying flower photos and allowing access to additional information from the database entries under scrutiny—a prototype of a quiz is presented in figure 3.1 on the following page.

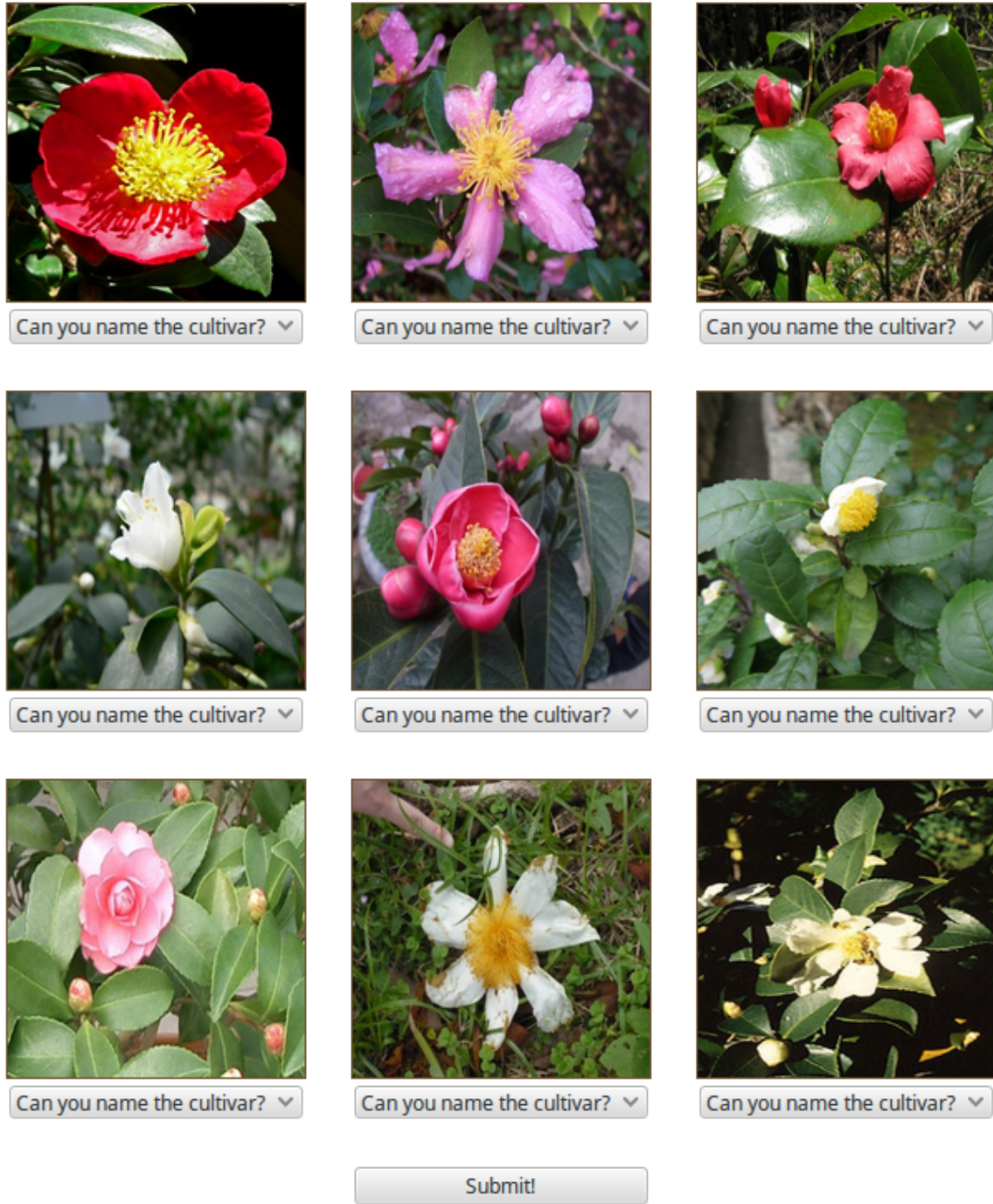


Figure 3.1: Prototype of a cultivar identification quiz

Statistical analysis of the answered quizzes will provide educated cultivar identification guesses. Metrics will be established to rate both the probability of correct cultivar identification and the identification expertise (reputation) of respondents. As new answered quizzes are taken into account, the system will be able to update these two ratings dynamically and also automatically notify the users concerned (identification requesters or quiz respondents) of any changes deemed significant. The expectation is that, as information becomes increasingly reliable, this will generate a ‘virtuous cycle’ of contributions to the system.

The specimen register may be empty to begin with, as it will be gradually fed by the identification requests; these can only be submitted by registered users. Quiz respondents must also be registered users, so that the system can keep track of their performance and update their rating accordingly.

To encourage participation, users may be rewarded based on number and completeness of identification forms submitted and/or performance as quiz respondents.

Since the philosophy of the system is to promote collaboration at global level, it will be designed to support a multilingual user interface.

The users and the goals that the user wants to achieve—identified from the aforementioned requirements—are presented as actors and use cases, respectively, in section 3.1 and section 3.2 on page 41.

3.1 ACTORS

The identified system actors and their relationships are shown in figure 3.2 on the next page and explained in the following subsections.

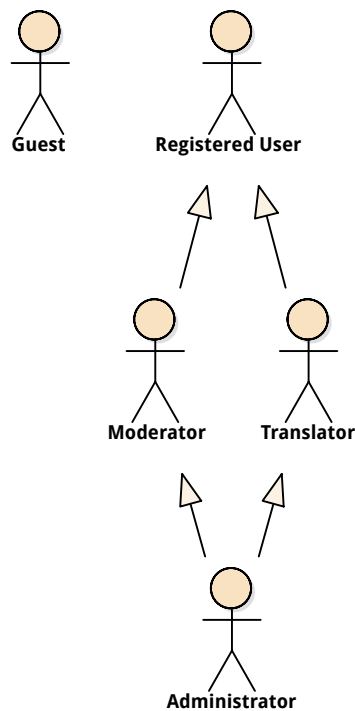


Figure 3.2: System actors

3.1.1 GUEST

The guest user is able to create an account (becoming a registered user), view cultivars, view specimens, search by cultivar name and view—on a map—specimens near them.

3.1.2 REGISTERED USER

The registered user is able to login, submit identification requests, answer quizzes, edit their own profile, receive notifications by email and has the roles of the guest actor with the exception of create an account.

3.1.3 TRANSLATOR

The translator inherits the role and properties of the registered user and is also able to manage the website languages and translations.

3.1.4 MODERATOR

The moderator inherits the role and properties of the registered user and is also able to approve identification requests, manage the cultivar register, view user profiles and statistics, and set auto-approval of identification requests to specific users.

3.1.5 ADMINISTRATOR

The administrator inherits the role and properties of the moderator and the translator and is also able to define quiz parameters and manage the users.

3.2 USE CASES

Figure 3.3 on the following page captures the requirements as a use case diagram presenting a technology independent view of the system.

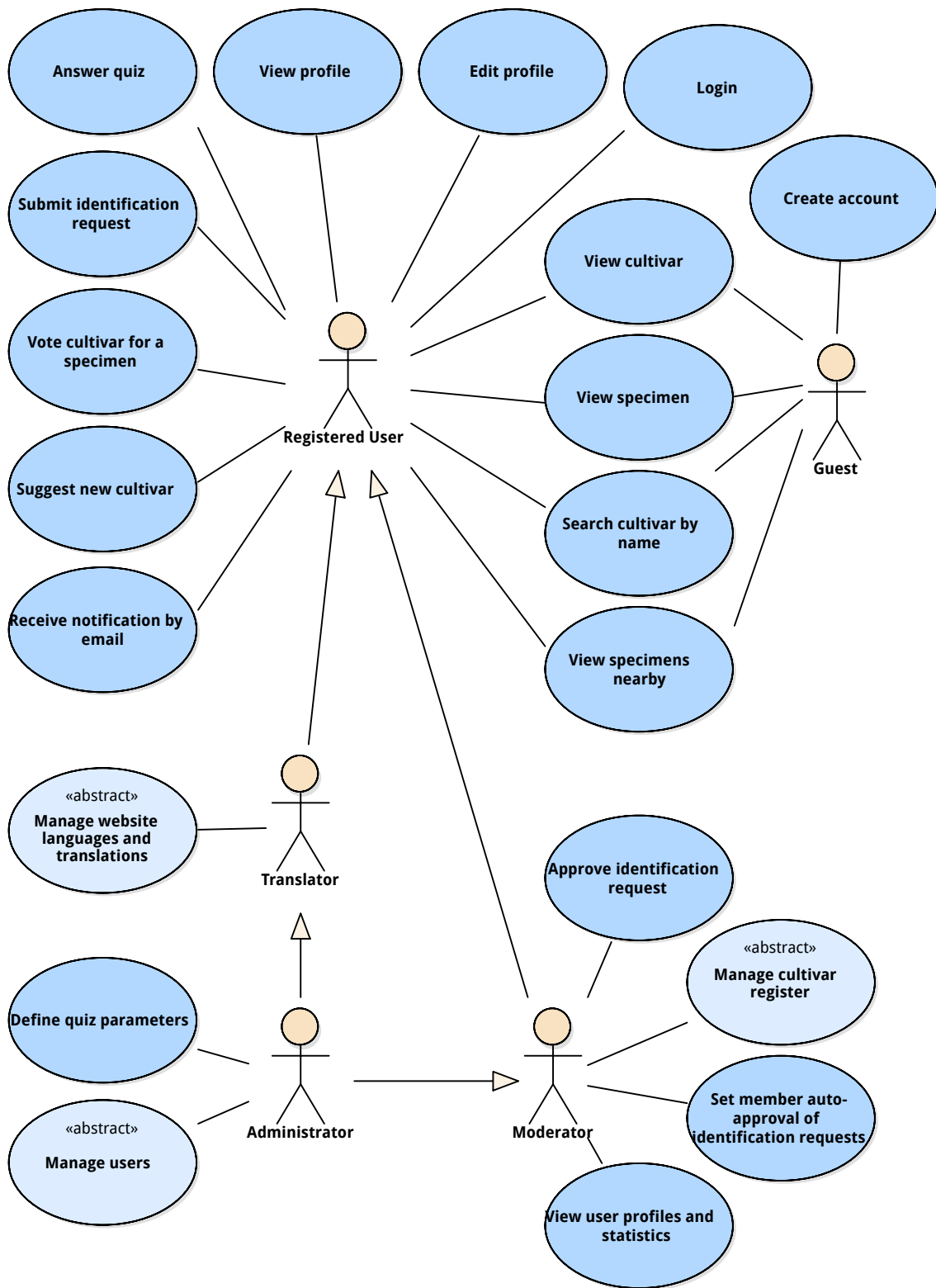


Figure 3.3: High-level system use case diagram

Manage users abstract use case is specialized by *Create User*, *Update User Profile*, *Update User Role* and *Delete User* use cases, as shown in figure 3.4 on the next page. The administrator

user, through the *update user role* use case, may set which role the user has: no role (registered user), moderator, translator, and administrator.

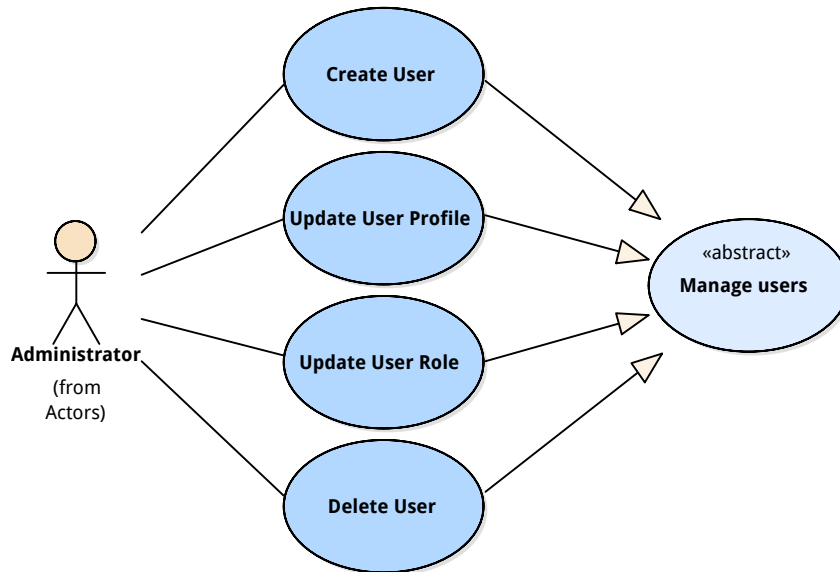


Figure 3.4: Manage users use case diagram

Manage cultivar register abstract use case is specialized by *Add Cultivar*, *Update Cultivar*, *Accept Suggested Cultivar* (when a user suggests a new cultivar a moderator must accept the suggestion) and *Delete Cultivar* use cases, as presented in figure 3.5.

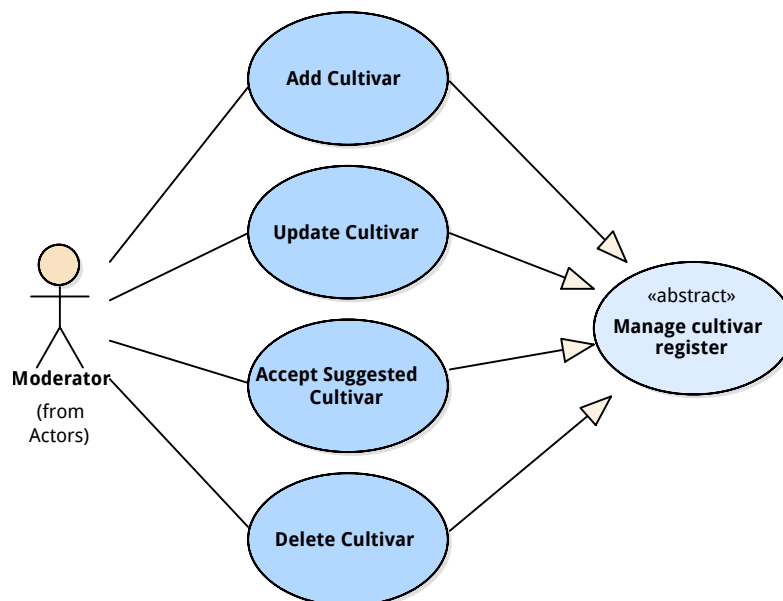


Figure 3.5: Manage cultivar register use case diagram

Manage website languages/translations abstract use case is specialized by *Add Language*, *Update Language*, *Delete Language*, *Add Translation*, *Update Translation* and *Delete Translation* use cases, as presented in figure 3.6.

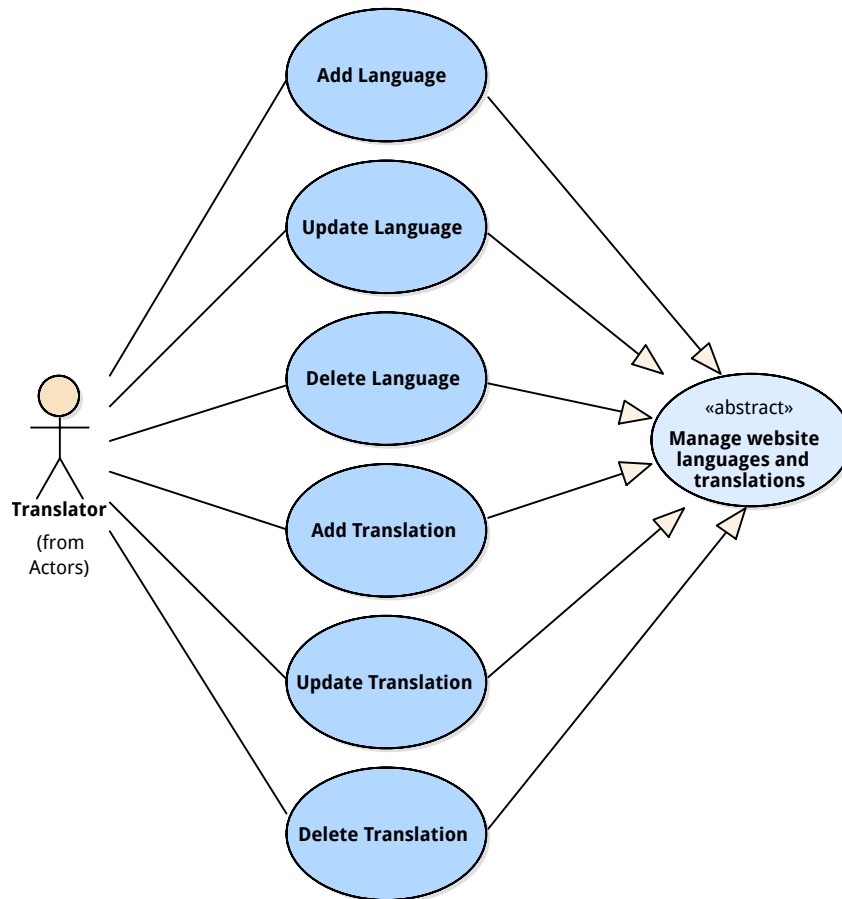


Figure 3.6: Manage website languages and translations use case diagram

The use cases are specified in more detail in the following subsections.

3.2.1 CREATE ACCOUNT

ID 1

Actors Guest

Brief Guest users register an account to be able to collaborate (e.g. submit identification requests, answer quizzes). The system sends a confirmation email to the user with a link for the user to activate the account, in an effort to reduce unwanted or malicious accounts. The user activates the account and is now a Registered User.

The create account activity diagram is shown in figure 3.11 on page 60.

3.2.2 LOGIN

ID 2

Actors Registered User

Brief A user with an account authenticates itself into the system to gain the registered user permissions (e.g. submit identification requests, answer quizzes)

The login activity diagram is shown in figure 3.12 on page 61.

3.2.3 SUBMIT IDENTIFICATION REQUEST

ID 3

Actors Registered User

Brief Registered users submit specimen identification requests, containing at least a photo and the location of the specimen, so that they get help from the community when other users vote on the cultivars for this specimen or answer quizzes where it appears

The submit identification request activity diagram is shown in figure 3.13 on page 62.

3.2.4 ANSWER QUIZ

ID 4

Actors Registered User

Brief Registered users answer quizzes, containing specimens to be identified, as a way to earn reputation and to help the community, as each answer represents a vote on one cultivar for that specimen. The system saves the votes and recalculates the user reputation and the specimens—that the user voted—probabilities

The answer quiz activity diagram is shown in figure 3.14 on page 63.

3.2.5 VIEW CULTIVAR

- ID** 5
- Actors** Guest & Registered User
- Brief** Users access the cultivar list, select a cultivar and view its details page containing a description, the species, and a list of standard specimens

The view cultivar activity diagram is shown in figure 3.15 on page 64.

3.2.6 SEARCH CULTIVAR BY NAME

- ID** 6
- Actors** Guest & Registered User
- Brief** Users access the cultivar list and enter a keyword on the filter field. The system shows the list of cultivars which names contain that keyword

The search cultivar by name activity diagram is shown in figure 3.16 on page 65.

3.2.7 VIEW SPECIMEN

- ID** 7
- Actors** Guest & Registered User
- Brief** Users access the specimens (identification requests) list and select a specimen to see its details page. The details page has the specimen gallery, location and all the characteristics

The view specimen activity diagram is shown in figure 3.17 on page 66.

3.2.8 VIEW SPECIMENS NEARBY

- ID** 8
- Actors** Guest & Registered User
- Brief** Users access the main page and view the nearby specimens on the map

The view specimens nearby activity diagram is shown in figure 3.18 on page 66.

3.2.9 VIEW PROFILE

ID 9

Actors Registered User

Brief Registered users access their profile page which contains information like their reputation, the number of sent identification requests, the number of answered quizzes and the number of voted cultivars

The view profile activity diagram is shown in figure 3.19 on page 67.

3.2.10 EDIT PROFILE

ID 10

Actors Registered User

Brief Registered users access their profile page and update their information (e.g. email address)

The edit profile activity diagram is shown in figure 3.20 on page 68.

3.2.11 RECEIVE NOTIFICATION BY EMAIL

ID 11

Actors Registered User

Brief Registered users receive notifications by email when the system detects a considerable change on the users identification requests answers (cultivar probability values or the number of votes)

The receive notification by email activity diagram is shown in figure 3.21 on page 69.

3.2.12 MANAGE WEBSITE LANGUAGES/TRANSLATIONS

This abstract use case is specialized by *Add Language*, *Update Language*, *Delete Language*, *Add Translation*, *Update Translation* and *Delete Translation* use cases, described below.

ID 12

Actors Translator

Brief Translators update the website languages and translations (e.g. add a new language, fix a translated phrase)

ADD LANGUAGE

ID 12.1

Actors Translator

Brief Translators access the “manage translations” page and add a new language

The add language activity diagram is shown in figure 3.22 on page 70.

UPDATE LANGUAGE

ID 12.2

Actors Translator

Brief Translators access the “manage translations” page and change an existing language

The update language activity diagram is shown in figure 3.23 on page 71.

DELETE LANGUAGE

ID 12.3

Actors Translator

Brief Translators access the “manage translations” page and delete a language that has no related translations. If the language has translations it can not be removed

The delete language activity diagram is shown in figure 3.24 on page 71.

ADD TRANSLATION

ID 12.4

Actors Translator

Brief Translators access the “manage translations”, select a language and add a missing translation

The add translation activity diagram is shown in figure 3.25 on page 72.

UPDATE TRANSLATION

ID 12.5

Actors Translator

Brief Translators access the “manage translations”, select a language and update an existing translation

The update translation activity diagram is shown in figure 3.26 on page 73.

DELETE TRANSLATION

ID 12.6

Actors Translator

Brief Translators access the “manage translations”, select a language and delete a translation

The delete translation activity diagram is shown in figure 3.27 on page 73.

3.2.13 MANAGE CULTIVAR REGISTER

This abstract use case is specialized by *Add Cultivar*, *Update Cultivar*, *Accept Suggested Cultivar* and *Delete Cultivar* use cases, described below.

ID 13

Actors Moderator

Brief Moderators update the cultivar register (e.g. add a new cultivar, accept a suggested cultivar, edit an existing cultivar)

ADD CULTIVAR

ID 13.1

Actors Moderator

Brief Moderators access the “manage cultivars” page and add a new cultivar

The add cultivar activity diagram is shown in figure 3.28 on page 74.

UPDATE CULTIVAR

ID 13.2

Actors Moderator

Brief Moderators access the “manage cultivars” page and edit an existing cultivar

The update cultivar activity diagram is shown in figure 3.29 on page 75.

ACCEPT SUGGESTED CULTIVAR

ID 13.3

Actors Moderator

Brief Moderators access the “manage cultivars” page, select a “pending” (suggested by a user) cultivar and accept it or reject it. If accepted the system makes the cultivar visible to everyone. If rejected it is marked as rejected and stays visible only to the moderators

The accept suggested cultivar activity diagram is shown in figure 3.30 on page 76.

DELETE CULTIVAR

ID 13.4

Actors Moderator

Brief Moderators access the “manage cultivars” page, select a cultivar and delete it. The system only allows the deletion if there are no related specimen, that is, the cultivar was not voted in a specimen

The delete cultivar activity diagram is shown in figure 3.31 on page 77.

3.2.14 VIEW USER PROFILES AND STATISTICS

ID 14

Actors Moderator

Brief Moderators access the “manage users”, select a user profile and view its statistics

The view user profiles and statistics activity diagram is shown in figure 3.32 on page 77.

3.2.15 APPROVE IDENTIFICATION REQUEST

ID 15

Actors Moderator

Brief New specimen identification requests, by default, need to be approved by a moderator before the other users can access it (specimens list or through quizzes). Moderators access the “manage specimens”, select a specimen and approves or rejects it. When approved, it is visible by anyone and appears on quizzes. If rejected it is marked as rejected as is only visible to the moderators

The approve identification request activity diagram is shown in figure 3.33 on page 78.

3.2.16 SET MEMBER AUTO-APPROVAL OF IDENTIFICATION REQUESTS

ID 16

Actors Moderator

Brief Moderators access a user profile and selects “grant confidence vote” (or “revoke confidence vote”). When granted, that user identification requests are auto-approved when submitted. When revoked it returns to the default behaviour and the identification requests will have to be approved by a moderator before being visible by everyone

The set member auto-approval of identification requests activity diagram is shown in figure 3.34 on page 79.

3.2.17 MANAGE USERS

This abstract use case is specialized by *Create User*, *Update User Profile*, *Update User Role* and *Delete User* use cases, described below.

ID 17

Actors Administrator

Brief Administrators access the “manage users” page and create a user, update a user role (translator, moderator, administrator), update a user profile, etc.

CREATE USER

ID 17.1

Actors Administrator

Brief Administrators access the “manage users” page and create a user by filling the details (e.g. user name, email, password). There will be no email sent to the chosen email address and that user account will be just activated

The create user activity diagram is shown in figure 3.35 on page 80.

UPDATE USER PROFILE

ID 17.2

Actors Administrator

Brief Administrators access the “manage users” page, select a user and update its profile (e.g. email address, name)

The update user profile activity diagram is shown in figure 3.36 on page 81.

UPDATE USER ROLE

ID 17.3

Actors Administrator

Brief Administrators access the “manage users” page, select a user and update its role (e.g. translator, moderator, administrator)

The update user role activity diagram is shown in figure 3.37 on page 82.

DELETE USER

ID 17.4

Actors Administrator

Brief Administrators access the “manage users” page, select a user and remove the user from the system. The user identification requests become hidden and the votes are removed, leading to a cultivar probability recalculation

The delete user activity diagram is shown in figure 3.38 on page 83.

3.2.18 DEFINE QUIZ PARAMETERS

ID 18

Actors Administrator

Brief Administrators access the “quiz settings” at the administration page and define how many specimens will appear on the quizzes and how many of them are a standard specimen. There has to be at least one standard specimen, so that the reputation calculations may be used

The define quiz parameters activity diagram is shown in figure 3.39 on page 84.

3.2.19 SUGGEST NEW CULTIVAR

ID 19

Actors Registered User

Brief Registered users access the cultivars page and select “suggest a new cultivar” if they can’t find it on the existing registry. They fill the specimen name and description and then the suggestion is sent to approval by a moderator prior to joining the registry and being available to everyone

The suggest new cultivar activity diagram is shown in figure 3.40 on page 85.

3.2.20 VOTE CULTIVAR FOR A SPECIMEN

ID 20

Actors Registered User

Brief Registered users access a specimen page, select “vote on a cultivar” and select a cultivar from the ones that already have votes or a new cultivar from the list of cultivars that is presented

The vote cultivar for a specimen activity diagram is shown in figure 3.41 on page 86.

3.3 SUPPLEMENTARY SPECIFICATIONS

3.3.1 SPECIMEN DATABASE

The specimen database must have the following information:

- Location of the garden (address and **Global Positioning System (GPS)** coordinates);
- Location within the garden (identifying label or map-based pointer); Owner;
- Documented historic data (e.g. planting date, origin);
- Photographic documentation;
- Cultivar identification and associated probability (normally, dynamic fields worked out by the system);
- Characteristics according to **International Union for the Protection of New Varieties of Plants (UPOV)** guidelines (details below).

A set of 50 characteristics (49 morphological features and time of flowering) are considered, in accordance with the guidelines recently issued by the **UPOV** for conducting distinctness, uniformity and stability tests in the specific case of ornamental camellia varieties [25], based on work by Jiyuan Li et al. [26]. The morphological features (examples in figure 3.7 on the next page and in figure 3.8 on page 56) regard the plant as a whole (growth habit), its branches, foliage, vegetative buds, shoots, leaves, petioles, sepals, flower buds, flowers, petals, stamens, style, stigma and ovary.

Although the database must allow the storage of very complete and detailed specimen information, the identification request form will be kept simple and intuitive, with a minimum number of mandatory fields – possibly just the precise location of the specimen and a photo of a flower meeting certain minimum technical criteria in terms of colour, resolution and format, so it can be included in quizzes. The remaining fields will be optional, and users will be allowed to update in their identification requests at any point, so as not to discourage submission.

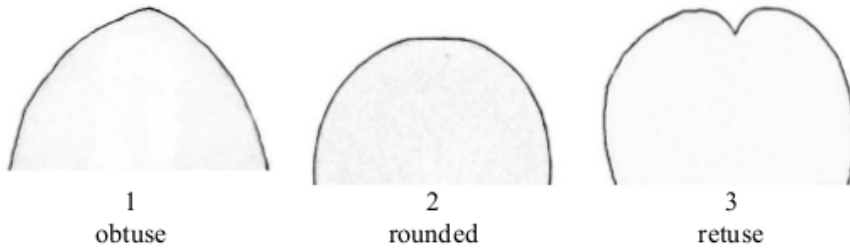
TG/275/1
Camellia/Camélia/Kamelie/Camelia, 2011-10-20
-16-

	English	français	deutsch	español	Example Varieties/ Exemples/ Beispielssorten/ Variedades ejemplo	Note/ Nota
27. VG (+)	Sepal: shape of apex (d)	Sépale : forme de la pointe	Kelchblatt: Form der Spitze	Sépalo: forma del ápice		
QN	obtuse	obtuse	stumpf	obtuso		1
	rounded	ronde	abgerundet	redondeado		2
	retuse	échancrée	eingedrückt	retuso		3
28. VG (*) (+)	Flower bud: arrangement	Bourgeon floral : disposition	Blütenknospe: Anordnung	Botón floral: disposición		
	terminal only	seulement terminale	nur terminal	sólo terminal		1
PQ	terminal and axillary	terminale et axillaire	terminal und axillar	terminal y axilar		2
	axillary only	seulement axillaire	nur axillar	sólo axilar		3
29. MG/ (*) VG	Flower: diameter (e)	Fleur : diamètre	Blüte: Durchmesser	Flor: diámetro		
QN	very small	très petit	sehr klein	muy pequeño	Wei Hua Lian Rui Cha	1
	small	petit	klein	pequeño	Xiao Mei Gui	3
	medium	moyen	mittel	medio	Hong Lu Zhen	5
	large	grand	groß	grande	Da Li Cha	7
	very large	très grand	sehr groß	muy grande	Fen Bao Jing Cha	9
30. VG (*) (+)	Flower: form (e)	Fleur : forme	Blüte: Typ	Flor: forma		
PQ	single	simple	einfach	simple	Da Hong Jin Xin	1
	semi-double	demi-double	halbgefüllt	semidoble	Chun Jiang Zhi Xia	2
	anemone form	en forme d'anémone	anemonenförmig	en forma de anémona	Jin Pan Li Zhi	3
	peony form	en forme de pivoine	päonienförmig	en forma de peonía	Hua Mu Dan	4
	rose form double	double en forme de rose	rosenförmig gefüllt	en forma de rosa doble	Zhuang Yuan Hong	5
	formal double	double imbriquée	vollständig gefüllt	doble formal	Xue Ta	6

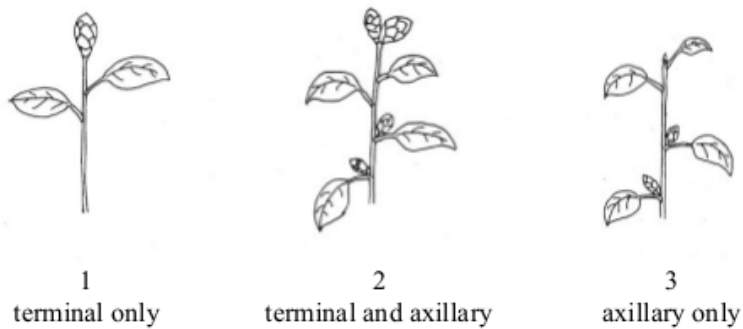
Figure 3.7: Example 1 of morphological features (UPOV guidelines[25])

TG/275/1
Camellia, 2011-10-20
-26-

Ad. 27: Sepal: shape of apex



Ad. 28: Flower bud: arrangement



Ad. 30: Flower: form

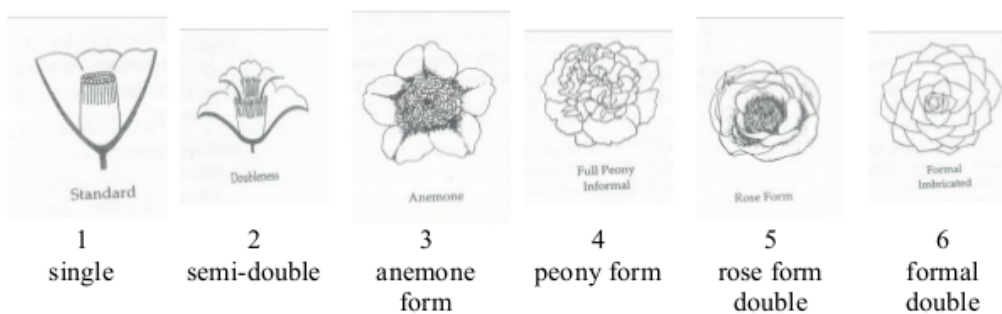


Figure 3.8: Example 2 of morphological features (UPOV guidelines[25])

3.3.2 STANDARD SPECIMENS

If the cultivar represented by a specimen is known with absolute certainty, a moderator user (or group of moderator users) with permission to do so can give that specimen the status of standard specimen; its characteristics then become automatically associated to the corresponding entry in the cultivar register. The answer to quizzes using standard specimens is objective; so long as there are enough of them in the system, reputation metrics can rely solely on objective data. Since the contribution of each response to an identification request is weighed according to the reputation of the respondent, this makes the system more reliable. In other words, if an effort is made to insert, as early as possible, a large number of specimens whose cultivar is known with absolute certainty, the operation of the system can be improved, with identifications converging more rapidly to the right answer. Moreover, for identification requests with at least some characteristics filled-in, and so long as the characteristics of the standard specimens are fully specified, the system can automatically assist the identification process by narrowing down the range of possible answers and detecting perfect matches, should they occur (in which case the specimen in question would be an obvious candidate to standard specimen).

3.3.3 USER REPUTATION AND CULTIVAR PROBABILITY METRICS

The reputation of the respondents is a figure of merit based on their quiz performance. Basing it only on questions referring to standard specimens, it can be computed very simply as the percentage of right answers. It is fair to assume that a user with higher reputation is more likely to correctly identify any given specimen. Therefore, in working out cultivar probability, i.e. the probability of a specimen belonging to a certain cultivar, the respondent's reputation is the weighing factor applied to her response. These two figures are not static, in the sense that they are recalculated whenever new quizzes are considered. Alternative, more complex reputation metrics can be envisaged to avoid reliance on standard specimens.

3.4 DOMAIN MODEL

After the identification of the use cases for the system (in section 3.2 on page 41), the main entities—resulting from the requirements analysis—will now be described in a domain model. In this phase, the application domain is analysed to discover the domain requirements of the system and create a high-level object model that describes how the system will be logically constructed. The logical implementation of the functional requirements described in the use case model is captured by a high-level class diagram in figure 3.9 on the following

page, reflecting initial domain knowledge.

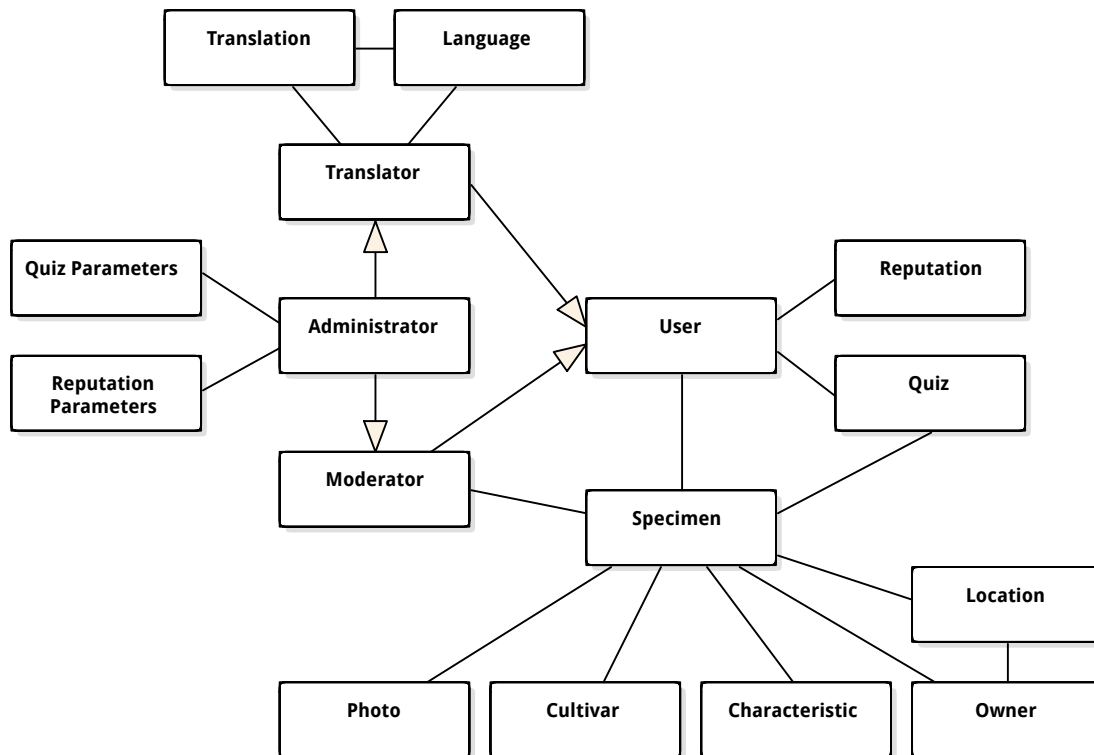


Figure 3.9: High-level system class diagram

The diagram shows that an Administrator is both a Moderator and a Translator, and they are both a User. An Administrator sets Quiz Parameters (how many specimens and how many of them are standard) and Reputation Parameters (the weight given to the standard specimens answers and the weight given to the number of user votes). A Translator manages Translations and Languages and each Language has Translations. A User has Reputation, submits Specimen requests (that are approved by a Moderator), votes Specimen cultivars, and answers Quizzes (that has Specimens). A Specimen has a Location, an Owner (that has a Location), Photos, Characteristics, may have possible Cultivars and appears on Quizzes.

Figure 3.10 on the next page presents a more detailed class diagram. Here, the system actors are turned into roles that a user may have. The main user information (e.g. username, password, email address) is stored in the User class, whereas the additional information (e.g. profile photo, reputation) is stored in the *User Profile* class, for the sake of organization. The *User Message* is used to save the messages the users send to the site owners (e.g. to report a problem or to suggest something). It is not related with the User class because it is available to the visitors that do not have an account. The translation tables is described in more detail in this diagram. For the sake of organization, each UPOV Characteristic table (type, name, value) has a related translation table. This choice of design is discussed ahead in section 4.7

on page 126. Most of the classes are self-explanatory and will be covered in chapter 4 on page 87 (Implementation).

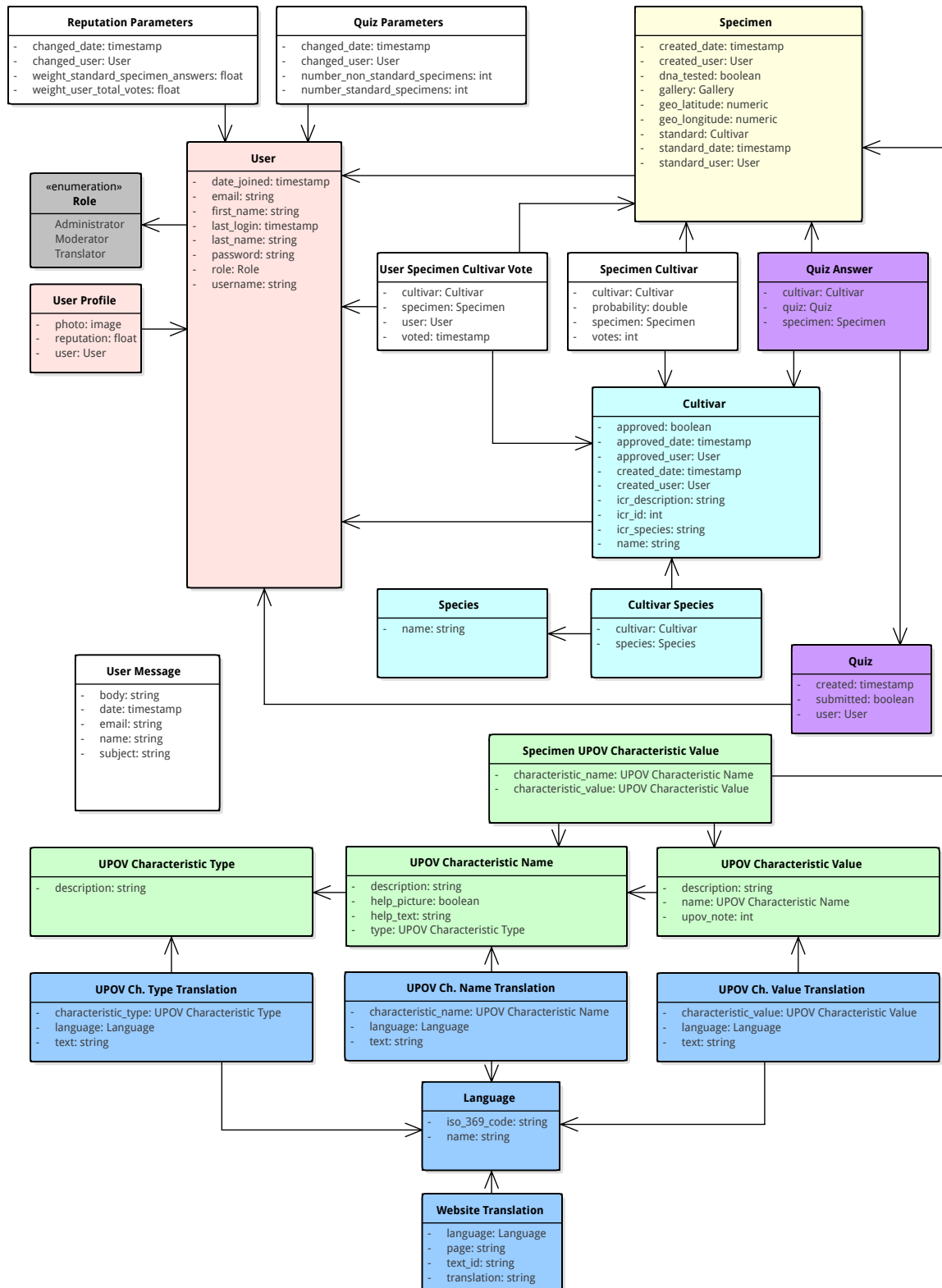


Figure 3.10: System class diagram

3.5 ACTIVITY DIAGRAMS

This section presents the activity diagrams for modeling the logic captured by the use cases discussed in section 3.2 on page 41.

3.5.1 CREATE ACCOUNT

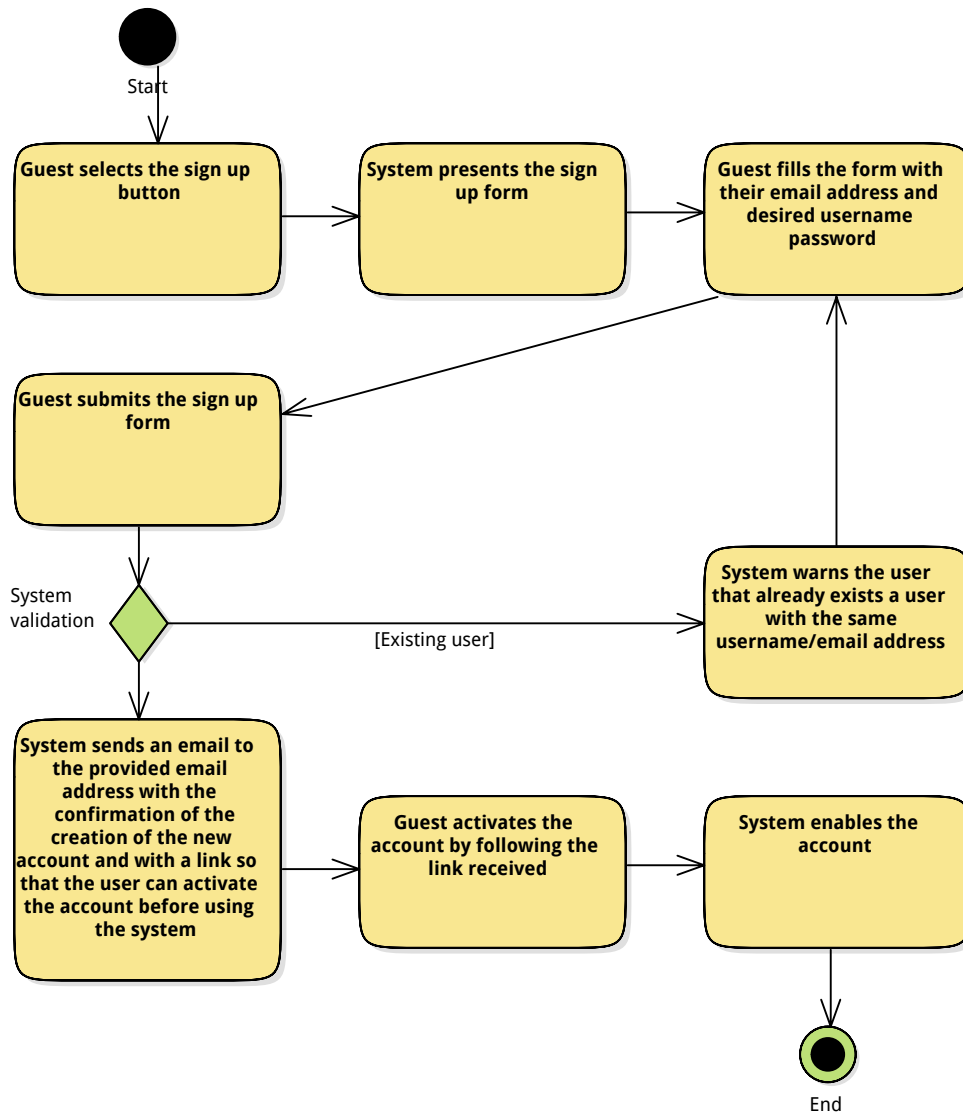


Figure 3.11: Activity diagram for the use case create account

3.5.2 LOGIN

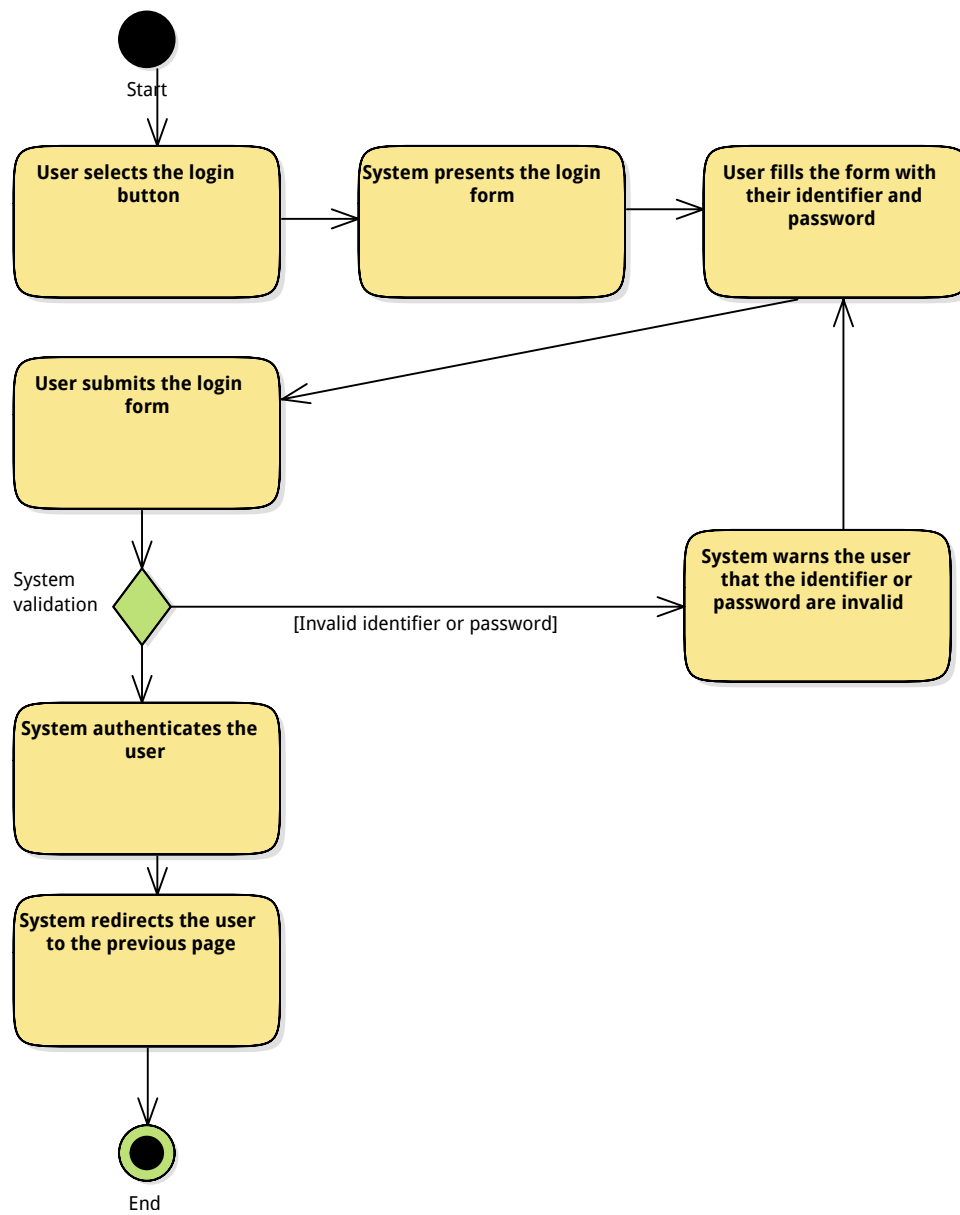


Figure 3.12: Activity diagram for the use case login

3.5.3 SUBMIT IDENTIFICATION REQUEST

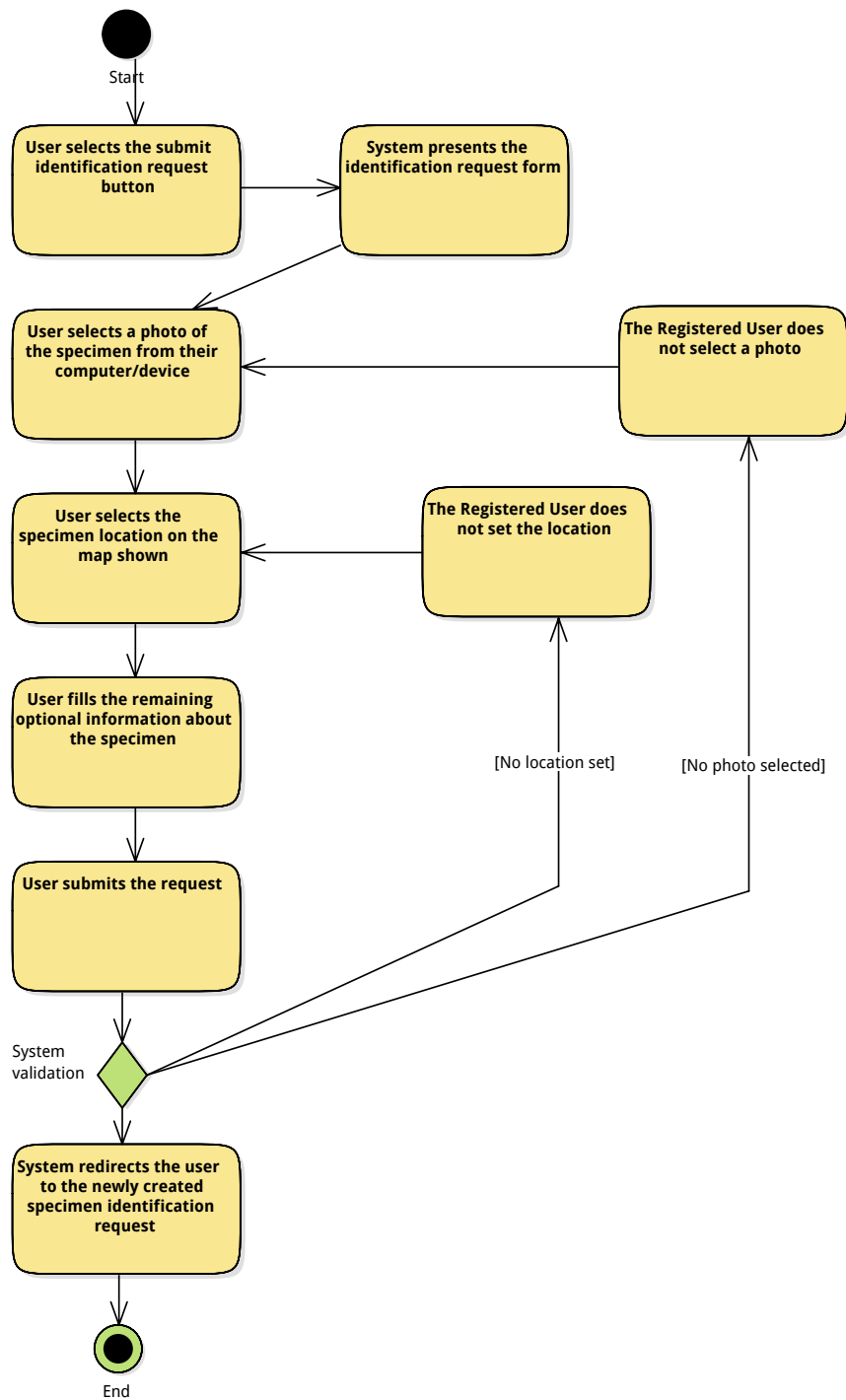


Figure 3.13: Activity diagram for the use case submit identification request

3.5.4 ANSWER QUIZ

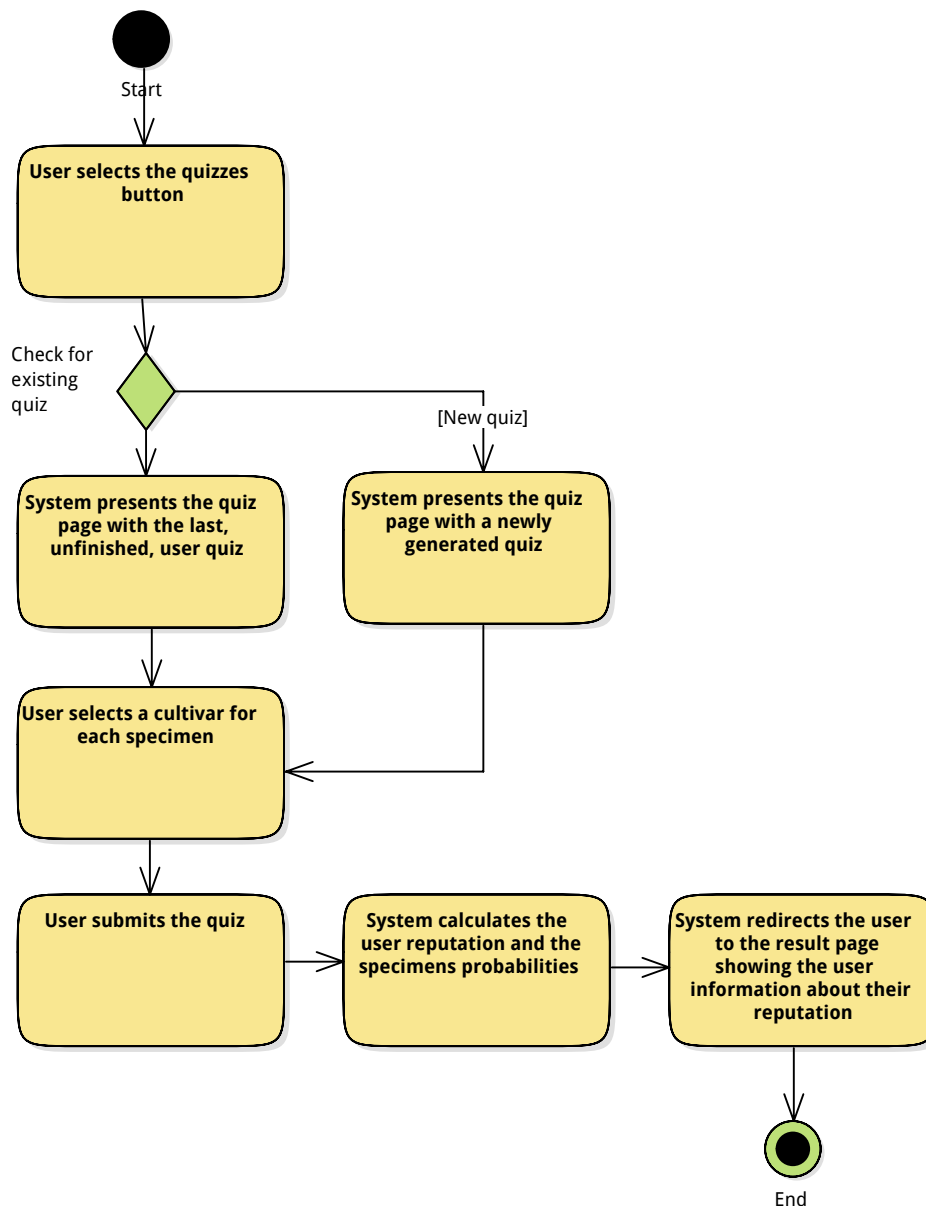


Figure 3.14: Activity diagram for the use case answer quiz

3.5.5 VIEW CULTIVAR

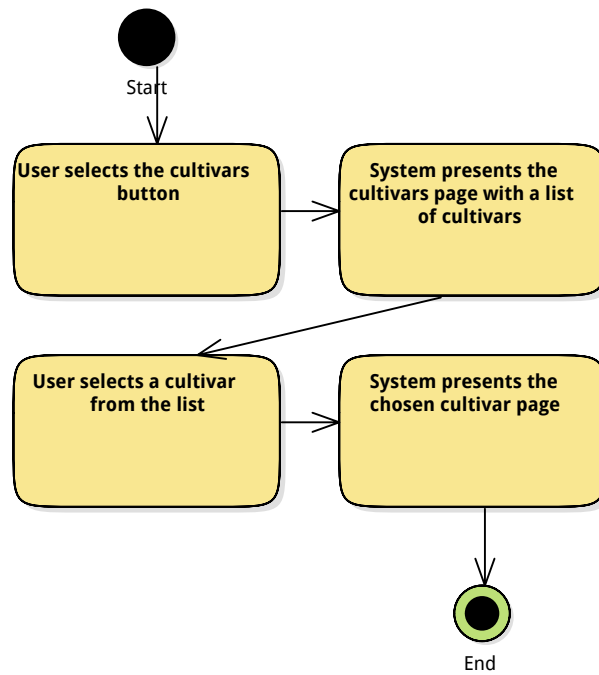


Figure 3.15: Activity diagram for the use case view cultivar

3.5.6 SEARCH CULTIVAR BY NAME

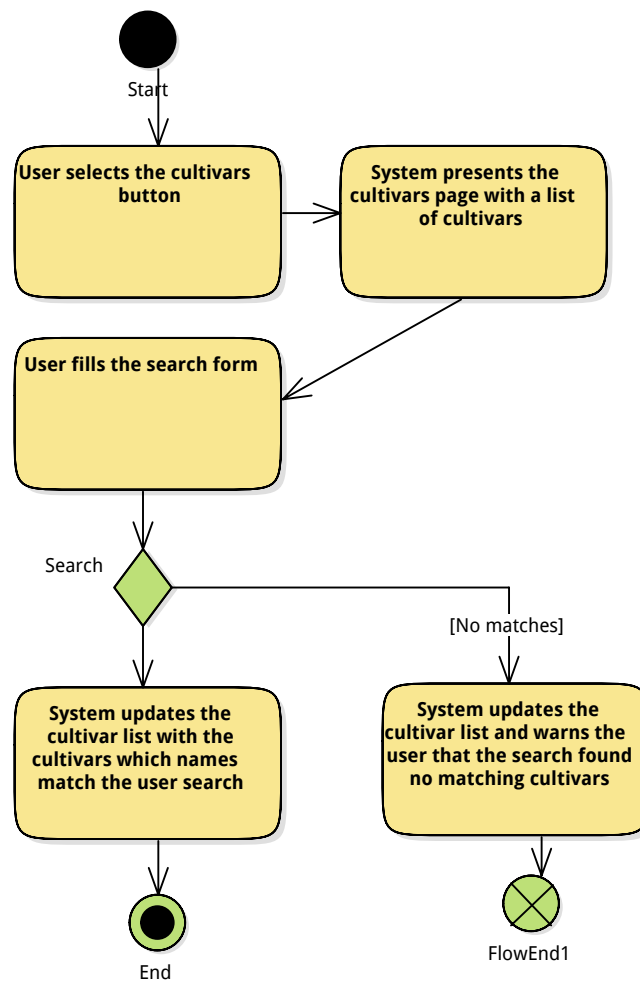


Figure 3.16: Activity diagram for the use case search cultivar by name

3.5.7 VIEW SPECIMEN

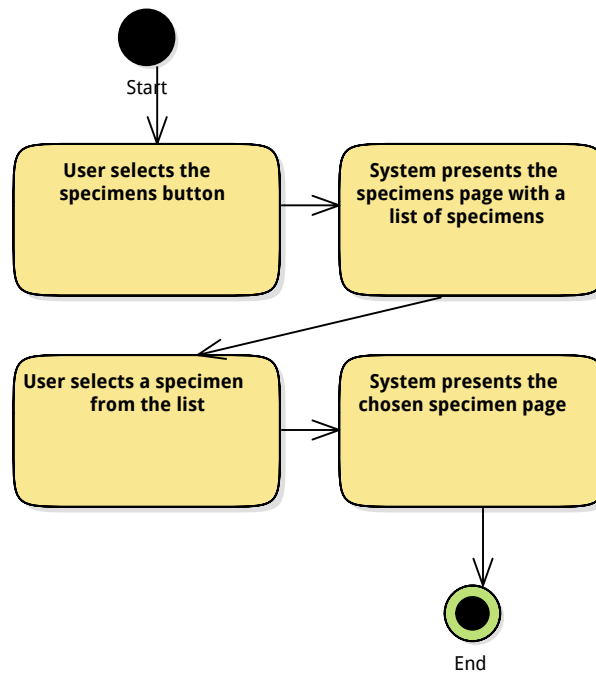


Figure 3.17: Activity diagram for the use case view specimen

3.5.8 VIEW SPECIMENS NEARBY

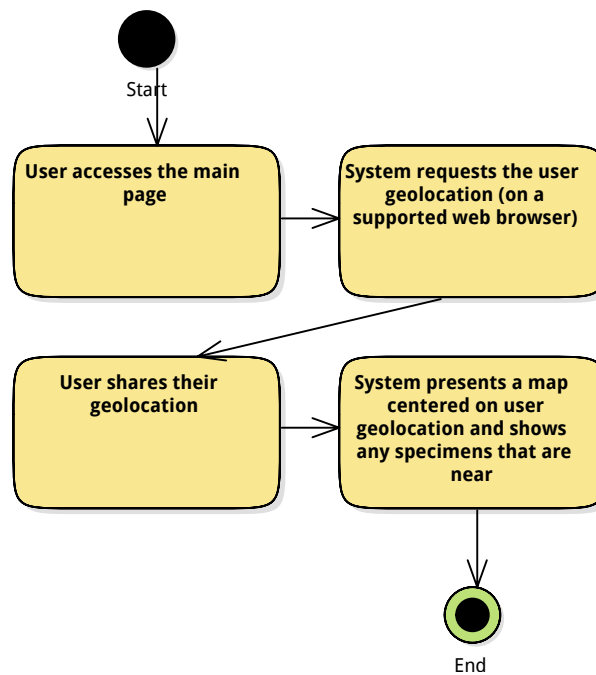


Figure 3.18: Activity diagram for the use case view specimens nearby

3.5.9 VIEW PROFILE

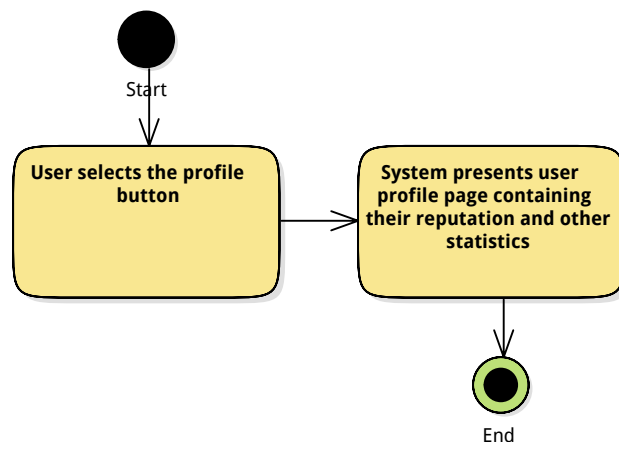


Figure 3.19: Activity diagram for the use case view profile

3.5.10 EDIT PROFILE

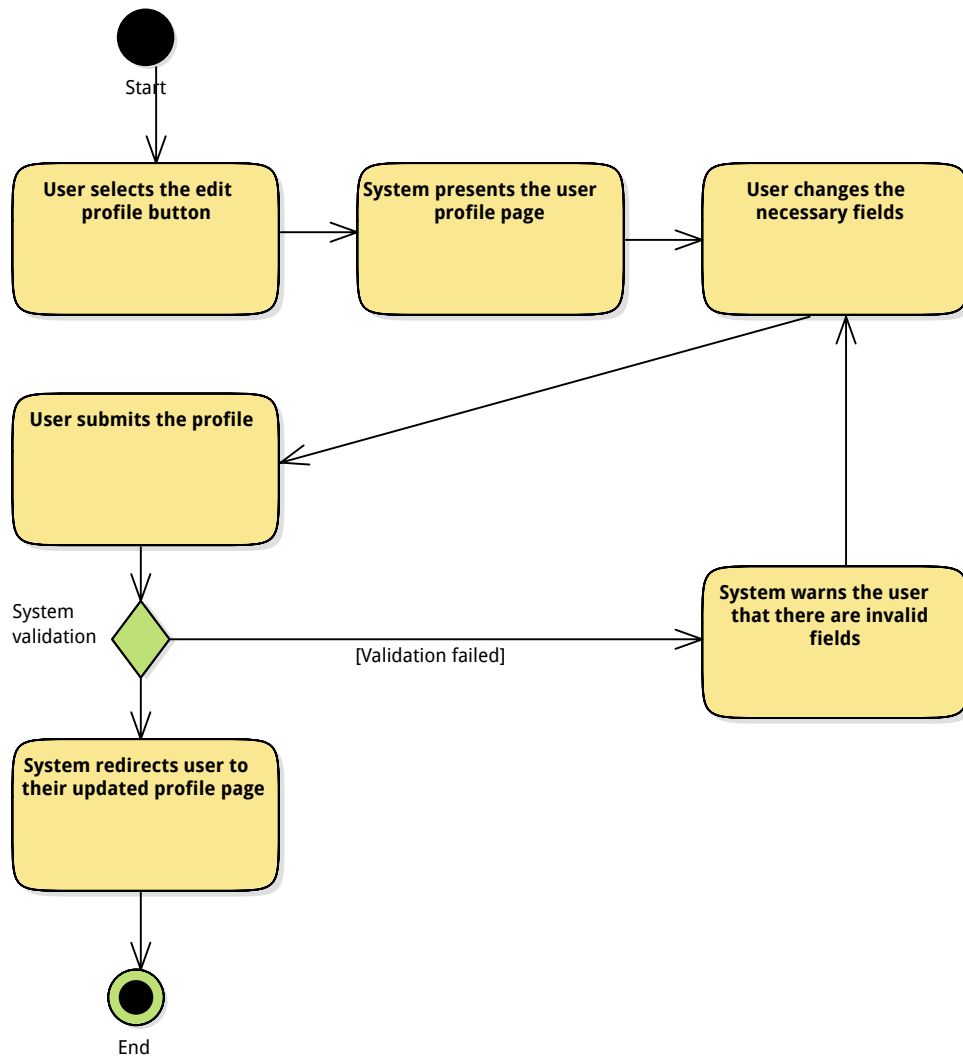


Figure 3.20: Activity diagram for the use case edit profile

3.5.11 RECEIVE NOTIFICATION BY EMAIL

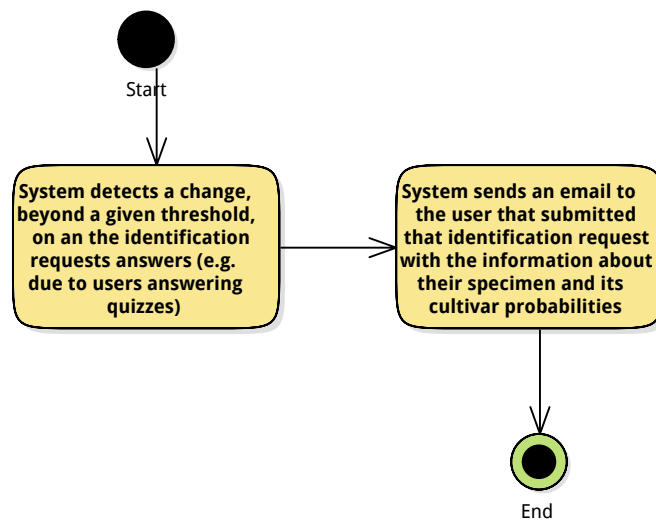


Figure 3.21: Activity diagram for the use case receive notification by email

3.5.12 MANAGE WEBSITE LANGUAGES/TRANSLATIONS

ADD LANGUAGE

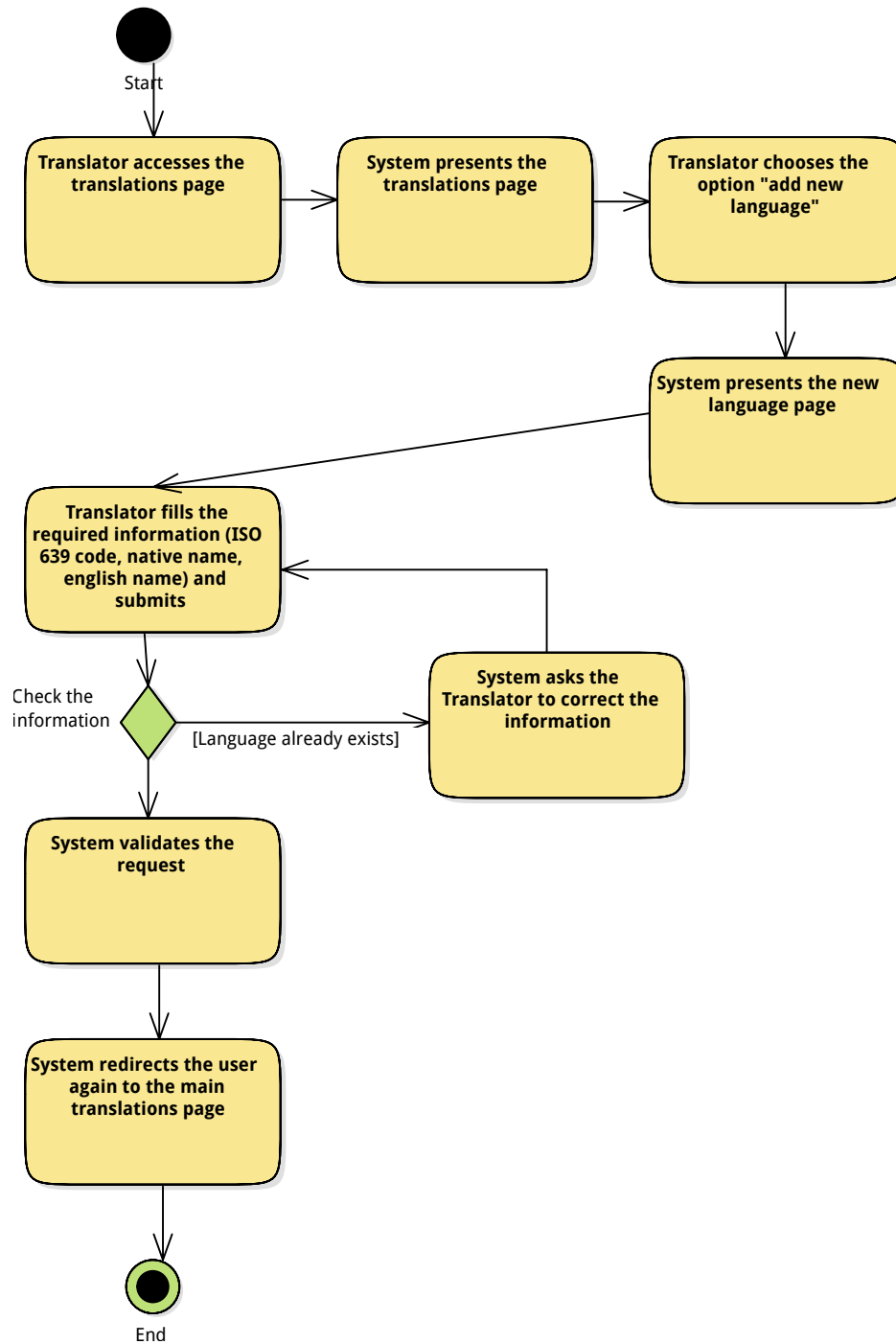


Figure 3.22: Activity diagram for the use case add language

UPDATE LANGUAGE

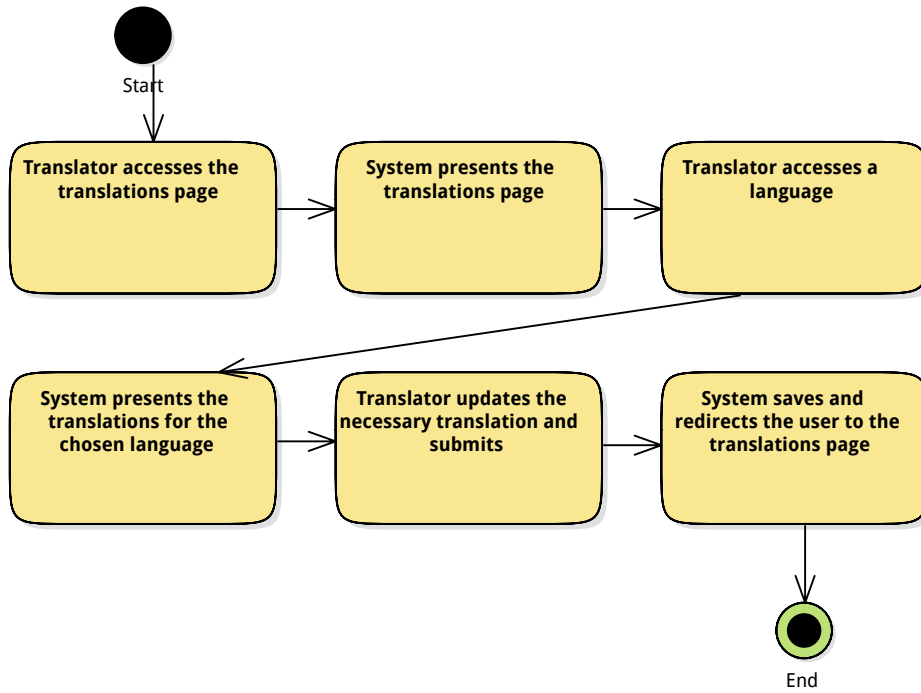


Figure 3.23: Activity diagram for the use case update language

DELETE LANGUAGE

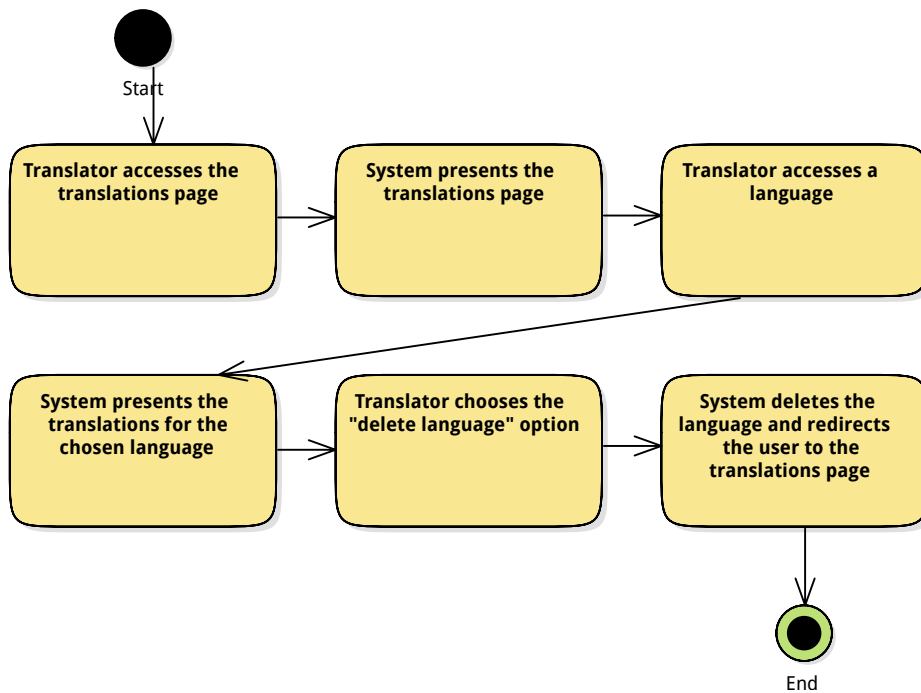


Figure 3.24: Activity diagram for the use case delete language

ADD TRANSLATION

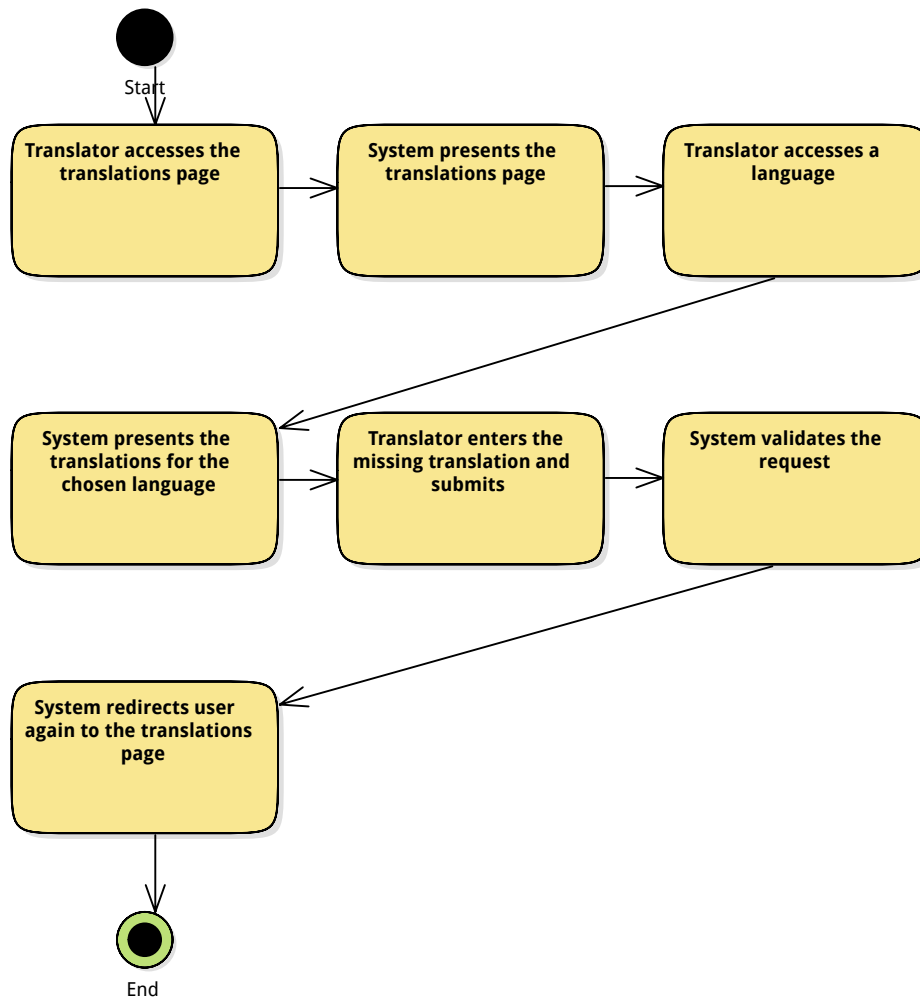


Figure 3.25: Activity diagram for the use case add translation

UPDATE TRANSLATION

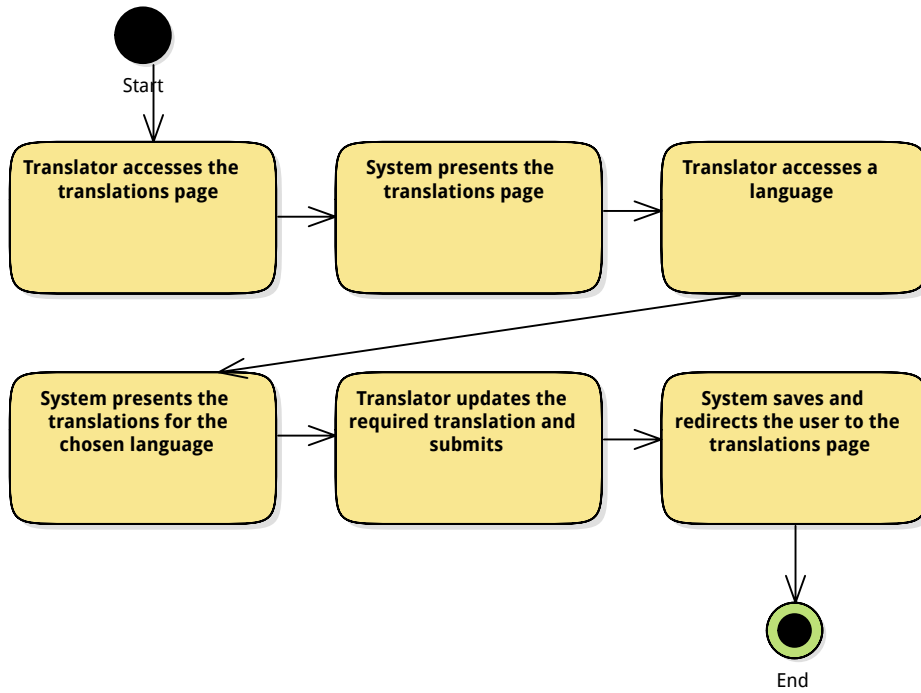


Figure 3.26: Activity diagram for the use case update translation

DELETE TRANSLATION

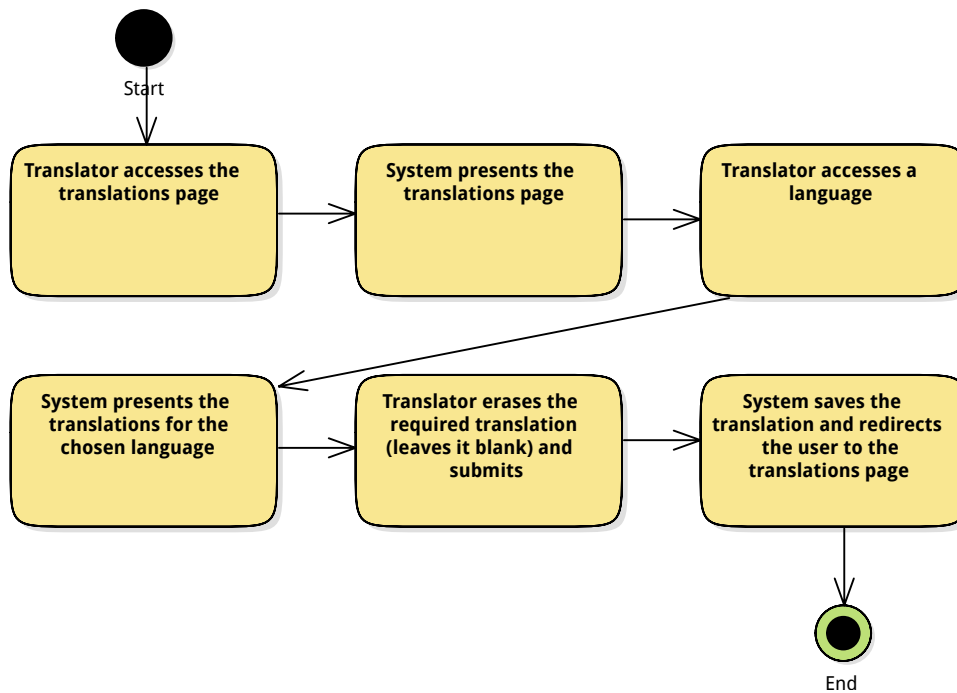


Figure 3.27: Activity diagram for the use case delete translation

3.5.13 MANAGE CULTIVAR REGISTER

ADD CULTIVAR

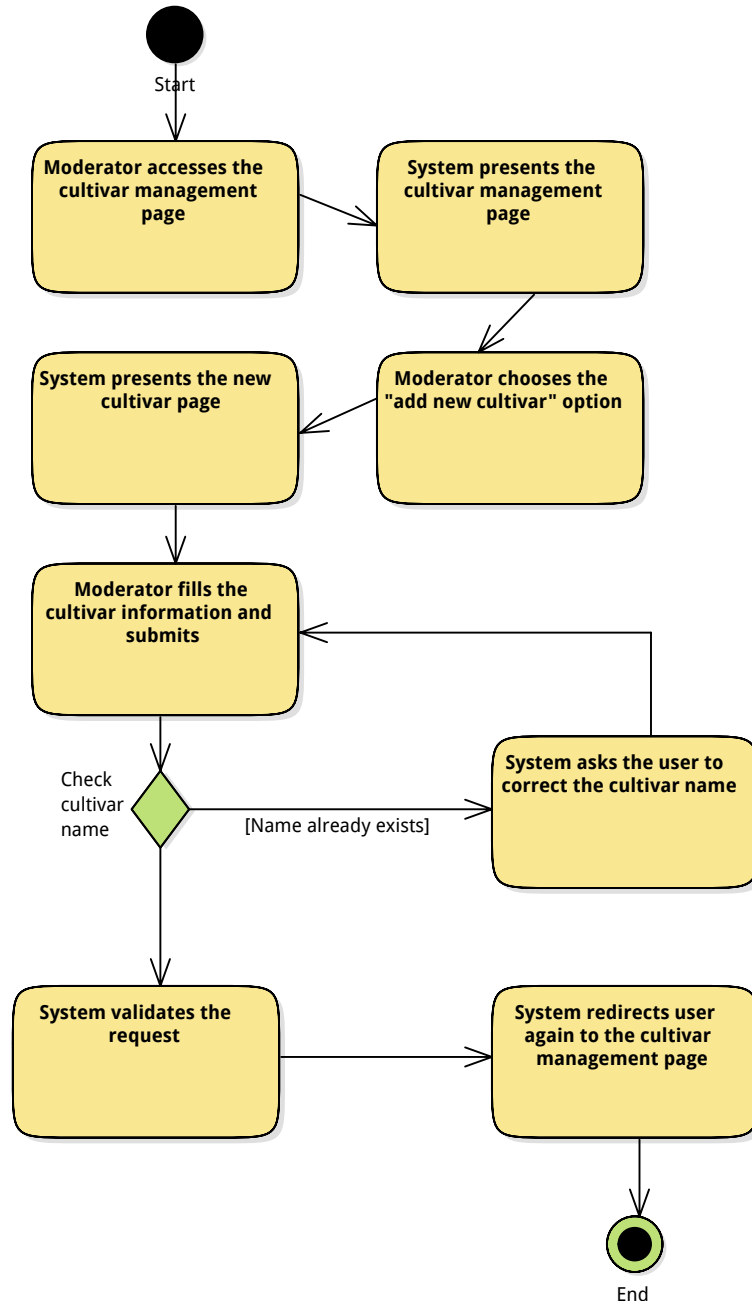


Figure 3.28: Activity diagram for the use case add cultivar

UPDATE CULTIVAR

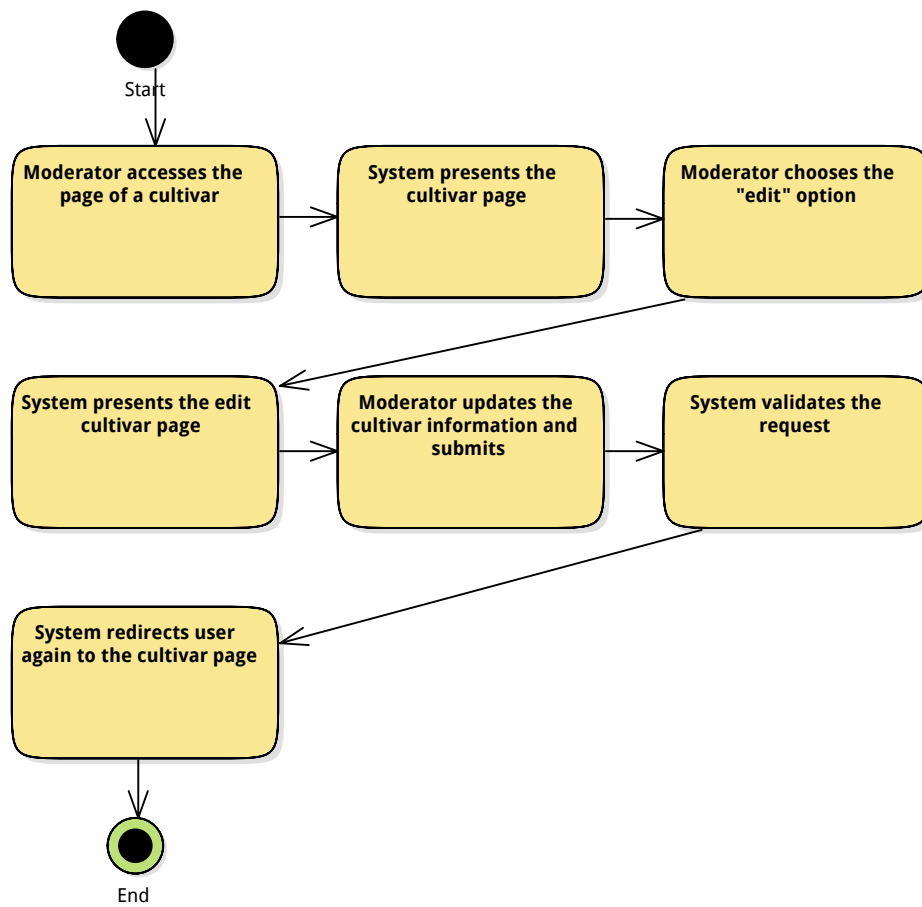


Figure 3.29: Activity diagram for the use case update cultivar

ACCEPT SUGGESTED CULTIVAR

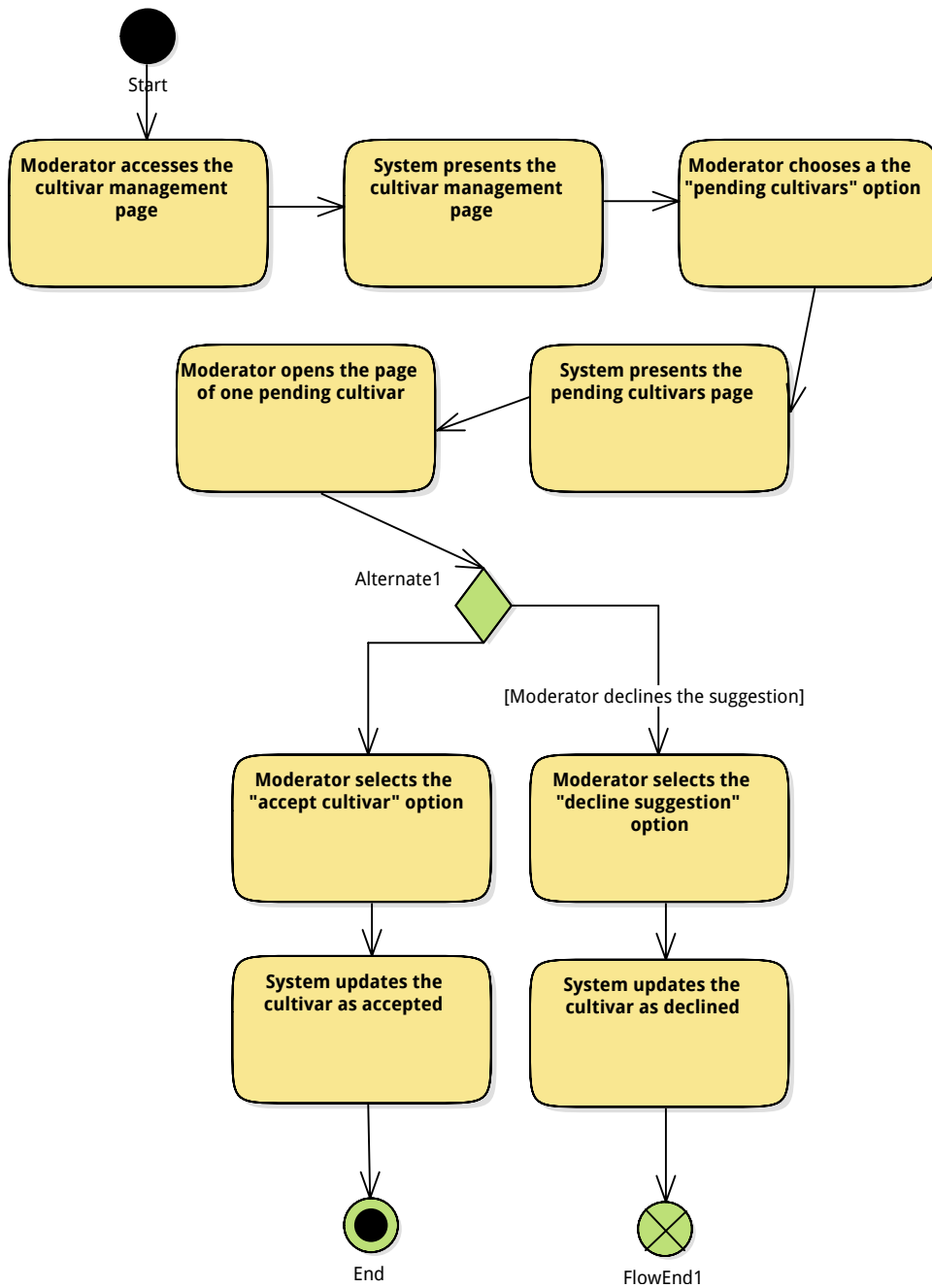


Figure 3.30: Activity diagram for the use case accept suggested cultivar

DELETE CULTIVAR

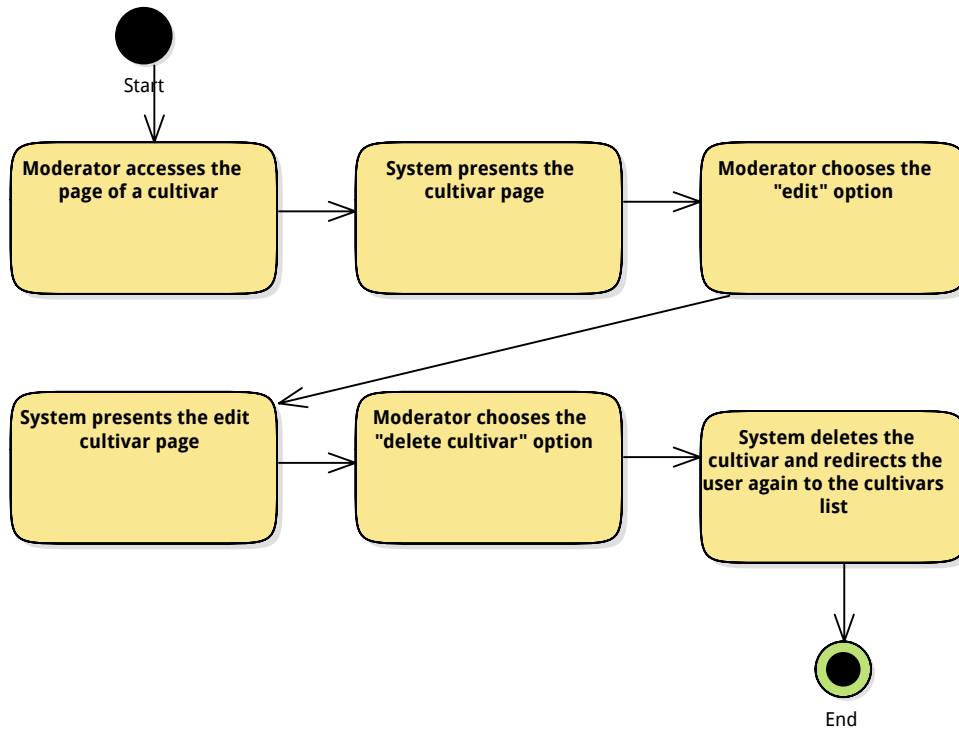


Figure 3.31: Activity diagram for the use case delete cultivar

3.5.14 VIEW USER PROFILES AND STATISTICS

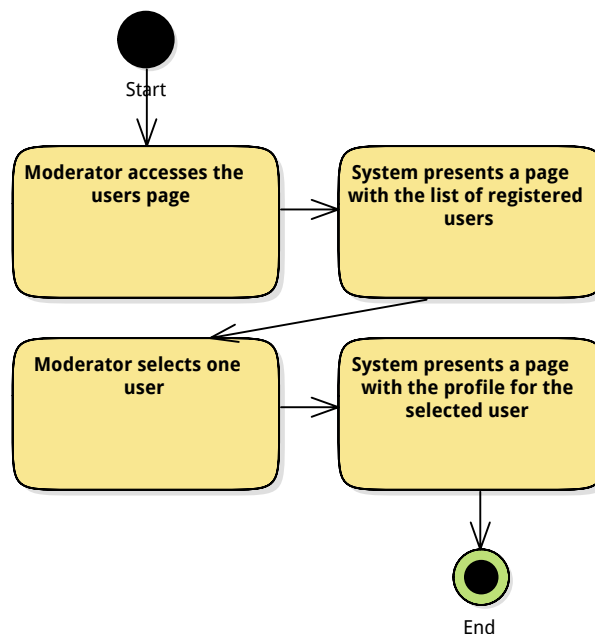


Figure 3.32: Activity diagram for the use case view user profiles and statistics

3.5.15 APPROVE IDENTIFICATION REQUEST

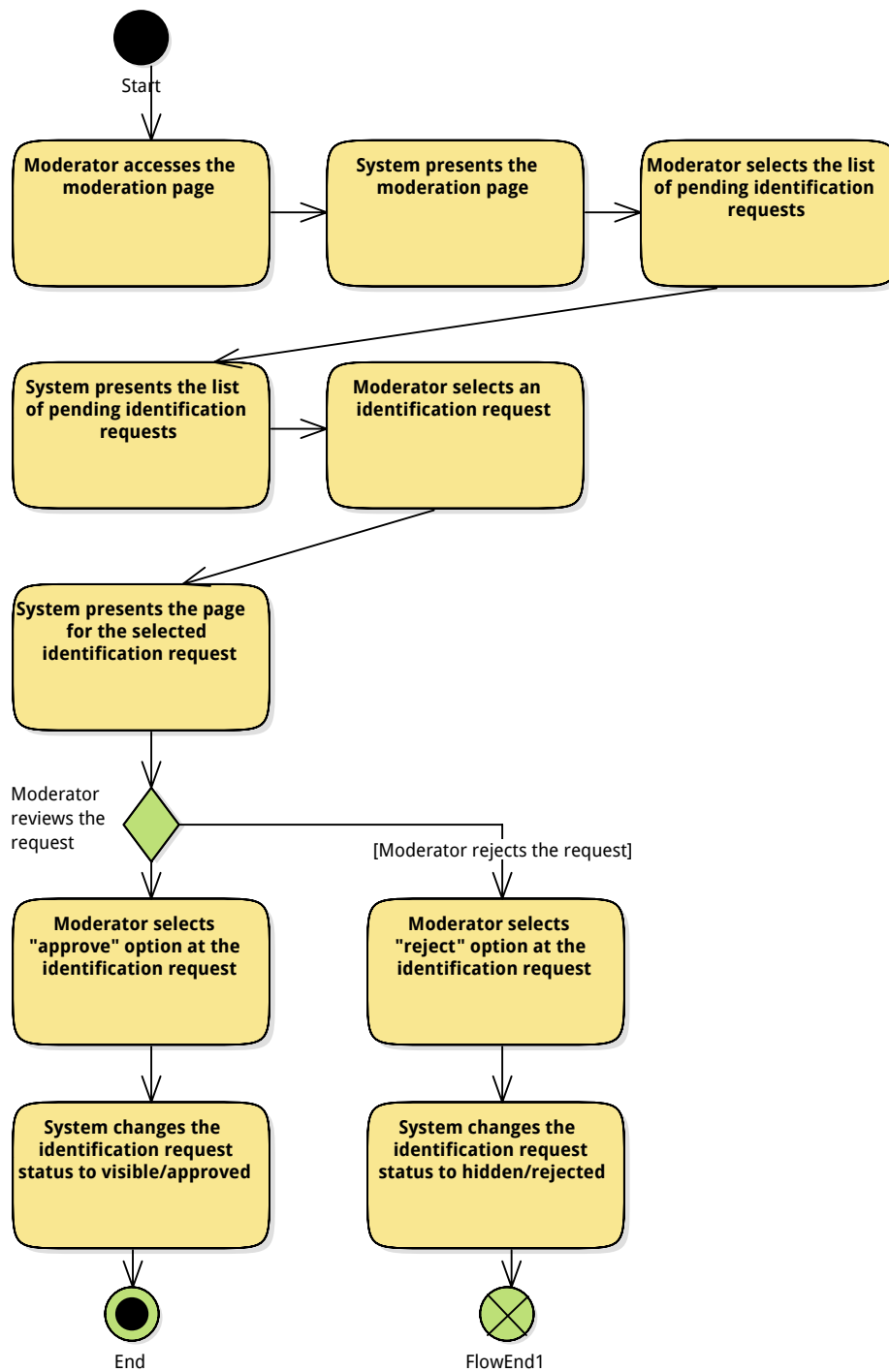


Figure 3.33: Activity diagram for the use case approve identification request

3.5.16 SET MEMBER AUTO-APPROVAL OF IDENTIFICATION REQUESTS

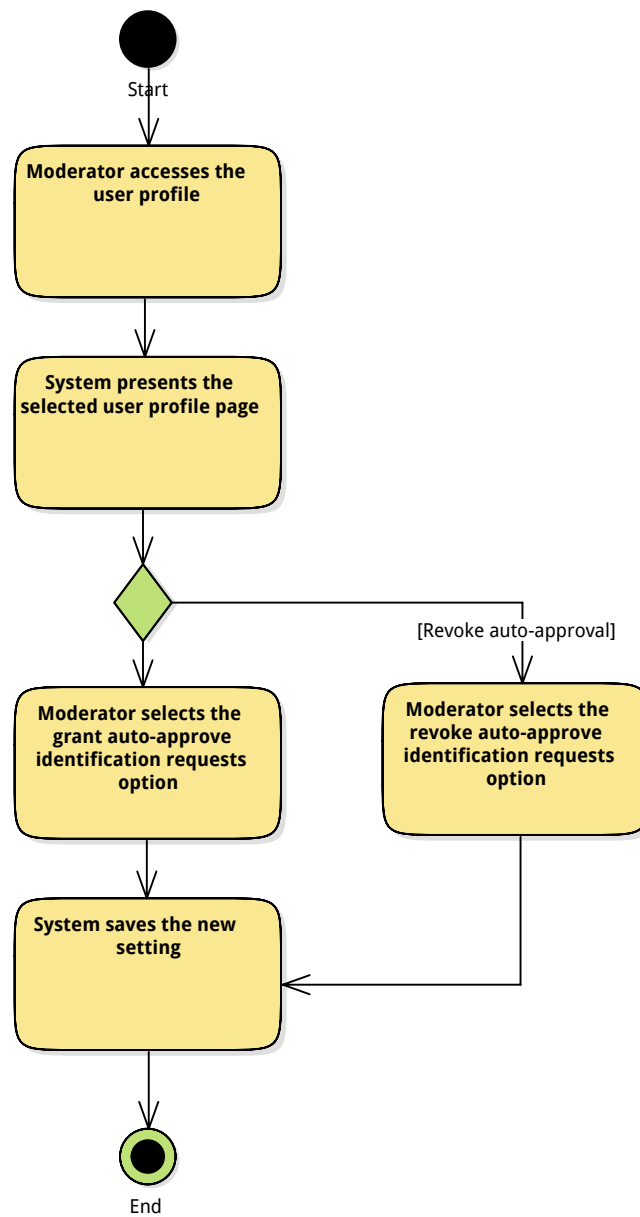


Figure 3.34: Activity diagram for the use case set member auto-approval of identification requests

3.5.17 MANAGE USERS

CREATE USER

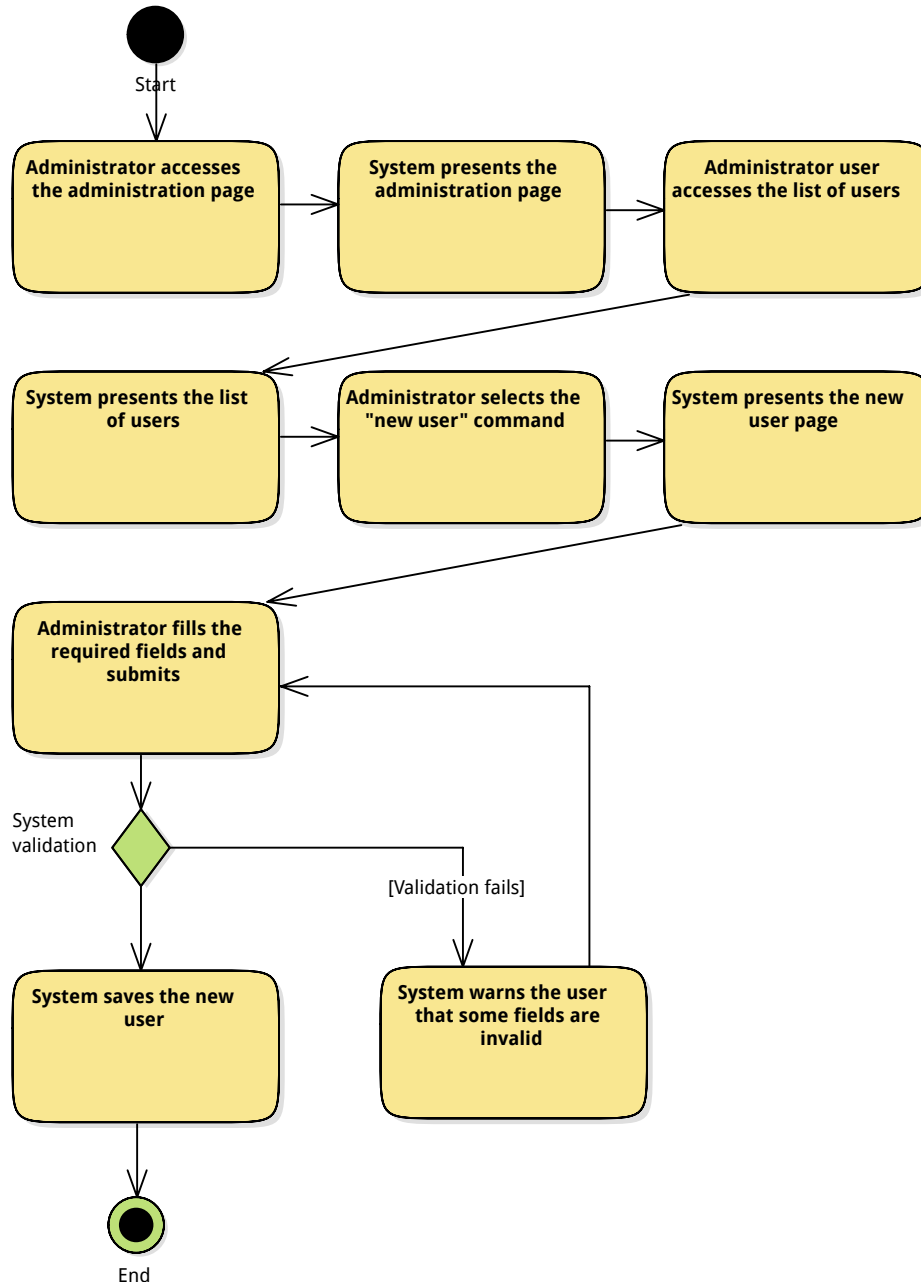


Figure 3.35: Activity diagram for the use case create user

UPDATE USER PROFILE

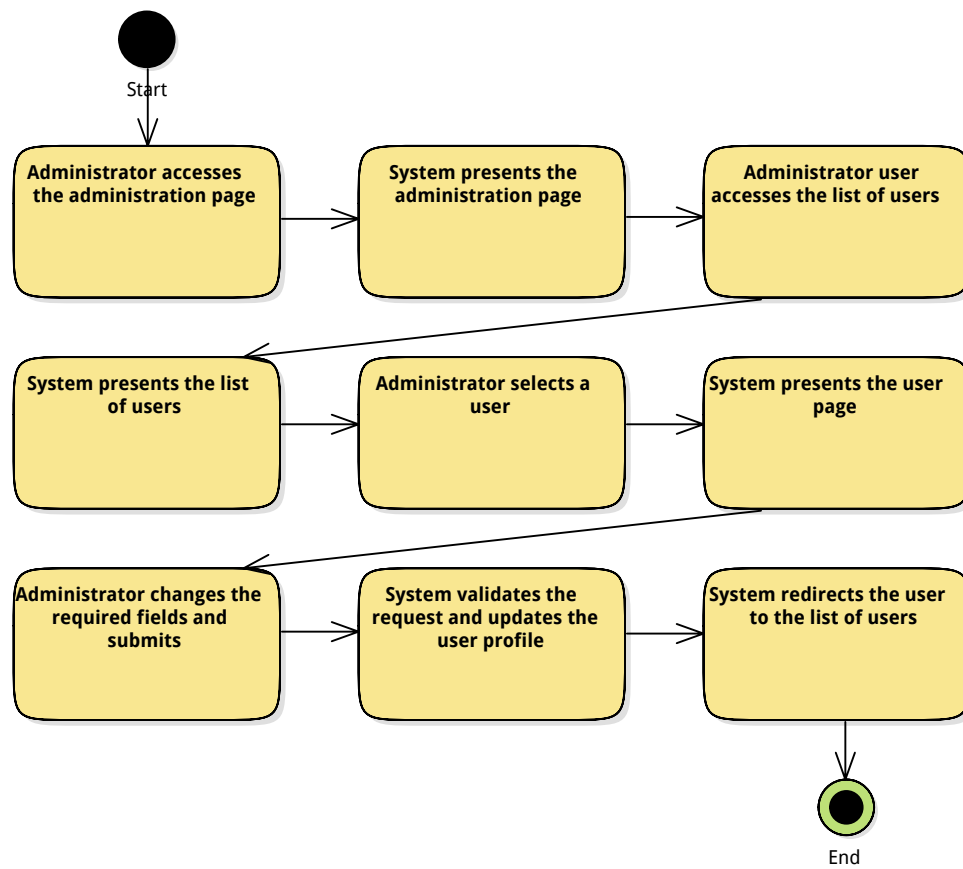


Figure 3.36: Activity diagram for the use case update user profile

UPDATE USER ROLE

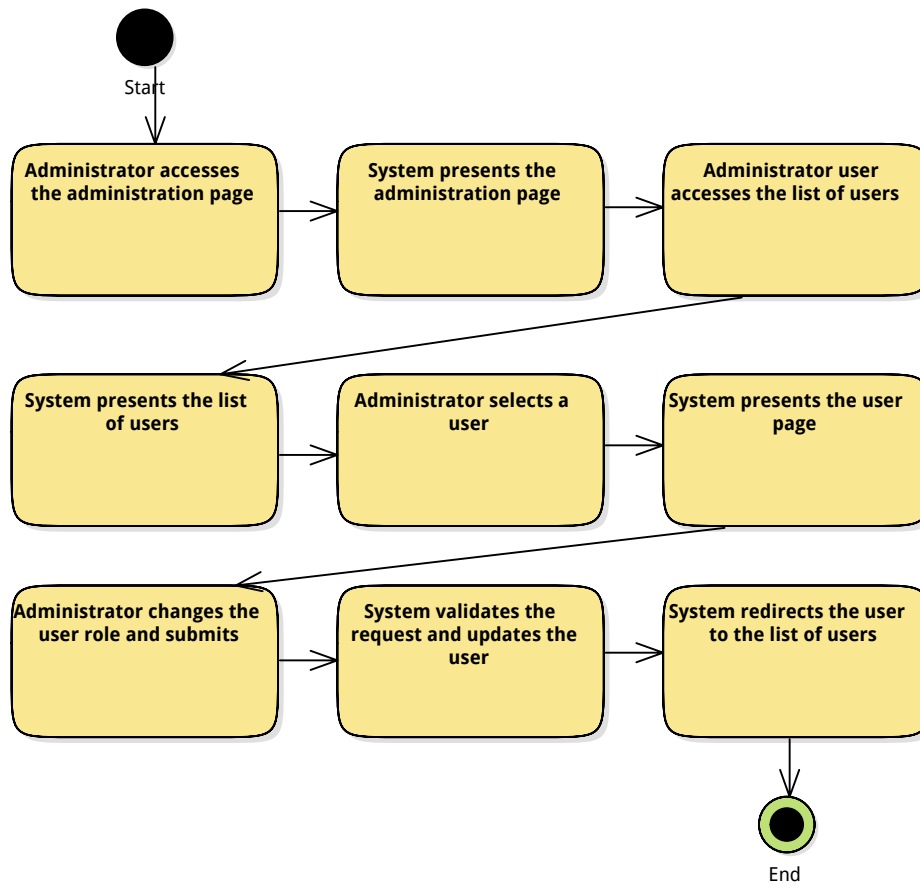


Figure 3.37: Activity diagram for the use case update user role

DELETE USER

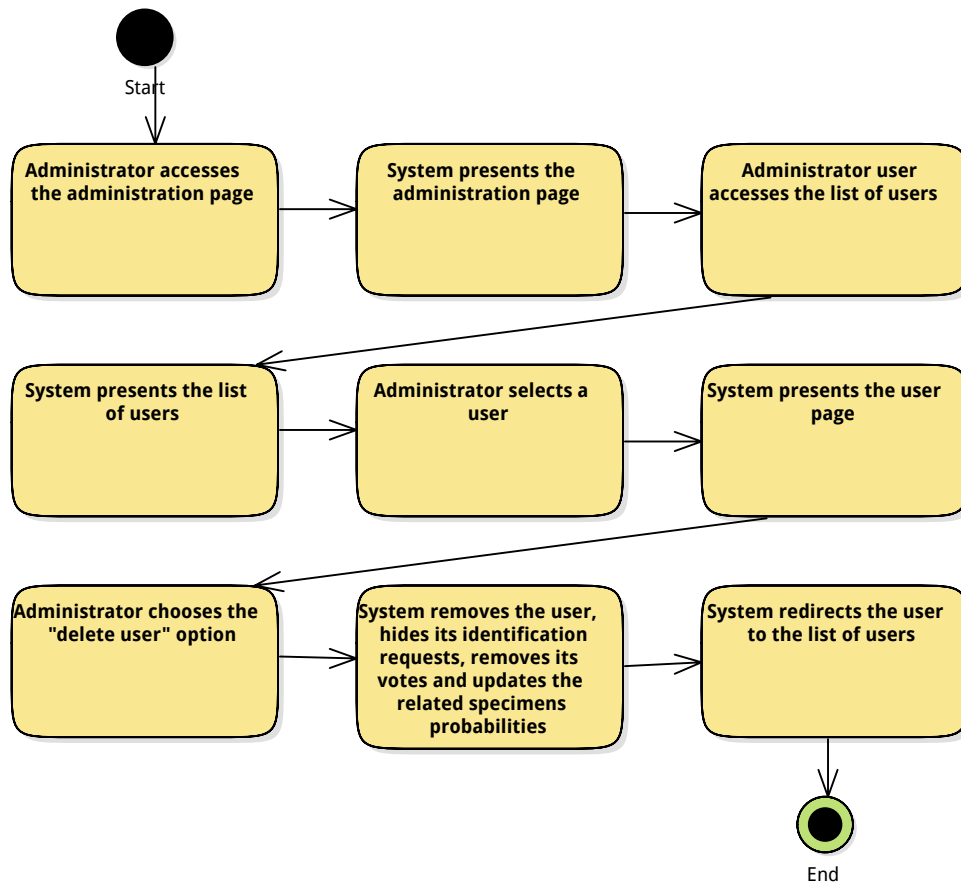


Figure 3.38: Activity diagram for the use case delete user

3.5.18 DEFINE QUIZ PARAMETERS

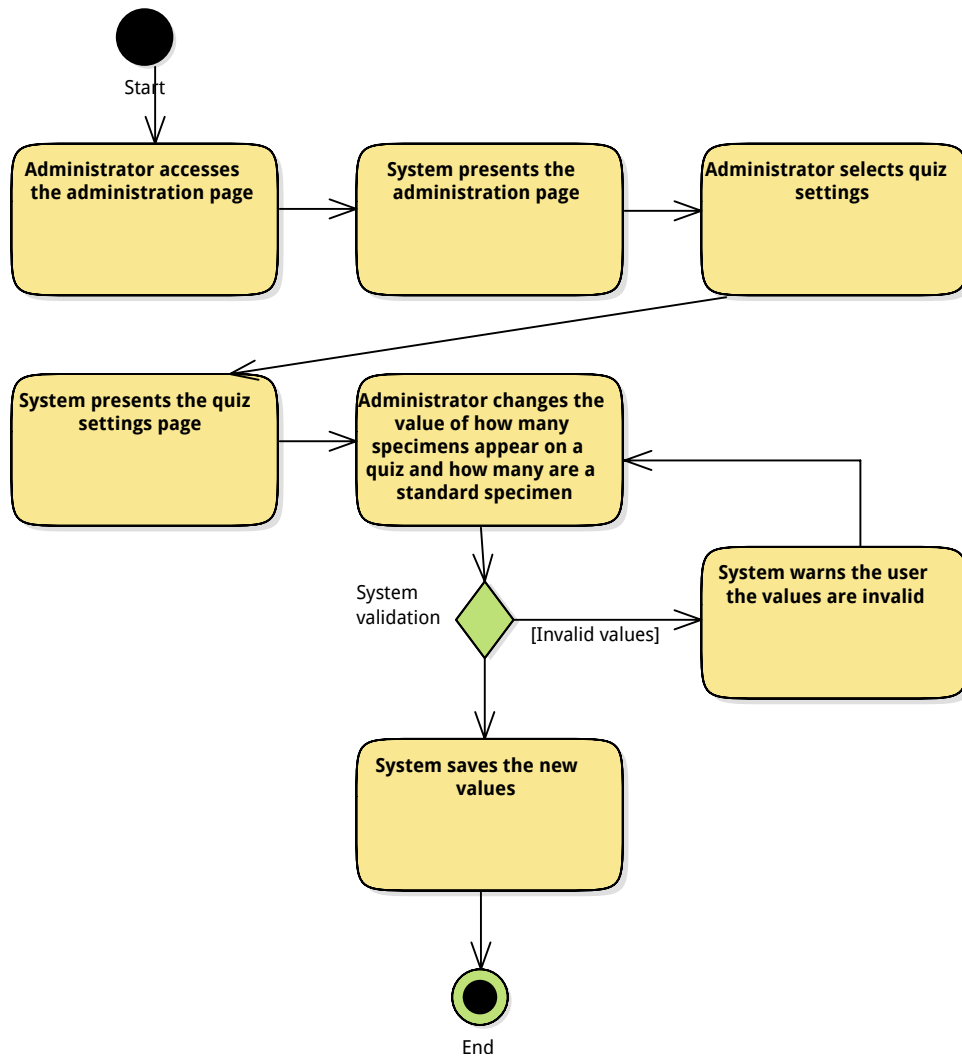


Figure 3.39: Activity diagram for the use case define quiz parameters

3.5.19 SUGGEST NEW CULTIVAR

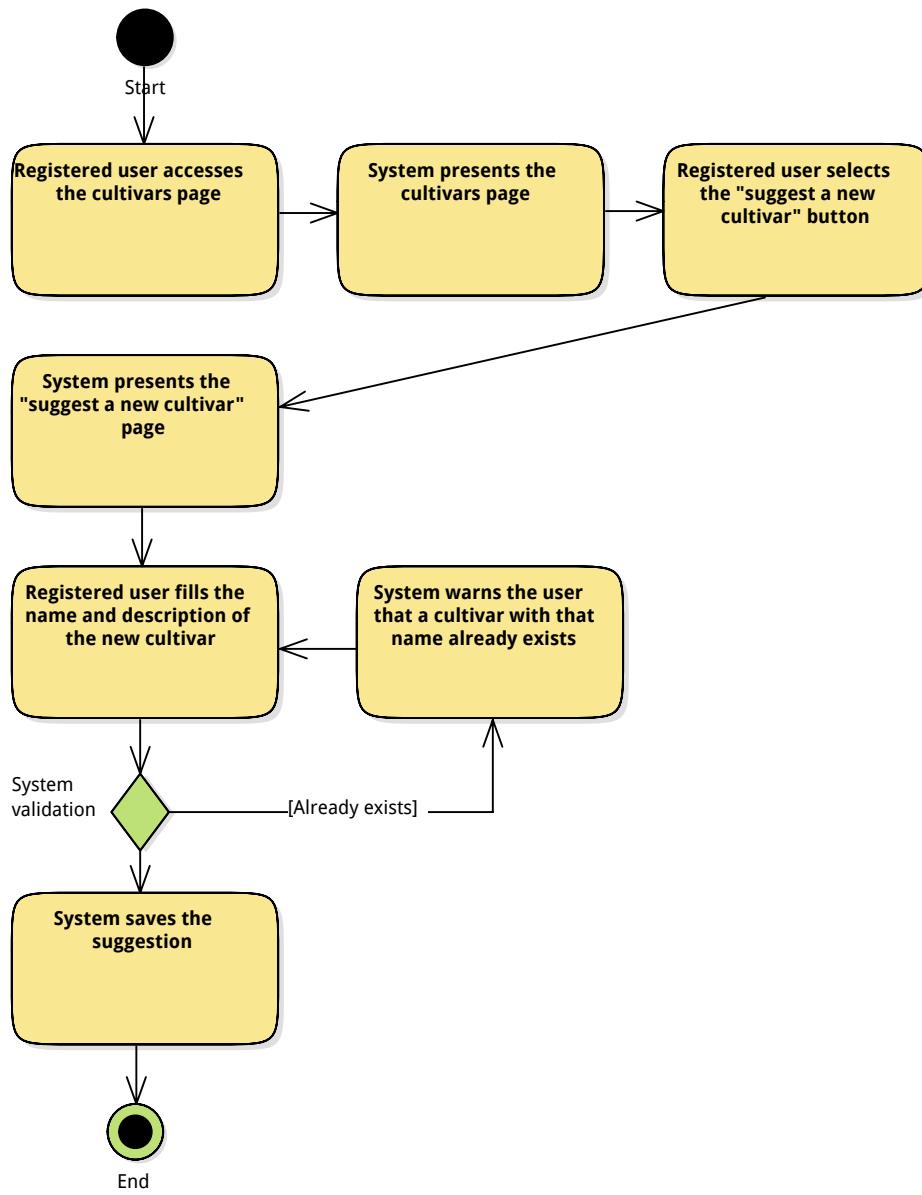


Figure 3.40: Activity diagram for the use case suggest new cultivar

3.5.20 VOTE CULTIVAR FOR A SPECIMEN

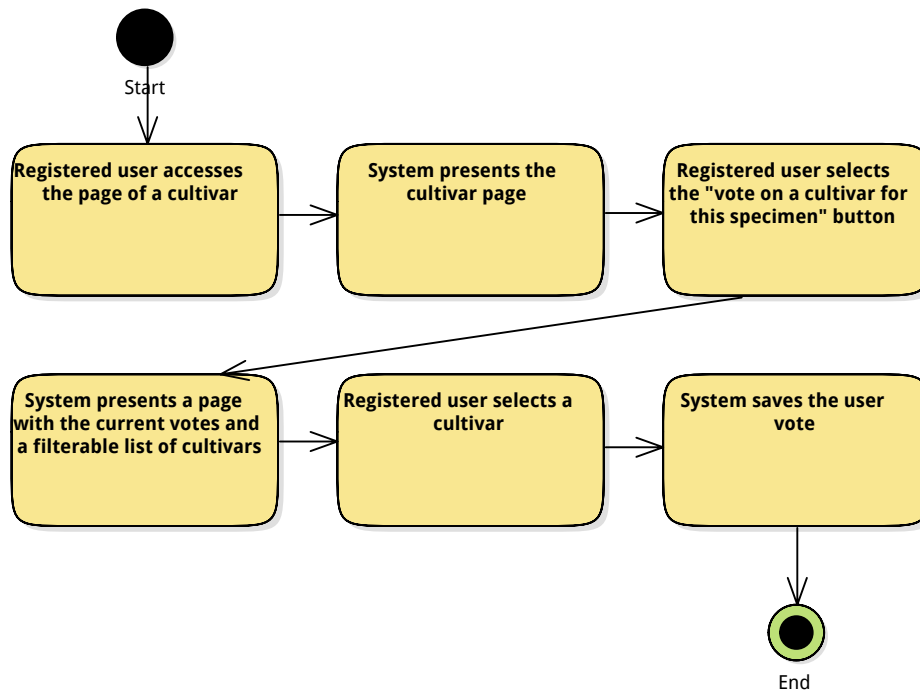


Figure 3.41: Activity diagram for the use case vote cultivar for a specimen

IMPLEMENTATION

The system is implemented in Python, making use of the Django Web framework, PostgreSQL as the **DBMS** and Bootstrap—an HTML, CSS, and JavaScript framework.

4.1 PYTHON

```
Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
Sparse is better than dense.  
Readability counts.  
Special cases aren't special enough to break the rules.  
Although practicality beats purity.  
Errors should never pass silently.  
Unless explicitly silenced.  
In the face of ambiguity, refuse the temptation to guess.  
There should be one--and preferably only one--obvious way to do it.  
Although that way may not be obvious at first unless you're Dutch.  
Now is better than never.  
Although never is often better than *right* now.  
If the implementation is hard to explain, it's a bad idea.  
If the implementation is easy to explain, it may be a good idea.  
Namespaces are one honking great idea--let's do more of those!
```

Figure 4.1: The Zen of Python, by Tim Peters

The back-end system is implemented in Python, which is a powerful, fast, easy to learn—the syntax of the language is designed to be readable—and dynamic programming language

that has been around for quite some time now (appeared in 1991) and is used in nearly every field and variety of application domains.

Python has a clean, concise and readable syntax that makes it a pleasant programming language to work with, and makes use of the **Don't Repeat Yourself (DRY)** principle of software development—stated as “Every piece of knowledge must have a single, unambiguous, authoritative representation within a system”—popularised by the 1999 Pragmatic Programmer coding practices book[27].

Python has a feature-rich standard library and, by being modular and allowing the core language kernel to be extended, has many frameworks and third-party libraries available.

4.1.1 PYTHON PACKAGES

One of the advantages of Python is the reuse of code used to solve similar problems, made easy by Python packages. There is a central repository for packages, the **Python Package Index (PyPI)**, with around fifty thousand packages to the date (e.g. authentication, testing, encoding/decoding). Even Django, the Web framework used in this project, is distributed as a Python package.

One of the easiest and fastest ways to install and manage Python packages is through the *pip* tool, which takes care of downloading and installing the package and its dependencies. Listing 4.1 shows an example of some of the commands this tool allows.

Listing 4.1: Example of pip commands

```
$ pip install django
$ pip search xml
$ pip install simplejson
$ pip uninstall simplejson
```

4.1.2 VIRTUAL ENVIRONMENT TOOLS

Virtualenv is a tool that allows the creation of Python sandboxes, that is, it clones the Python system so that a complete and isolated copy of Python is provided to the user to develop the projects they want. In this sandbox the user may install any packages they want, without interfering with the system-wide Python installation. Listing 4.2 on the facing page shows an example of some of the commands this tool allows and shows how the python pathname changes when inside the environment. The source command shown in the example is used to activate the virtual environment by *sourcing* a script which modifies the \$PYTHONPATH environment variable, allowing the Python interpreter to import packages

from the correct location instead of the default system location.

Listing 4.2: Example of virtualenv usage

```
$ which python
/usr/bin/python
$ virtualenv camellia_env      # create the sandbox
$ source camellia_env/bin/activate  # work on the sandbox
(camellia_env)$ which python
/home/user/.virtualenvs/camellia_env/bin/python
(camellia_env)$
(camellia_env)$ pip install django  # install django inside the sandbox
(camellia_env)$
(camellia_env)$ deactivate      # exit the sandbox
$
```

Virtualenvwrapper, another tool, extends the virtualenv and simplifies its usage. It organizes all the virtual environments in one place, uses a single command to switch between them, and allows tab completion for commands that take a virtual environment as argument. Its usage is exemplified in listing 4.3.

Listing 4.3: Example of virtualenvwrapper usage

```
$ pip install virtualenvwrapper
$ mkvirtualenv camellia_env
(camellia_env)$
(camellia_env)$ deactivate
$
$ workon example_env      # virtualenvwrapper wrapper command
(example_env)$ workon camellia_env  # switch between environments
(camellia_env)$
```

4.2 DJANGO

The system is implemented making use of the Django Web framework, which was designed to make common Web-development tasks fast and easy[28].

*Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design.*¹

4.2.1 MODEL–VIEW–CONTROLLER

Django follows the **MVC** pattern, a software architectural pattern for implementing user interfaces—originally developed for desktop computing but already widely adopted as an

¹www.djangoproject.com

architecture for Web applications in major programming languages—where the application is divided into three tiers or components so that the internal representations of information are separated from the ways that the information is presented to or accepted from the user[29]. The three components, as the name implies, are the model, the view, and the controller.

Typically, the controller component interacts with the model to update its state (e.g. add new user, change some system property) and can also send commands to its associated view so that the view can be updated. In Django, the controller can be seen as the framework itself, as it is the machinery that for dispatches the requests to the appropriate view.

The model component notifies its associated views and controllers when its state changes, allowing the views to produce updated output and the controllers to update the available set of commands. It is also possible to the MVC to be “passive”, meaning that the model must be polled for updates rather than notifying the other components.

The view component requests information from the model and then uses it to generate an output representation to the user. In Django’s interpretation of the MVC, the “view”—a Python callback function for a particular URL—describes which data gets presented to the user. A Django view does not define how the information is presented or looks like—that is the role of templates.

It might be said that Django is a “MTV” framework—that is, “model”, “template”, and “view”.

4.2.2 REUSABLE APPS

Many Python and Django projects share common problems and one of most appealing aspects of Django is that of *reusable apps*. Besides separating the models, views and controllers, Django encourages developers to separate the functionality of the overall system into loosely-coupled *apps*—each one defining its own models, views and templates—that can then be packaged up as Python modules and published for the community for reuse in other projects.

As an example, an app providing user registration and authentication using social networks—*allauth*—is available and commonly used in systems which require this functionality.

In this project there are some 3rd party *apps* used with Django, like the *allauth* (authentication related), *south* (database migrations) and *photologue* (photo galleries), discussed in more detail in the following sections.

4.2.3 MODELS

The models define the persistent data—that will be available to the application—as one Python class for each data “entity”. They also define how that data should be accessed.

There are some prerequisites: the database and database user must be created, the database adapter for Python must be installed, and the connection must be configured on the project settings file (`settings.py`). This is explained in more detail in section 4.4 on page 104.

Listing 4.4: The ‘Quiz’ model

```
1 class Quiz(models.Model):
2     uuid = models.CharField(max_length=36, default=make_uuid,
3                             editable=False)
4     created = models.DateTimeField(auto_now_add=True)
5     user = models.ForeignKey(User, null=True, related_name='quizzes')
6     submitted = models.BooleanField(default=False)
7
8     def __str__(self):
9         return "Quiz %d" % self.id
10
11     class Meta:
12         ordering = ['-pk']
```

Listing 4.4 shows the definition of the Quiz model, which has four attributes: `uuid` (a universally unique identifier), `created` (the date/time at which the quiz was created/generated), `user` (a foreign key to the User model, which tells which user the quiz belongs to), and `submitted` (a boolean value indicating whether or not the quiz was already finished/submitted by the user).

The model definition also specifies how an instance of the model should be displayed if it is printed on screen (the `__str__()` method) and what is the default ordering when obtaining lists of objects (defined by the `ordering` list inside the `Meta` class), in this case being ordered by primary key, in descending order (denoted by the ‘-’ prefix).

Django provides access to this data through its [Object-Relational Mapper \(ORM\)](#), providing a high-level [Application Programming Interface \(API\)](#) for querying the database. This saves time and reduces mistakes in the application code. Listing 4.5 shows an example of how one might query the database for all the Quizzes submitted by the User with ID 3.

Listing 4.5: Querying for all Quizzes the User 3 submitted

```
>>> Quiz.objects.filter(user__id=3, submitted=True)
[<Quiz: Quiz 73>, <Quiz: Quiz 72>, <Quiz: Quiz 71>, <Quiz: Quiz 63>]
>>>
```

MIGRATIONS

By default, changes to models causing database table modifications must be followed by the *syncdb* command, but each time it is run (e.g. after only adding, deleting or modifying a field) the entire table is deleted and recreated.

Listing 4.6: Updating the database without a migration tool

```
$ python manage.py syncdb
```

However, there is a much more useful tool—South—that provides consistent, easy-to-use and database-agnostic migrations for Django[30]. South can update a field without having to recreate the table, thus maintaining all the existing data, and even supports rollback. Listing 4.14 on page 105 shows an example of creating a migration—after adding a field on a model—, and applying it.

Listing 4.7: Database migrations using the South tool

```
(camellia_env)$ ./manage.py schemamigration users --auto
+ Added field reputation on users.UserProfile
Created 0002_auto__add_field_userprofile_reputation.py. You can now apply this
migration with: ./manage.py migrate users

(camellia_env)$ ./manage.py migrate users
Running migrations for users:
- Migrating forwards to 0002_auto__add_field_userprofile_reputation.
> users:0002_auto__add_field_userprofile_reputation
- Loading initial data for users.
Installed 0 object(s) from 0 fixture(s)
```

The latest version of Django (1.7, released September 2014) has built-in support for schema migrations, although this project was developed with Django version 1.6 and wasn't yet migrated.

4.2.4 VIEWS

The view's purpose is to parse the incoming HTTP requests, extract the relevant data (if any) from the data models, and respond with an appropriate HTTP response. This response is usually constructed by rendering a template. There are two types of views: function-based and class-based. Class-based views provide an alternative way to implement views as Python objects instead of functions, having the advantage of providing a better organization of code related to specific HTTP methods (e.g. GET, POST) as they can be addressed by separate methods (see listing 4.8 on the facing page) instead of conditional branching when using a function (see listing 4.9 on the next page). Another advantage is the use of object oriented features, such as multiple inheritance, to factor code into reusable components[31].

Listing 4.8: Class-based view handling HTTP GET request

```
1 class QuizView(View):
2     def get(self, request):
3         template = 'quiz.html'
4         args = {}
5
6         if request.user.is_authenticated():
7             quiz_and_answers =
8                 get_or_create_quiz_and_answers_for_user(request.user)
9             args['quiz'] = quiz_and_answers['quiz']
10            args['answers'] = quiz_and_answers['answers']
11        else:
12            args['quiz'] = None
13            args['answers'] = None
14            pass
15        return render(request, template, args)
```

Listing 4.8 shows a class-based view with a `get` function that handles HTTP GET requests dispatched by Django.

Listing 4.9: Example of a function-based view

```
1 from django.http import HttpResponse
2
3 def my_view(request):
4     if request.method == 'GET':
5         # <view logic for GET>
6         return HttpResponse('result')
7     elif request.method == 'POST':
8         # <view logic for POST>
9         return HttpResponse('result')
```

Listing 4.9 shows a function-based view using conditional branching to check the request method.

4.2.5 TEMPLATES

A template—the “view” on some frameworks and on the MVC pattern—is a text file that can generate any text-based format (e.g. HTML, XML, CSV) and is meant to express presentation, not program logic, although it supports variables and a limited set of control logic. The variables are replaced by values when the template is evaluated, and the control logic is possible by using tags (e.g. *if*, *for*). When dealing with the web interface, using an

HTML format—which is what is used on this project—, the template describes how the interface is rendered to the end user.

Listing 4.10: Example of a template

```
1 {% extends "base.html" %}
2 {% load i18n %}
3 {% load staticfiles %}
4
5 {% block content %}
6     <h1>Welcome{% if user.first_name %} back, {{ user.first_name
7         }}{% endif %}!</h1>
8         :
9     <h3>Latest specimens</h3>
10    <div id="latest_specimens" style="text-align: center">
11    {% for specimen in latest_specimens %}
12        <div class="specimen" style="display: inline-block">
13            {% if specimen.gallery %}
14                :
15            </div>
16        </div>
17    {% endfor %}
18 </div>
19 </block content %}
```

Listing 4.10 shows an example of a template. It does use inheritance, a powerful part of Django’s template engine, that allows to build a base “skeleton” template that contains all the common elements of the site and defines blocks that child templates can override[32].

In this case, the template inherits from `base.html` (`{% extends "base.html" %}` line) and this is why there is no `<html>` and `<head>` tags on this file (they are on the parent template). Also, this template overrides its parent content block (`{% block content %}` line), resulting on a web page defined by `base.html` with some blocks defined or overridden by this child.

Inside the content block we can see that there are some template tags and variables. For example, the `if` tag checks if the user has a first name, `{% if user.first_name %}...{% endif %}` and if it does, prints the name, using the `{{ user.first_name }}` variable. There is also a `for` tag, `{% for specimen in latest_specimens %}` that iterates through all the specimens found in the `latest_specimens` variable.

In this example there are some variables that come from the view and others that are automatically injected on the template, like the `{{ user }}` variable, which is injected on all templates automatically by the Django’s authentication support that is bundled as a Django contrib module (`django.contrib.auth`, added to the project settings via the `INSTALLED_APPS` tuple). The `{{ latest_specimens }}` variable is passed to this template by the view that renders it, as shown on the last two lines of listing 4.11 on the next page.

Listing 4.11: Example passing a variable from the view to the template

```

1           :
2 class IndexView(View):
3     def get(self, request):
4         template = 'index.html'
5         args = {}
6         latest_specimens = Specimen.objects.order_by('-id')[:3]
7         args['latest_specimens'] = latest_specimens
8         return render(request, template, args)

```

4.3 DJANGO PROJECT STRUCTURE

Django has historically used the term project to describe an installation of Django. A project is defined primarily by a settings module.

A project is separated into multiple “applications”. The term application describes a Python package that provides some set of features and is just a set of code that interacts with various parts of the framework and include some combination of models, views, templates, static files, URLs, etc.

The idea is that applications should be loosely coupled, so that they may be reused in various projects, allowing developers to simply “plug in” the desired functionality.

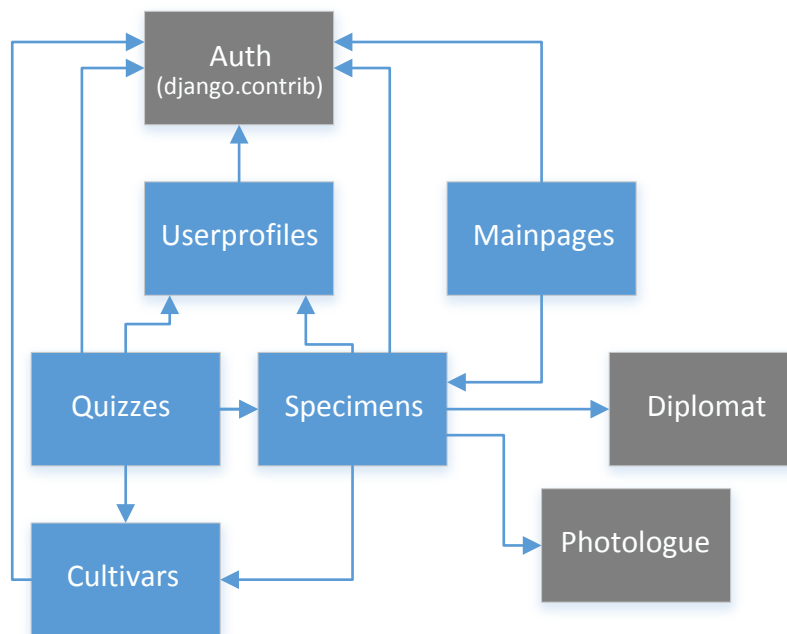


Figure 4.2: Dependency diagram for the Django project applications

This project is separated into five applications: `specimens`, `cultivars`, `quizzes`, `userprofiles` and `mainpages`. The dependencies between them are shown in figure 4.2 on the preceding page, where the applications on the grey boxes (`auth`, `photologue` and `diplomat`) are external applications and will be explained later. In this project the applications are not so loosely coupled—as almost every one depends on another—and could even be merged, but it was decided to maintain them separate so that the structure can easily be understood and so that the code is easier to read. Each application will now be described in more detail.

4.3.1 SPECIMENS APPLICATION

The specimens application contains all the models, views and templates related to the specimens.

MODELS

The models defined on the specimens application are the following:

Specimen

Contains the creation date, latitude and longitude geographic coordinates, the user that submitted the identification request and—if it is a standard specimen—the corresponding `Cultivar`, the user that set it as standard and the date it happened.

SpecimenGallery

Connects a `Specimen` to a `Gallery` from the `Photologue` application

UPOVCharacteristicType

Contains the description of the UPOV characteristic type (e.g. plant, branch, sepal, petal)

UPOVCharacteristicName

Has a `UPOVCharacteristicType` as foreign key and contains the description of the UPOV characteristic name (e.g. for the plant type: density of foliage and growth habit; for the branch type: zigzagging)

UPOVCharacteristicValue

Has a `UPOVCharacteristicName` as foreign key and contains the description of the UPOV characteristic value (e.g. for the density of foliage: sparse, medium and dense; for the growth habit: upright, semi-upright, spreading, drooping and horizontal)

SpecimenUPOVCharacteristicValue

Connects a `Specimen` with a `UPOVCharacteristicName` and a `UPOVCharacteristicValue`, where `UPOVCharacteristicName` is unique because each `Specimen` only have, at

most, one characteristic value (e.g. upright, semi-upright) for each characteristic name (e.g. growth habit)

SpecimenCultivar

Connects a Specimen with a Cultivar and has the probability of the specimen being from that cultivar, and the number of user votes, that is, how many users think that specimen belongs to that cultivar

UserSpecimenCultivarVote

Represents a user vote on this specimen. Each user may only vote for one cultivar on each specimen, meaning that a vote on another cultivar replaces the previous vote on that specimen

OtherCharacteristic

Saves characteristics beyond the UPOV ones, such as if the specimen was DNA tested

FORMS

The forms—used to accept input from site visitors, and then process and respond to the input—defined on the specimens application are the following:

SpecimenUPOVCharacteristicValueForm

This is the class responsible for constructing the new identification request form, used by the `NewSpecimenView`. It gets all the existing and possible characteristics from the models described before, and for some of the characteristics—that have an example image—it constructs the form field with that image instead of, say, a simple radio button option, as shown in figure 4.3, making use of a customized `RadioSelect` widget.

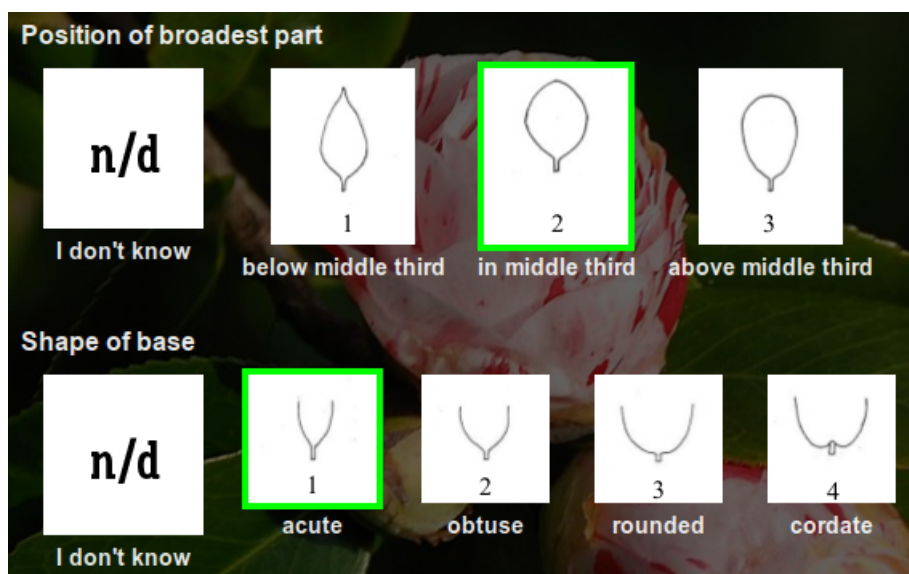


Figure 4.3: Form fields images example for the identification request form

VIEWS

The views defined on the specimens application are the following:

SpecimenView

This view is responsible for rendering a specimen page, using the `specimen.html` template.

SpecimensView

This view receives the request to the list of specimens and renders a page, using the `specimens.html` template, with a paginated list

NewSpecimenView

This view prepares the page with the form for the users to submit a new specimen identification request. When the user submits the form, this view validates it and saves the information at the corresponding models

EditSpecimenView

As the name implies, this view is used to edit an existing specimen. It renders the same form used to submit a new identification request, but it is pre-populated with the existing information. It then validates the submit request and saves the changed information

VoteCultivarView

This is the view used to present a list of cultivars so the user may vote on one, for a particular specimen. When receiving a POST request with the user vote it saves the vote accordingly

SpecimensLocationsView

Used to return a [JavaScript Object Notation \(JSON\)](#) response with all the specimens close to a given geographical point.

TEMPLATES

The templates defined on the specimens application are the following:

new_specimen.html

This template receives a form from the `NewSpecimenView` and defines how the page for the user to submit a new specimen identification request is presented

specimen.html

This template defines how the specimen details page is presented

specimens.html

This template receives the specimen list, paginated, and shows the pagination controls (e.g. go to the previous, next or last page)

vote_cultivar.html

This template shows the list of current possible cultivars (voted by the users) to the current specimen, allowing the user to vote in one of them or to vote in a new cultivar (not yet voted for this specimen) from the filterable list of cultivars that is presented

4.3.2 CULTIVARS APPLICATION

The `cultivars` application contains all the models, views and templates related to the cultivars.

MODELS

The models defined on the `cultivars` application are the following:

Species

Contains the name of a species

Cultivar

Has the name and description (initially from the [ICR](#)) of a cultivar, plus the creation date and the user who created it

CultivarSpecies

Makes the connection between a `Cultivar` and a `Species`

FORMS

The forms—used to accept input from site visitors, and then process and respond to the input—defined on the `cultivars` application are the following:

NewCultivarForm

This form class extends the `ModelForm` class, meaning that it uses a model to know which fields to present on the form. In this case, it uses the `Cultivar` model and excludes the `created` and `user` fields as they are automatically saved when the form is validated by the `NewCultivarView`

VIEWS

The views defined on the `cultivars` application are the following:

CultivarView

Receives the request to present the page of a specific cultivar and renders a page, using the `cultivar.html` template, with that cultivar information

CultivarsView

Receives the request to present the list of cultivars and renders a page, using the `cultivars.html` template, with a paginated list

NewCultivarView

Uses the `new_cultivar.html` template to present a form for a user to suggest a new cultivar, and receives the response, validates it and saves the information

EditCultivarView

This view is used to render a page for editing an existing cultivar

SearchNameView

Receives a search keyword, via a POST request method, and returns a list of cultivars which name contains that keyword, using the `cultivar_result.html` template

TEMPLATES

The templates defined on the `cultivars` application are the following:

cultivar.html

This template defines the cultivar details page

cultivar_result.html

This template receives a paginated list of filtered cultivars from `SearchNameView`, shows the pagination controls (e.g. go to the previous, next or last page) and—if the request comes from the specimen voting page or from a quiz—provides the JavaScript functions to allow the user to select one of the cultivars from the filtered list and, accordingly to which page the request originated—specimen or quiz—use the selected cultivar as a vote to the specimen or as an answer to the quiz, respectively

cultivars.html

This template receives the cultivar list, paginated, and shows the pagination controls (e.g. go to the previous, next or last page)

new_cultivar.html

This template receives a form from the `NewCultivarView` and presents it on a page for the user to submit a new cultivar suggestion

4.3.3 QUIZZES APPLICATION

The `quizzes` application contains all the models, views and templates related to the quizzes.

MODELS

The models defined on the quizzes application are the following:

Quiz

Contains the creation date, the associated user and a boolean field telling if the quiz was already finished/submitted

Answer

This model saves all the quizzes answers. It is the relation between a Quiz, a Specimen and a Cultivar

QuizParameters

This model is a “singleton” and holds the quizzes settings in one record. It has the number of standard specimens that appear on quizzes (minimum of one) and the number of non-standard specimens.

FORMS

The quizzes application has no forms defined.

VIEWS

The views defined on the quizzes application are the following:

QuizView

When receiving a GET request this view renders the quiz page for the user, using the `quiz.html` template, and when receiving a POST request it validates the user answers to the quiz and renders the results page, using the `quiz_result.html` template

VoteCultivarQuizView

This view receives a POST request each time the user answers or changes one of the quiz answers, saving that answer to the Answer model, so that the user may leave the quiz page and return at a later time without losing the answers already answered

TEMPLATES

The templates defined on the quizzes application are the following:

quiz.html

This template receives a quiz from the QuizView and has the proper JavaScript functions to load the specimens gallery and to present the list of possible cultivars for the user to choose one (the list is obtained by sending a request to the SearchNameView from the Cultivars application)

quiz_result.html

This template receives the information from the `QuizView` POST result, showing how many answers the user filled (it is not mandatory for the users to answer all the answers before submitting a quiz), how many quizzes the user already answered, and its current reputation

4.3.4 USERPROFILES APPLICATION

The `userprofiles` application contains all the models, views and templates related to the user profiles.

MODELS

The models defined on the `userprofiles` application are the following:

UserProfile

This model is an extension of the `User` model from the Django's authentication system (`django.contrib.auth.models`) and stores additional user information like an avatar and the user reputation

ReputationParameters

This model is a “singleton” and holds the reputation settings in one record. It has the weight given to the user answers to quizzes (standard specimens) and the weight given to the number of votes the user has given to the specimens/cultivars. The sum of the weights must be between zero and one.

FORMS

The forms—used to accept input from site visitors, and then process and respond to the input—defined on the `userprofiles` application are the following:

MySignupForm

This form is used by the `django-allauth` application to extend the sign up form and must be configured at the project's `settings.py` file as shown in listing 4.12, so that the `allauth` knows which form to use. In this case, this form adds the first and last names to the sign up form and reorders the fields (e-mail and user name first, followed by first and last name, and the desired password).

Listing 4.12: Configuration of `allauth` custom sign up form

```
1 ACCOUNT_SIGNUP_FORM_CLASS = 'userprofiles.forms.MySignupForm'
```

UserProfileForm

This is a `ModelForm` that uses the `User` model so that the users may edit their profiles (e.g. first name, last name, email address, profile picture)

VIEWS

The views defined on the `userprofiles` application are the following:

profile

The profile view is used to pass the user information to the `profile.html` template

edit_profile

This view creates a `UserProfileForm` which it passes to the `edit_profile.html` and also validates the form when submitted, saving the updated values

TEMPLATES

The templates defined on the `userprofiles` application are the following:

profile.html

This template receives the user profile variables from the `profile` view and presents the user profile page

edit_profile.html

This template is used to present a form for editing the user profile

4.3.5 MAINPAGES APPLICATION

The mainpages application contains all the views and templates related to special-case pages, such as “About”, “FAQ” and “Contact”.

MODELS

This application has no models.

FORMS

The mainpages application has no forms defined.

VIEWS

The views defined on the mainpages application are the following:

IndexView

This view renders the `index.html` template, passing the latest `specimens` variable to it

AboutView

This view renders the `about.html` template

FAQView

This view renders the `faq.html` template

ContactView

This view renders the `contact.html` template

TEMPLATES

The templates defined on the `mainpages` application are the following:

index.html

This is the template that shows the website main page, containing a map with the nearest specimens and a list of thumbnails of the latest submitted identification requests

about.html

This template presents a simple paragraph about the website

faq.html

This template presents some useful questions and answers

contact.html

This template presents a form which users fill to send a message to the site owner

4.4 POSTGRESQL

As referred in section 2.4 on page 34, the chosen **DBMS** for this project was PostgreSQL.

Django, as seen before, manages the data storage through its models, but the database must be created beforehand. Listing 4.13 shows the database creation for this project.

Listing 4.13: Creation of the project database

```
[user@server ~]$ sudo su - postgres # change to the postgres system user
[postgres@server ~]$ createdb camellia # create the database
[postgres@server ~]$ createuser -P # create the database user (it will be
called django)
[postgres@server ~]$ psql # activate the PostgreSQL command line interface
postgres=# GRANT ALL PRIVILEGES ON DATABASE camellia TO django; -- grant
the new user (django) access to the database (camellia)
```

After the database creation, the Django project must be configured to use it. This is done by installing the database adapter (listing 4.14 on the next page) and by configuring

the connection on the project settings file (`settings.py`), as shown in listing 4.15 on the facing page.

Listing 4.14: Installing the PostgreSQL database adapter for Python

```
(camellia_env)$ pip install psycopg2
```

Listing 4.15: Django database connection configuration

```
1 DATABASES = {
2     'default': {
3         'ENGINE': 'django.db.backends.postgresql_psycopg2',
4         'NAME': 'camellia',
5         'USER': 'django',
6     }
7 }
```

After this, running `syncdb` will make Django add its initial configuration and other tables (from the defined models) to the database (listing 4.16 on the next page).

Listing 4.16: Running syncdb and listing the database tables

```
[user@server ~]$ workon camellia_env
(camellia_env)[user@server ~]$ python manage.py syncdb
(camellia_env)[user@server ~]$ sudo su - postgres
[postgres@server ~]\$ psql camellia

camellia=# \dt
                List of relations
 Schema |          Name          | Type | Owner
-----+-----+-----+-----
 public | account_emailaddress  | table | django
 public | account_emailconfirmation | table | django
 public | auth_group            | table | django
 public | auth_group_permissions | table | django
 public | auth_permission      | table | django
 public | auth_user            | table | django
 public | auth_user_groups     | table | django
 public | auth_user_user_permissions | table | django
 public | cultivars_cultivar    | table | django
 public | cultivars_cultivar_species | table | django
 public | cultivars_species    | table | django
 public | diplomat_isocountry  | table | django
 public | diplomat_isolanguage | table | django
 public | django_admin_log     | table | django
 public | django_content_type  | table | django
 public | django_session      | table | django
 public | django_site         | table | django
 public | mainpages_usermessage | table | django
 public | photologue_gallery   | table | django
 public | photologue_gallery_photos | table | django
 public | photologue_galleryupload | table | django
 public | photologue_photo     | table | django
 public | photologue_photoeffect | table | django
 public | photologue_photosize | table | django
 public | photologue_watermark | table | django
 public | quizzes_answer      | table | django
 public | quizzes_quiz        | table | django
 public | quizzes_quizparameters | table | django
 public | registration_registrationprofile | table | django
 public | socialaccount_socialaccount | table | django
 public | socialaccount_socialapp | table | django
 public | socialaccount_socialapp_sites | table | django
 public | socialaccount_socialtoken | table | django
 public | south_migrationhistory | table | django
 public | specimens_specimen   | table | django
 public | specimens_specimen_cultivar | table | django
 public | specimens_specimen_gallery | table | django
 public | specimens_specimen_upov_characteristic_value | table | django
 public | specimens_upov_characteristic_name | table | django
 public | specimens_upov_characteristic_name_translation | table | django
 public | specimens_upov_characteristic_type | table | django
 public | specimens_upov_characteristic_type_translation | table | django
 public | specimens_upov_characteristic_value | table | django
 public | specimens_upov_characteristic_value_translation | table | django
 public | specimens_user_specimen_cultivar_vote | table | django
 public | userprofiles_reputationparameters | table | django
 public | userprofiles_userprofile | table | django
(47 rows)
```

4.5 BOOTSTRAP

Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web[33].

Bootstrap is an open source front-end web development framework that provides templates that save a lot of design and implementation work and time. It comes with a bunch of customized (and customizable) elements, such as toggleable contextual menus, responsive navigation bars that collapse in smaller or mobile views, pagination links, alerts, among many others. Figure 4.4 shows an example of a Bootstrap template with some elements.

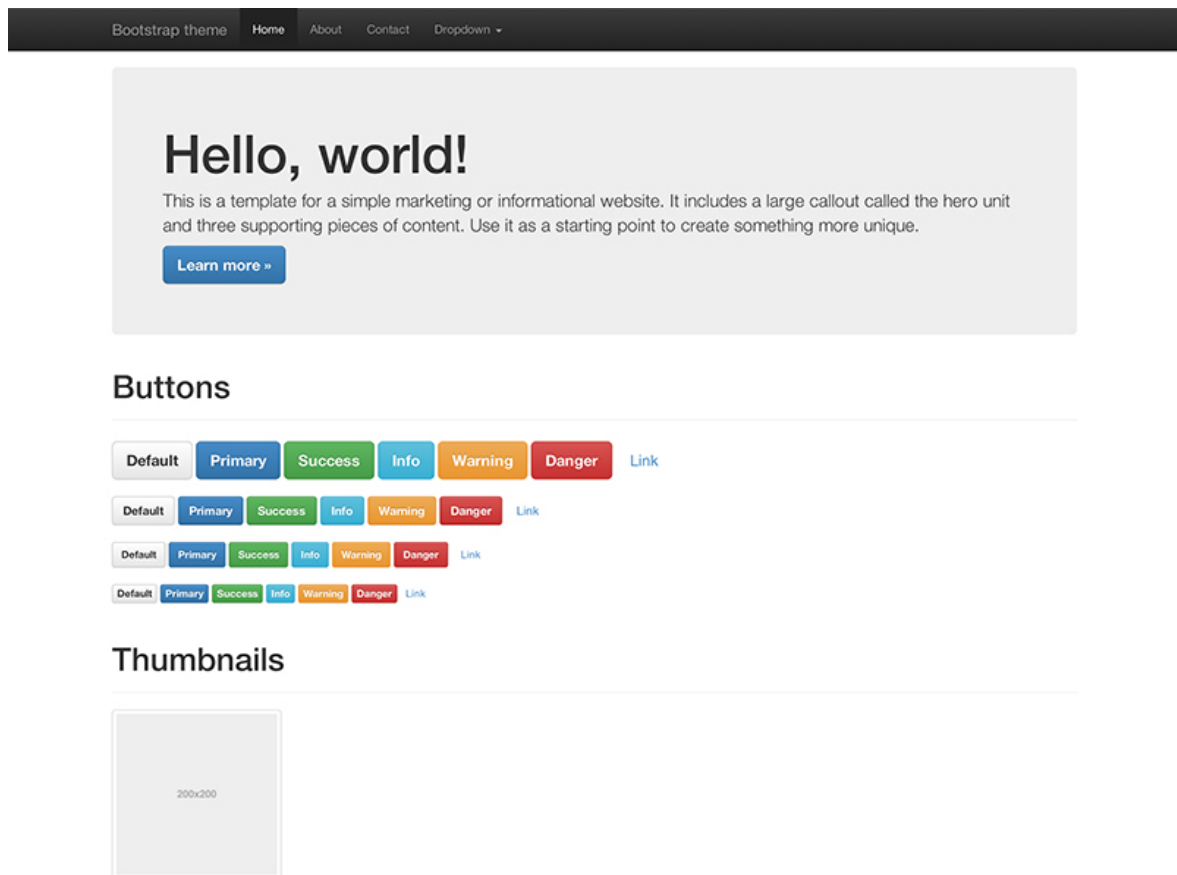


Figure 4.4: Bootstrap template example

4.5.1 DJANGO INTEGRATION

Bootstrap, being a front-end framework, doesn't clash with Django (a back-end framework), and the only thing that must be done to start using Bootstrap is to add the stylesheet and the JavaScript links to the web page. Listing 4.17 on the next page shows how this is done on the base .html template. All the other project templates inherit from this, meaning that the Bootstrap import must be done only in this parent template.

Listing 4.17: Including Bootstrap on the base.html template

```

1           :
2 <head>
3           :
4     <!-- Bootstrap -->
5     <link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap
6         /3.1.1/css/bootstrap.min.css">
7         :
7 </head>
8 <body>
9         :
10    <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
11    <script
12        src="//ajax.googleapis.com/ajax/libs/jquery/1.11.0/
13            jquery.min.js">
14    </script>
15    <!-- Latest compiled and minified JavaScript -->
16    <script
17        src="//netdna.bootstrapcdn.com/bootstrap/3.1.1/js/
18            bootstrap.min.js">
19    </script>
19           :

```

Although this is the only required configuration, there are some Python packages that help with the integration with the Django template forms. By default, to get a form into a template, one needs only to use the form variable passed by the view, such as `{{ the_form }}`. This will render its `<label>` and `<input>` elements appropriately but there are other output options for the `<label>/<input>` pairs^[34]:

- `{{ the_form.as_table }}`: renders them as table cells wrapped in `<tr>` tags
- `{{ the_form.as_p }}`: renders them wrapped in `<p>` tags
- `{{ the_form.as_ul }}`: renders them wrapped in `` tags

One of the packages that extend this functionality is `django-bootstrap-form`, that allows the use of a simple Django template tag, `{{ the_form|bootstrap }}`, instead of the output options described above, customizing the form by applying Bootstrap CSS classes to the `<label>/<input>` pairs.

4.6 WEB INTERFACE

4.6.1 MAIN PAGE

When the users access the website they will see a page like the one presented in figure 4.5, with the main navigation bar at the top with links to the specimens, cultivars and quizzes pages, and—if the user is not already logged in—two links for logging in or creating a new account. The page has a button for the users to open the new identification request page and has a map where the user can browse the nearby specimens. Selecting a specimen from the map allows the users to go to that specimen page containing its details. After the map, there is a list of the latest submitted specimen requests. At the bottom, there is another navigation bar with links to the about, FAQ, and contacts pages.

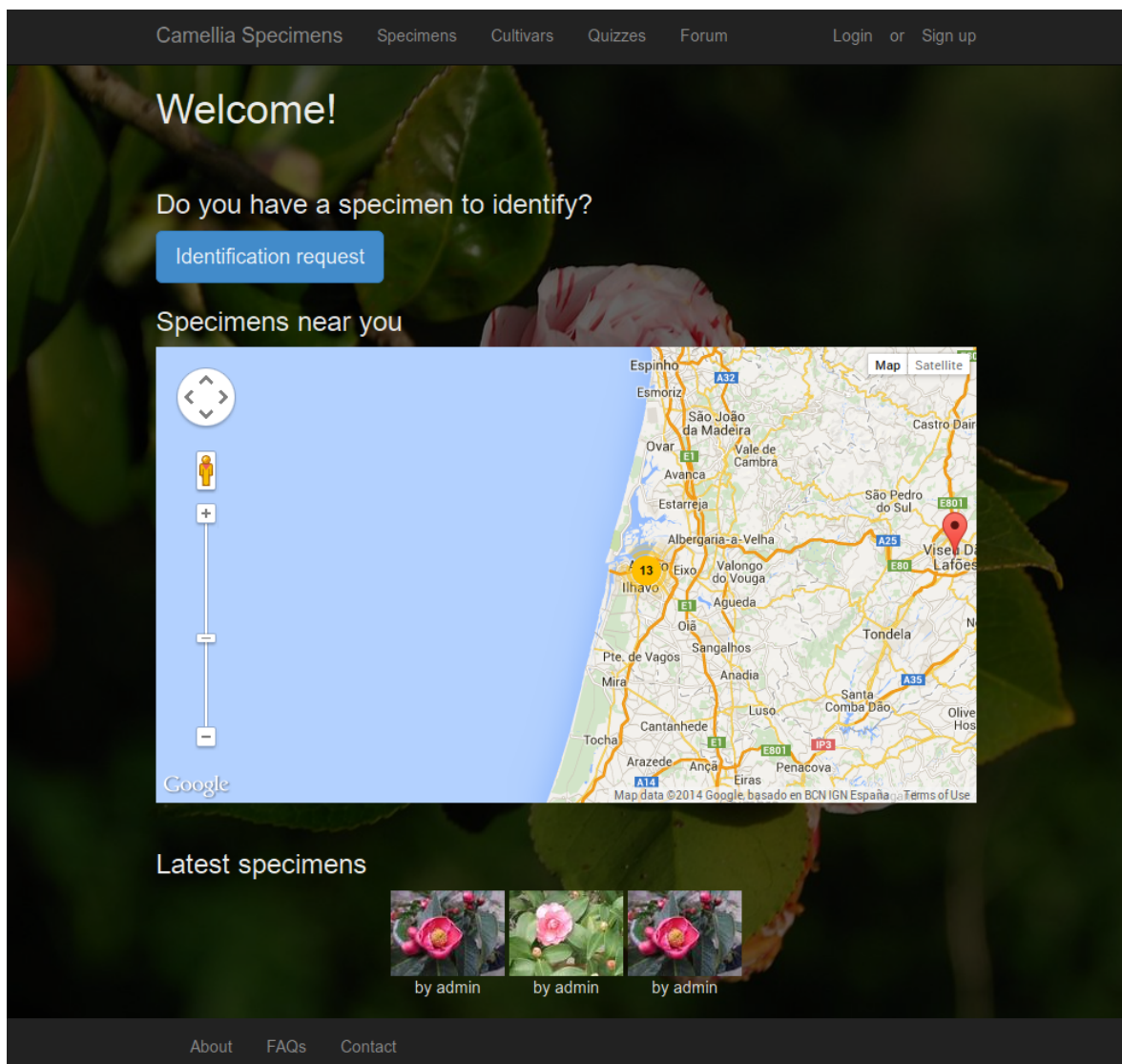


Figure 4.5: Website: main page

4.6.2 IDENTIFICATION REQUEST

The users may submit an identification request by filling a form where the only mandatory fields are the specimen photo and its location (figure 4.6 on the next page). The other fields are optional, although helpful, and consist of the fifty morphological characteristics defined by the UPOV. Figure 4.7 on page 112 shows an example of two characteristics, one with a visual aid and another with textual information. After submitting the request, it must be approved by a moderator and after that it will appear on the specimens page and on quizzes that users will answer.


Camellia Specimens Specimens Cultivars Quizzes Forum André ▾

New specimen/identification request

The location and photo are the only mandatory fields.

Location

Drag the marker on the map



Latitude
40.633162

Longitude
-8.659126

Photo

Photo
Browse... No file selected.

Figure 4.6: Website: identification request form (location and photo)

Camellia Specimens Specimens Cultivars Quizzes André ▾

Stamens

Arrangement

<input checked="" type="radio"/> n/d I don't know	<input type="radio"/> 1 sasanqua	<input type="radio"/> 2 circular	<input type="radio"/> 3 apricot	<input type="radio"/> 4 tea whisk
<input type="radio"/> 5 pinched	<input type="radio"/> 6 tubular	<input type="radio"/> 7 split	<input type="radio"/> 8 dispersed	

Style

Number of splits

- I don't know
- one
- two
- three
- four
- five

Figure 4.7: Website: identification request form (some characteristics)

4.6.3 QUIZZES

The quizzes page shows some specimens that the user must try to identify, selecting a cultivar for each one (figure 4.8 on the next page). The user may select a photo of a specimen to see its gallery and its characteristics. When clicking on the Answer button there is a list of cultivars from which the user selects one as the answer (figure 4.9 on page 115). On that list, the user may see the cultivar details so that they make a more informed choice. When the user submits the quiz the system will present a page with the results and the user current reputation. Before submitting a quiz the user may leave the page whenever they want and return later, as the answers are saved each time the user changes one. The results (user reputation and specimens/cultivars probabilities) are calculated only when the user submits the quiz and the next time the user goes to the Quizzes page there will be a newly generated quiz.

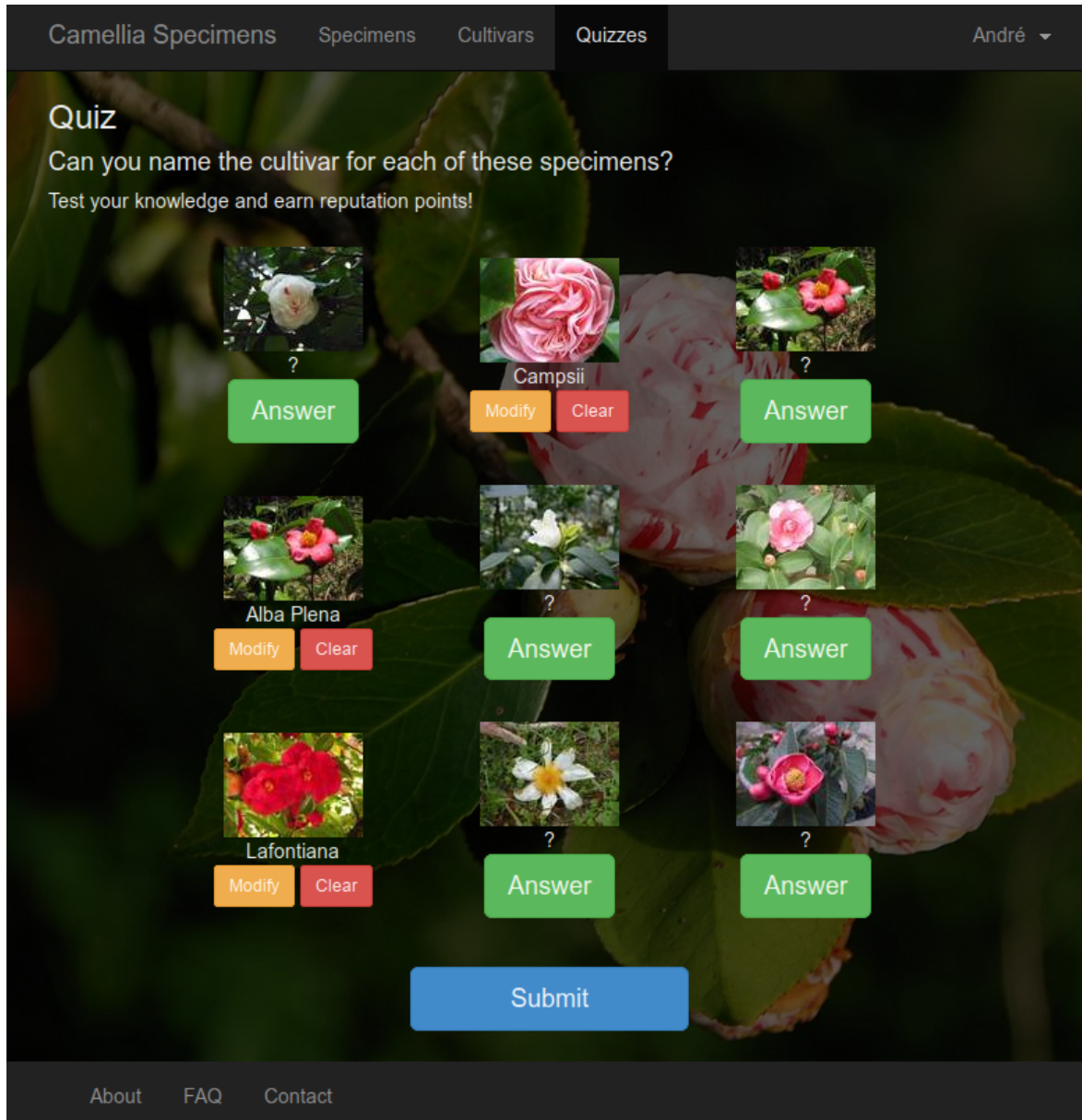


Figure 4.8: Website: quiz example

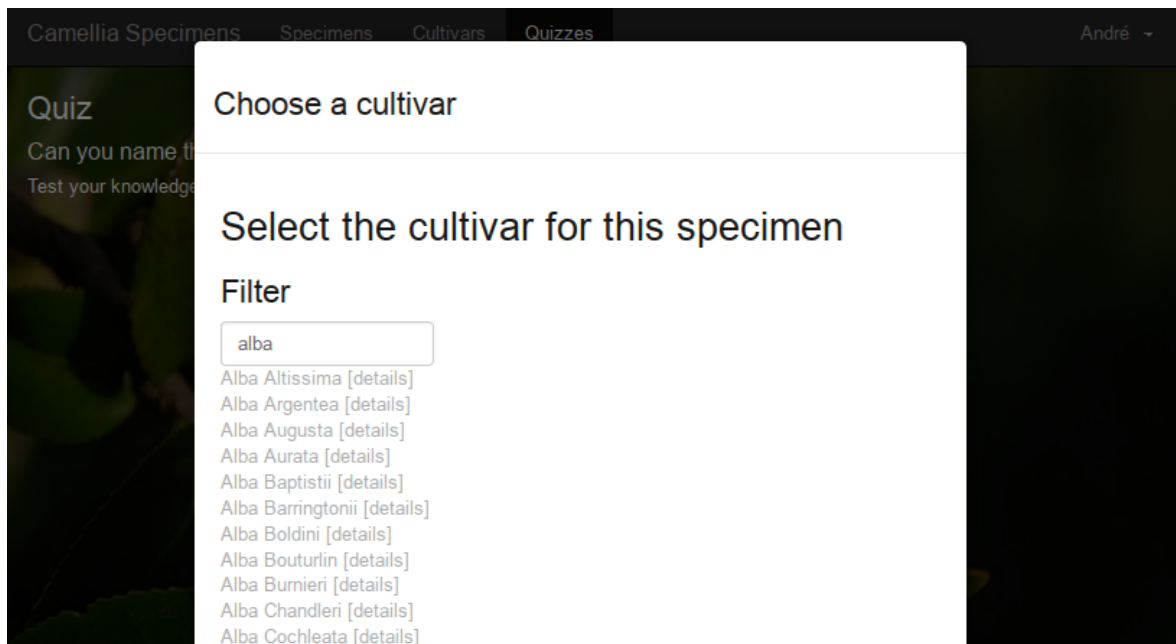
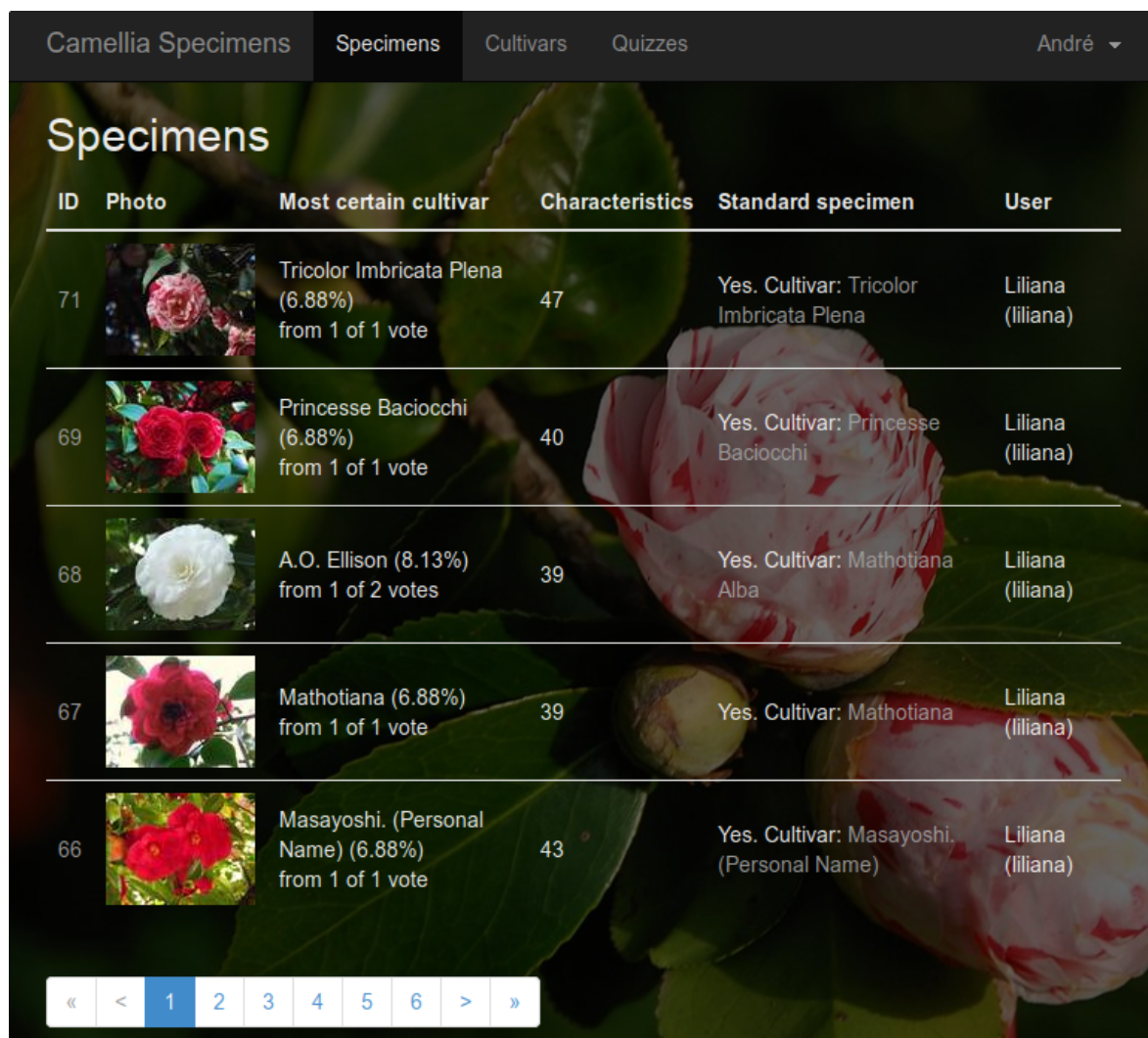







Figure 4.9: Website: selecting a cultivar for a quiz answer

4.6.4 SPECIMENS

When the user accesses the specimens page, a list with all the specimens—each originating from a user identification request—and their number of votes is presented (figure 4.10). The user can open one of the specimens and see which cultivars were voted, how many votes, and the probability of the specimen belonging to each of them (figure 4.11 on the facing page). On the specimen page there are the photos, location on a map and the known characteristics (figure 4.12 on page 118). The user may suggest or vote on a cultivar for this specific specimen (figure 4.13 on page 119) and may also, on the voting page, suggest a new cultivar if they can't find what they are looking for on the existing ones.



ID	Photo	Most certain cultivar	Characteristics	Standard specimen	User
71		Tricolor Imbricata Plena (6.88%) from 1 of 1 vote	47	Yes. Cultivar: Tricolor Imbricata Plena	Liliana (liliana)
69		Princesse Baciocchi (6.88%) from 1 of 1 vote	40	Yes. Cultivar: Princesse Baciocchi	Liliana (liliana)
68		A.O. Ellison (8.13%) from 1 of 2 votes	39	Yes. Cultivar: Mathotiana Alba	Liliana (liliana)
67		Mathotiana (6.88%) from 1 of 1 vote	39	Yes. Cultivar: Mathotiana	Liliana (liliana)
66		Masayoshi. (Personal Name) (6.88%) from 1 of 1 vote	43	Yes. Cultivar: Masayoshi. (Personal Name)	Liliana (liliana)

« < 1 2 3 4 5 6 > »

Figure 4.10: Website: specimens list

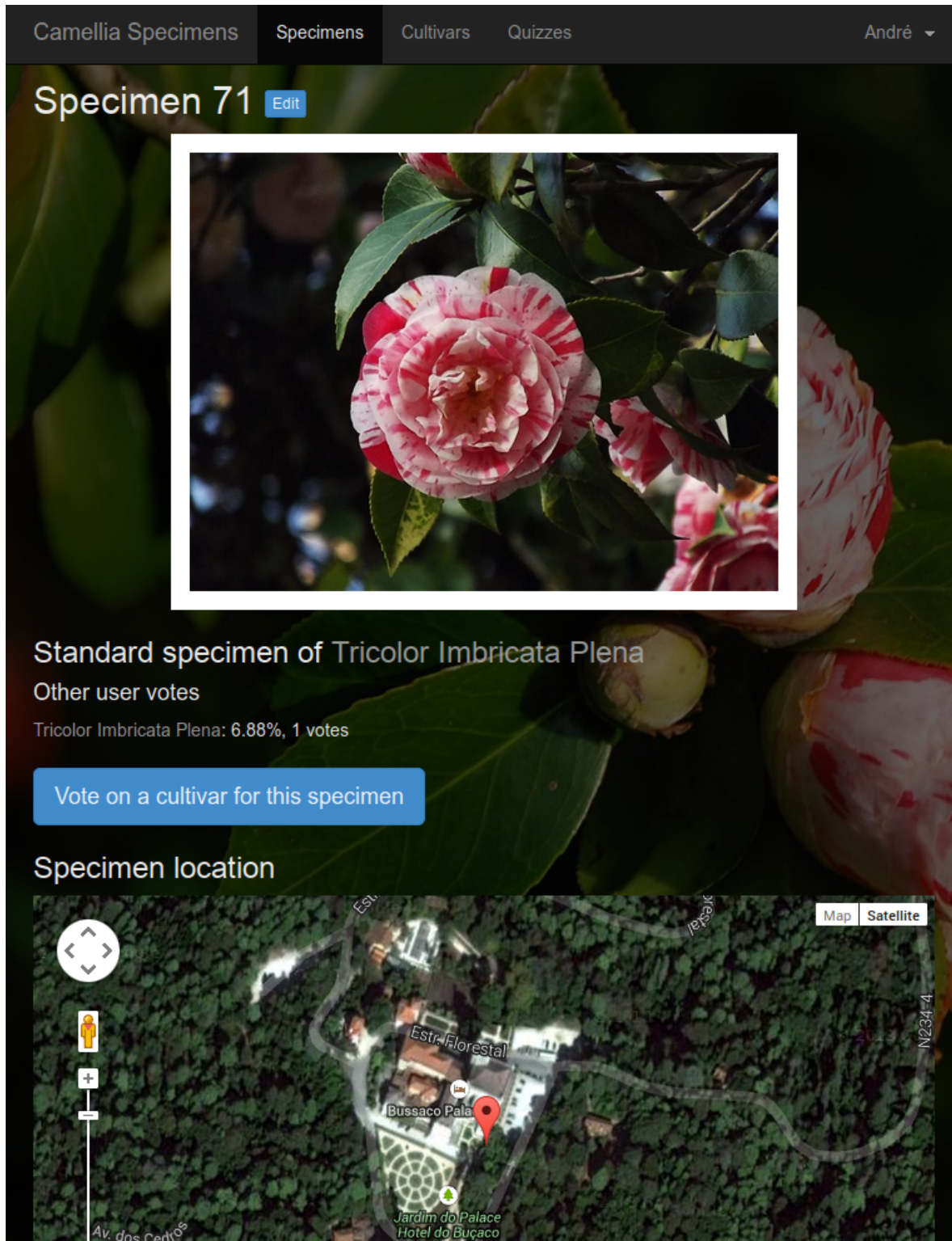
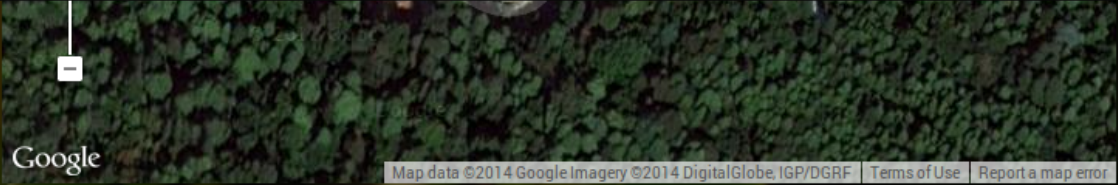


Figure 4.11: Website: specimen example (photos, map and voted cultivars)

Camellia Specimens Specimens Cultivars Quizzes André ▾

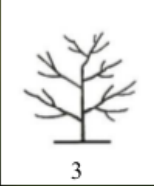


Google Map data ©2014 Google Imagery ©2014 DigitalGlobe, IGP/DGRF Terms of Use Report a map error

UPOV characteristics

Plant

Growth habit



3
Spreading

Density of foliage

Medium

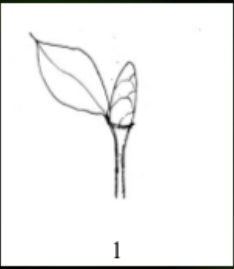
Vegetative Bud

Color

Green

Terminal Vegetative Bud

Number



1
One

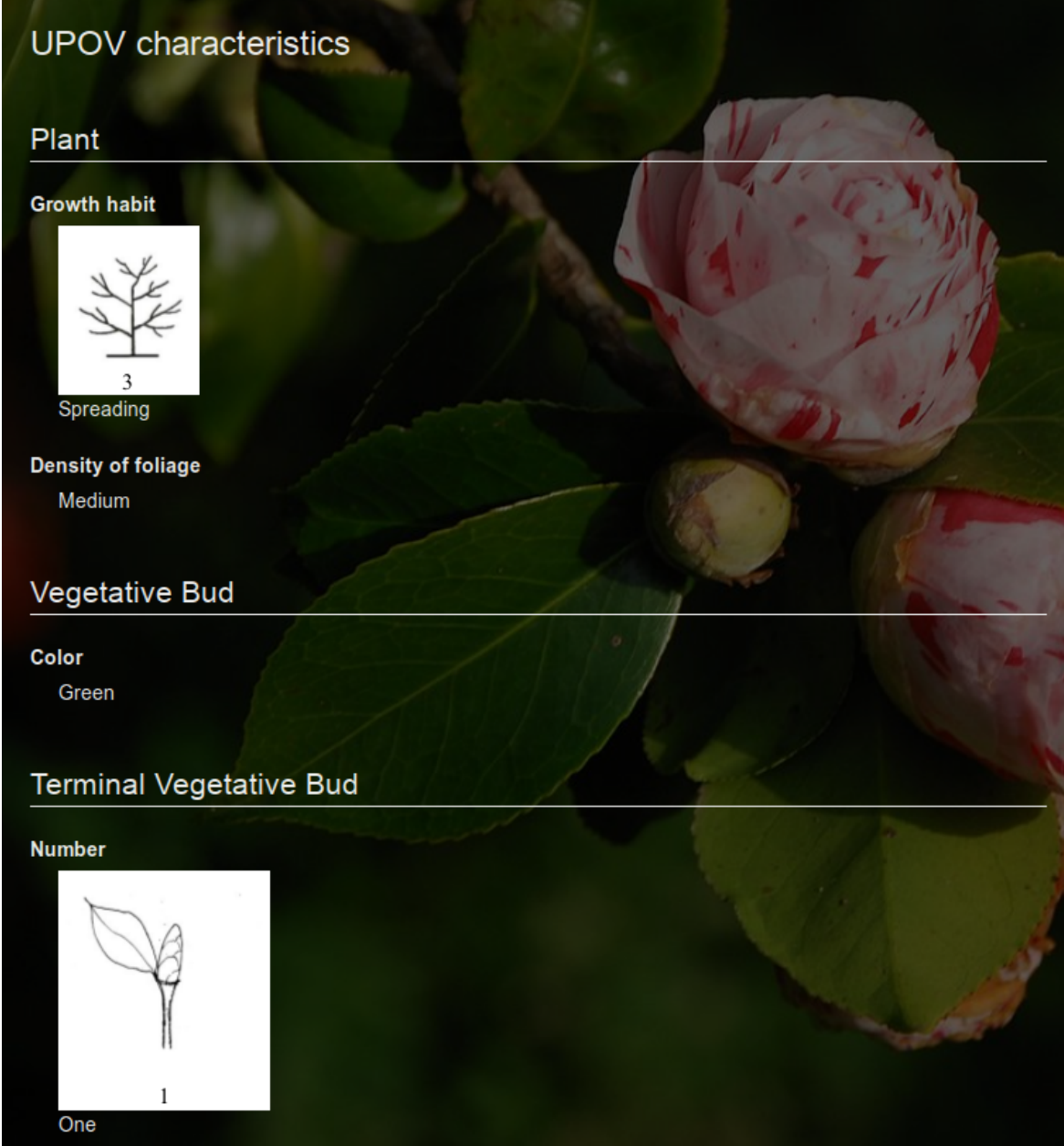
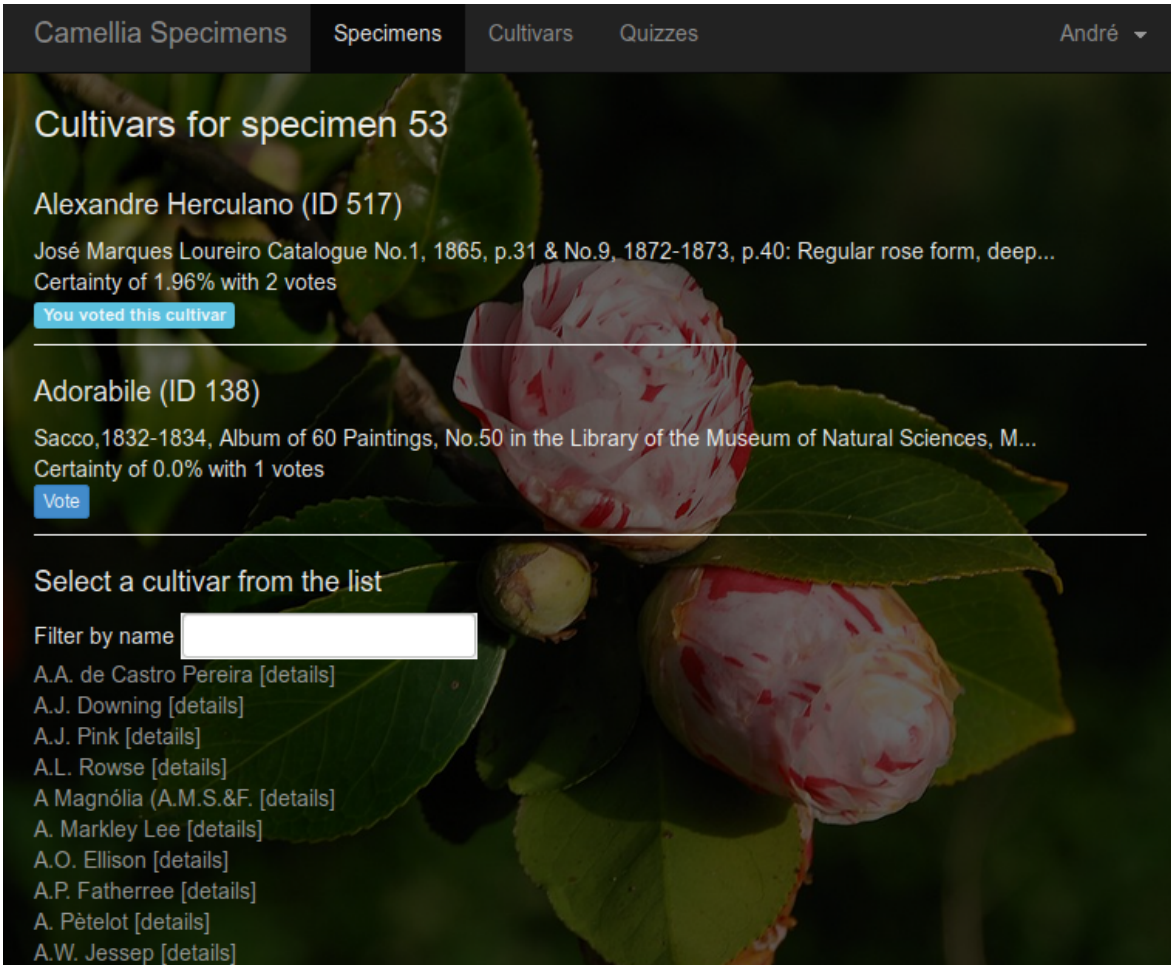


Figure 4.12: Website: specimen example (characteristics)



The screenshot shows a web interface for 'Camellia Specimens'. The navigation bar includes 'Specimens', 'Cultivars', and 'Quizzes', with a user profile 'André' on the right. The main content area is titled 'Cultivars for specimen 53' and features a background image of a pink and white camellia flower. Two cultivars are listed for voting:

- Alexandre Herculano (ID 517)**
José Marques Loureiro Catalogue No.1, 1865, p.31 & No.9, 1872-1873, p.40: Regular rose form, deep...
Certainty of 1.96% with 2 votes
[You voted this cultivar](#)
- Adorable (ID 138)**
Sacco, 1832-1834, Album of 60 Paintings, No.50 in the Library of the Museum of Natural Sciences, M...
Certainty of 0.0% with 1 votes
[Vote](#)

Below the list is a section titled 'Select a cultivar from the list' with a search filter: 'Filter by name' followed by an input field. A list of cultivars is provided, each with a '[details]' link:

- A.A. de Castro Pereira [details]
- A.J. Downing [details]
- A.J. Pink [details]
- A.L. Rowse [details]
- A Magnólia (A.M.S.&F. [details]
- A. Markley Lee [details]
- A.O. Ellison [details]
- A.P. Fatherree [details]
- A. Pételot [details]
- A.W. Jessep [details]

Figure 4.13: Website: specimen vote example

4.6.5 CULTIVARS

By accessing the cultivars page the user sees a filterable list of all the cultivars (figure 4.14) and a button to suggest a new cultivar, that will have to be approved by a moderator before appearing on the list. By selecting a cultivar the user sees its description and a list of standard specimens, the specimens which are certainly of from this cultivar (figure 4.15 on the facing page).

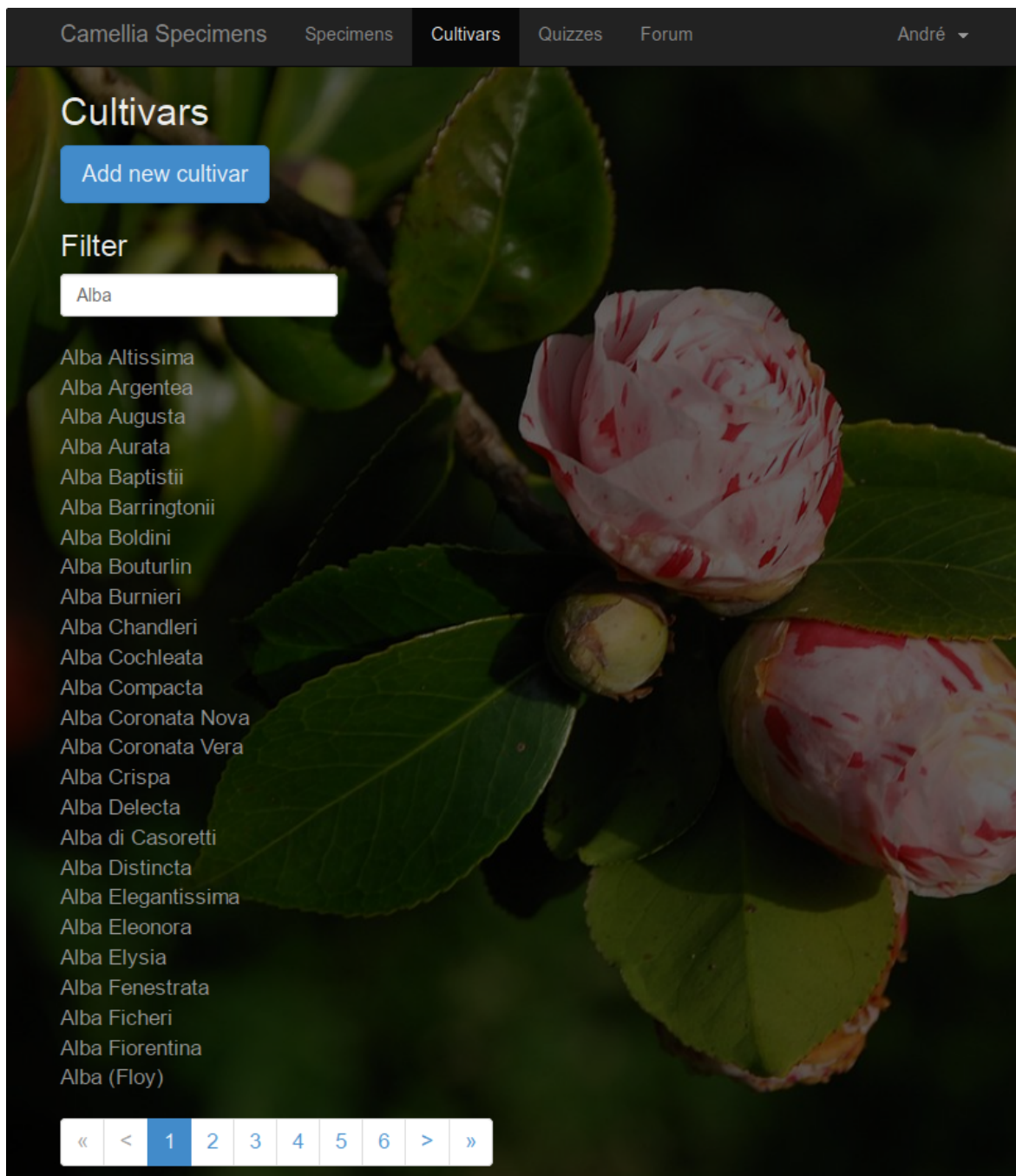


Figure 4.14: Website: cultivars list



Figure 4.15: Website: cultivar example

4.6.6 USER PROFILE

When the users access their profile page they can see some information about them, such as how many identification requests they submitted, how many quizzes they answered, how many cultivars they voted and also their current reputation (figure 4.16 on the next page). The users may edit some profile information such as the first and last names and the email address (figure 4.17 on the following page).

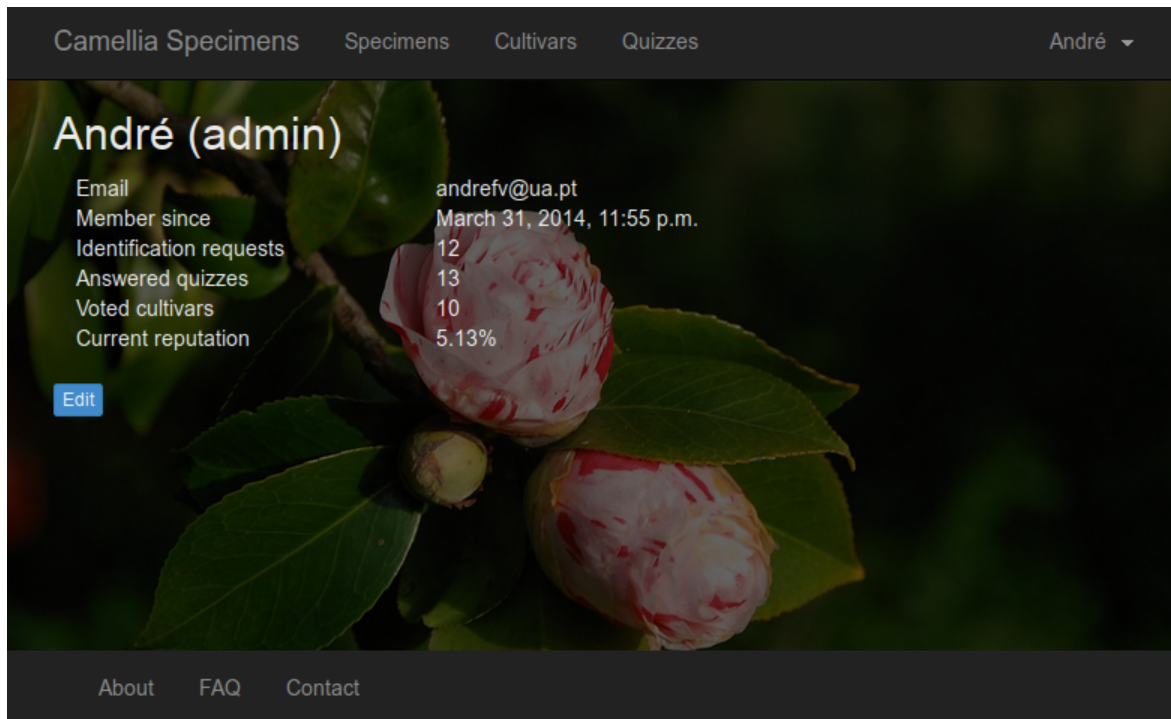


Figure 4.16: Website: user profile

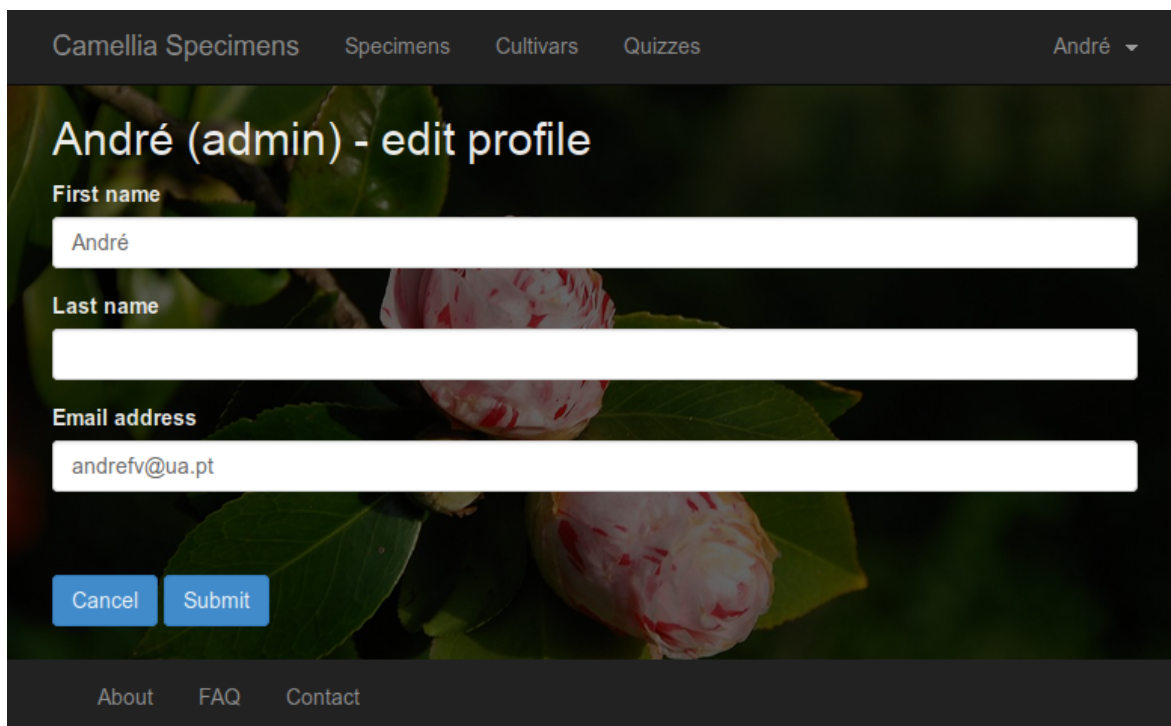
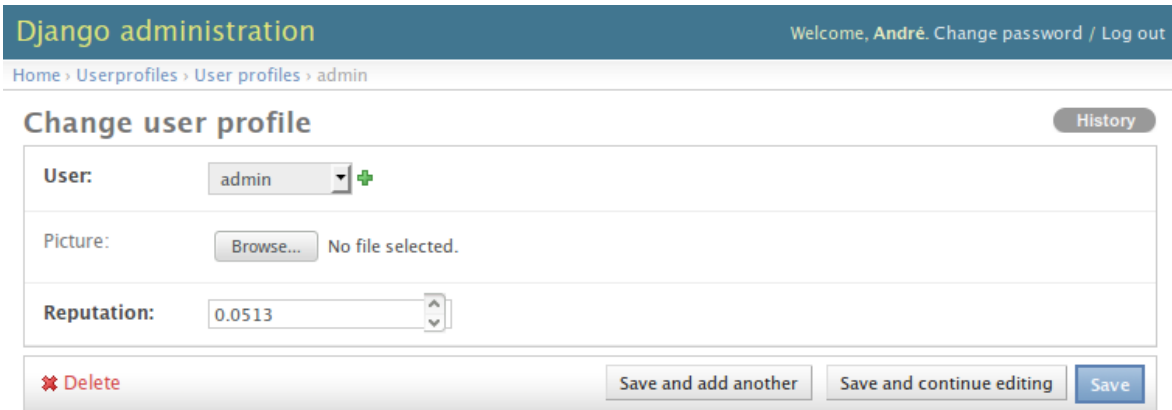


Figure 4.17: Website: edit profile

4.6.7 MANAGE USERS

The administrators manage the users through the Django’s admin interface. This interface works by reading the project models metadata and provides a powerful and production-ready interface.



The screenshot displays the Django administration interface for editing a user profile. The page title is "Django administration" and the user is logged in as "André". The breadcrumb trail is "Home > Userprofiles > User profiles > admin". The main heading is "Change user profile" with a "History" button. The form contains the following fields:

- User:** A dropdown menu showing "admin" with a plus sign to add more options.
- Picture:** A "Browse..." button and the text "No file selected."
- Reputation:** A numeric input field containing "0.0513" with up and down arrows.

At the bottom, there are four buttons: "Delete" (with a red 'x' icon), "Save and add another", "Save and continue editing", and "Save".

Figure 4.18: Website: edit user profile through Django admin site

4.6.8 ABOUT, FAQ, AND CONTACT

The about page (figure 4.19 on the next page) shows a simple text explaining the context of the website and giving credit to the Portuguese Camellia Society—which provided the web domain name and hosting—and the International Camellia Society—from where the first cultivar list came.

The FAQ page will have the frequently asked questions that will help the users getting around the site and contributing.

The contact page (figure 4.20 on page 125) has a form for users to send a message. The messages will be saved on the database and accessible through the administration page.

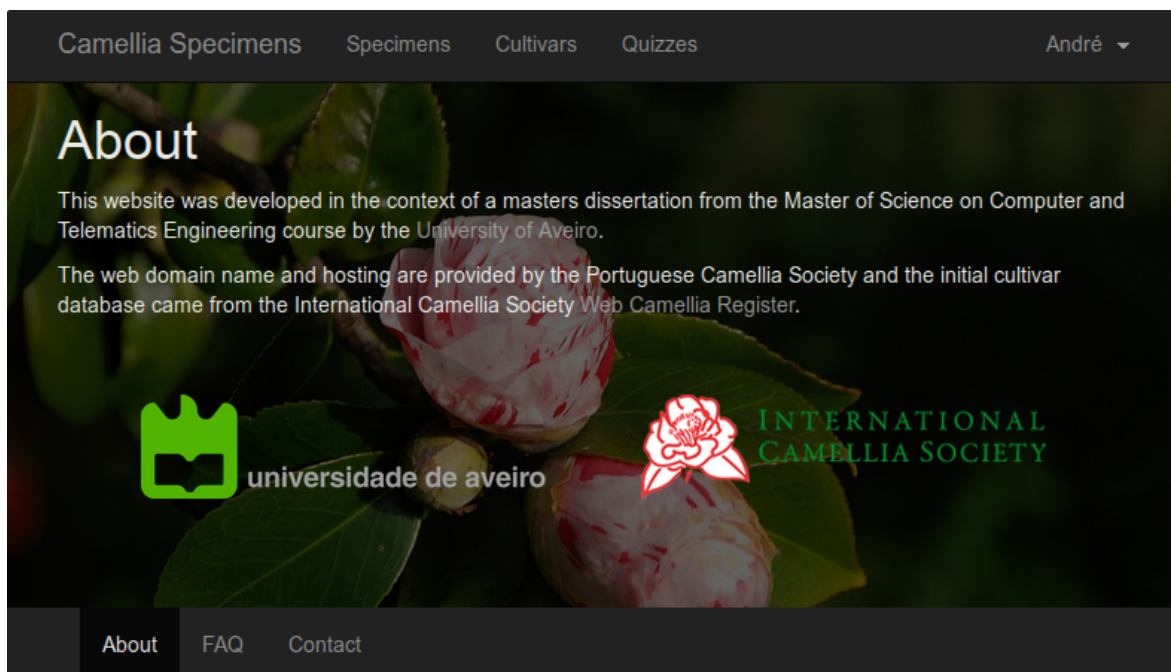
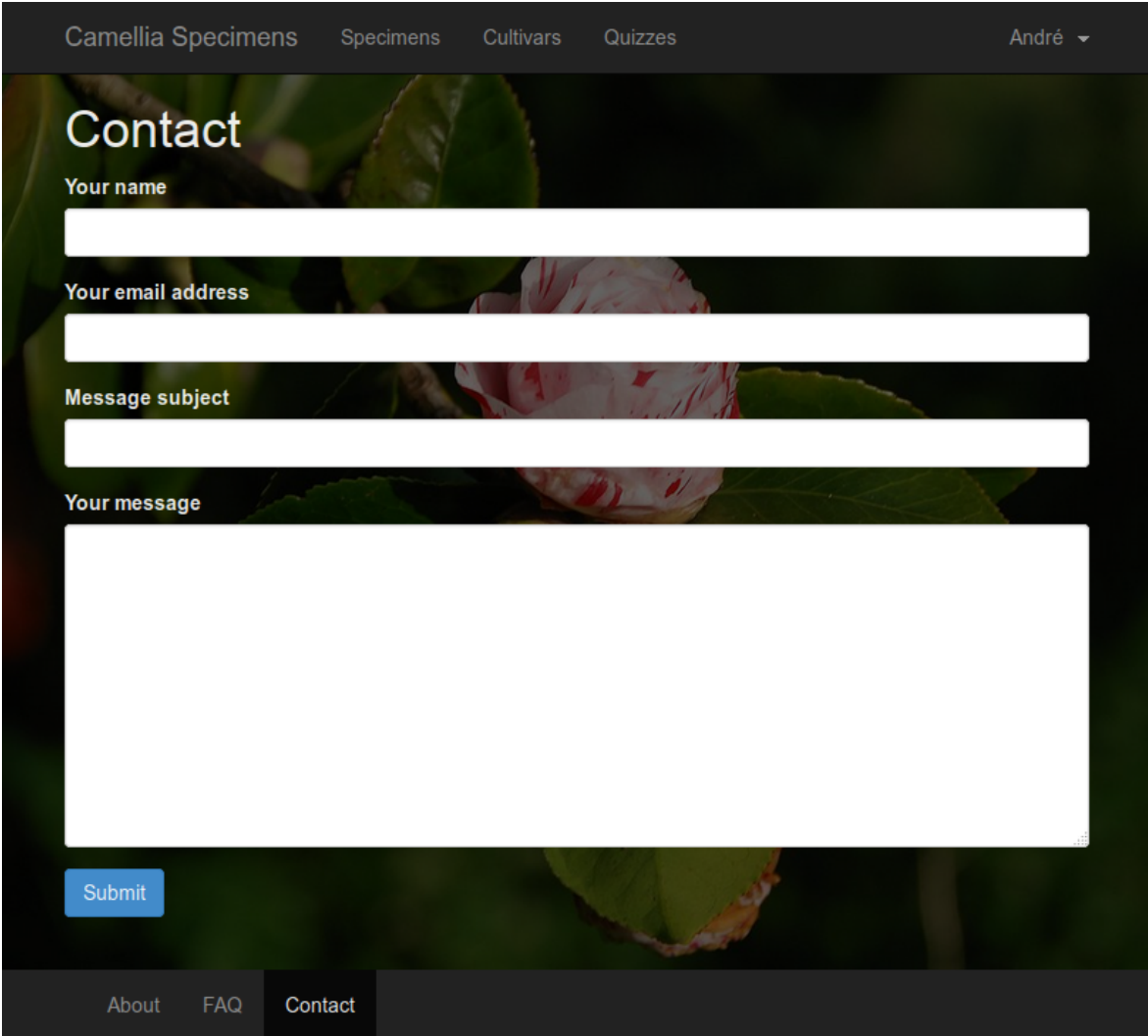


Figure 4.19: Website: about page



Camellia Specimens Specimens Cultivars Quizzes André ▾

Contact

Your name

Your email address

Message subject

Your message

Submit

About FAQ Contact

The image shows a contact form on a website. The background is a dark, close-up photograph of a camellia flower with pink and white petals and green leaves. The form is overlaid on this background. At the top, there is a dark navigation bar with the text 'Camellia Specimens', 'Specimens', 'Cultivars', 'Quizzes', and 'André' with a dropdown arrow. Below the navigation bar, the word 'Contact' is written in a large, white, sans-serif font. Underneath, there are four input fields: 'Your name', 'Your email address', 'Message subject', and 'Your message'. Each field is a simple white rectangle. Below the 'Your message' field is a blue 'Submit' button. At the bottom of the page, there is a dark footer bar with the text 'About', 'FAQ', and 'Contact' in white, with 'Contact' being the active page.

Figure 4.20: Website: contact page

4.7 INTERNATIONALIZATION AND LOCALIZATION

There are multiple techniques of creating a database for a multi-language web site. For example:

- Column approach
- Multirow approach
- Single translation table approach
- Additional translation table approach

4.7.1 COLUMN APPROACH

This is the simplest approach. It works by having an additional column for each language (example in table 4.1) but, although simple, it is hard to maintain and it is not scalable, as a new language requires a table schema update to add a new column to each multi-language table, and the website code must be updated to consider that new language and choose the right table column to use.

id	description_en	description_fr	description_de	description_es
1	plant	plante	pflanze	planta
2	branch	ramification	zweig	rama
3	vegetative bud	bourgeon	vegetative knospe	yema de madera
4	terminal vegetative bud	bourgeon végétatif	terminale vegetative knospe	yema de madera terminal
5	young shoot	jeune pousse	jungtrieb	tallo joven
6	leaf	feuille	blatt	hoja
			⋮	

Table 4.1: Multi-language database: column approach

4.7.2 MULTI-ROW APPROACH

This technique is similar to the column one and differs by using a row for each translation (example in table 4.2 on the facing page). In this case it is still hard to maintain, as inserting a new language requires cloning the record for the default language and there is duplicate content (type_id column, in this specific example).

id	type_id	lang_id	description
1	1	en	plant
2	1	fr	plante
3	1	de	pflanze
4	1	es	planta
5	2	en	branch
6	2	fr	ramification
7	2	de	zweig
8	2	es	rama
9	3	en	vegetative bud
10	3	fr	bourgeon
11	3	de	vegetative knospe
12	3	es	yema de madera
		:	

Table 4.2: Multi-language database: multi-row approach

4.7.3 SINGLE TRANSLATION TABLE APPROACH

This solution, from database structure perspective, is cleaner, as all the texts that need to be translated are stored on a single translation table and there is no duplicate content (table 4.3 on the next page). This approach also makes adding a new language easier (no schema changes), although the querying is more complex due to the multiple table joins required to retrieve each translated name.

type_id	description	id	lang_code	text	code	name
1	28	28	en	plant	en	english
2	33	28	fr	plante	fr	french
3	46	28	de	pflanze	de	german
⋮		28	es	planta	es	spanish
Characteristic Type		33	en	branch	⋮	
		33	fr	ramification	Language	
		33	de	zweig		
		33	es	rama		
		46	en	vegetative bud		
		46	fr	bourgeon		
		46	de	vegetative knospe		
		46	es	yema de madera		
		⋮				
Translation						

Table 4.3: Multi-language database: single translation table approach

4.7.4 ADDITIONAL TRANSLATION TABLE APPROACH

This technique is an improvement from the single translation table discussed above. Here, there is a translation table for each table that has information to be translated. This improves the translations maintainability and is easier to work with (requires just one table join).

Table 4.4 on the facing page shows the example for two tables—characteristic type and characteristic name—and their corresponding translation table.

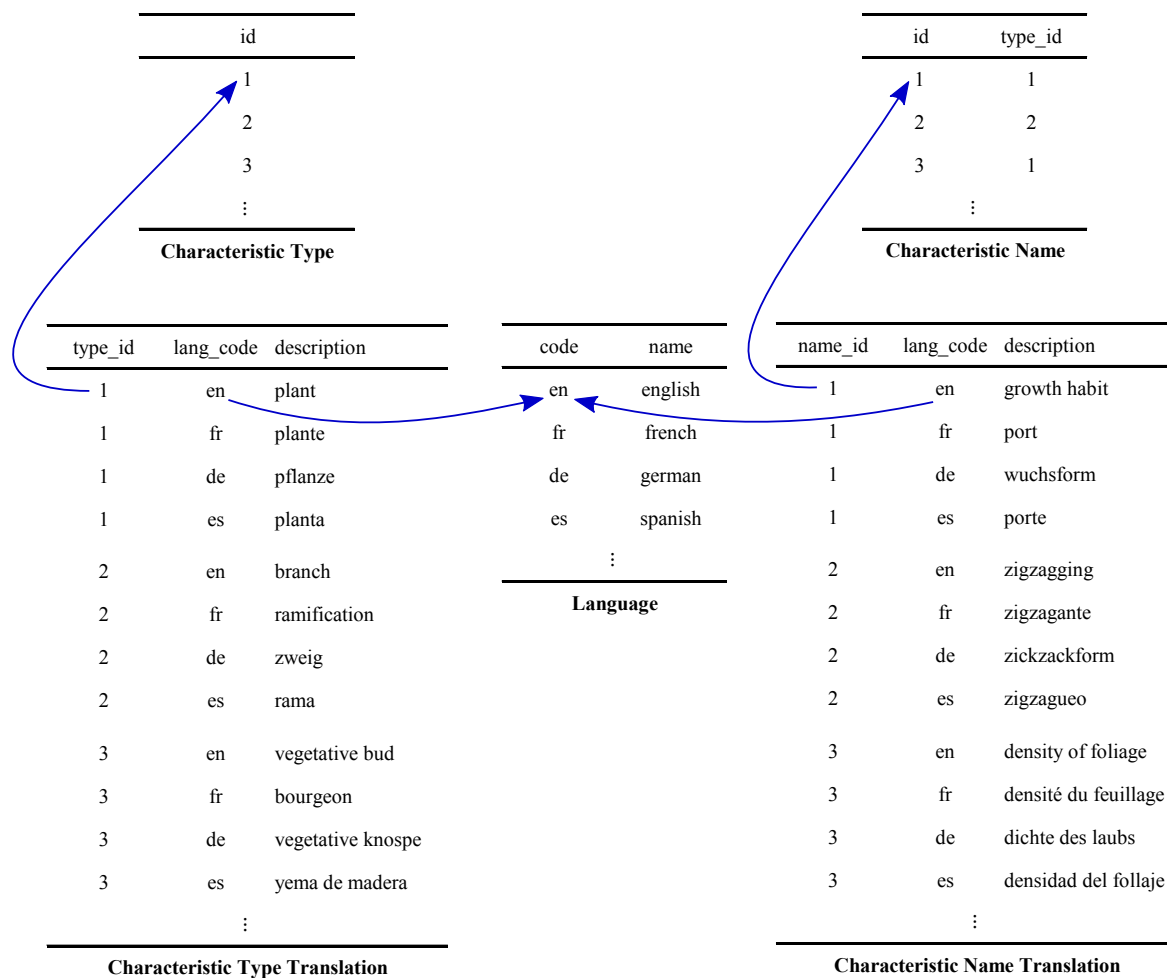


Table 4.4: Multi-language database: additional translation table approach

This approach provides proper normalization, does not require schema changes when adding a new language and there are no suffixes on column names (e.g. “_lang”, “_en”, “_fr”).

Due to this technique advantages and this project requirements, this last solution was the chosen one, making use of the *django-diplomat* application to provide the models for the countries and languages covered by the ISO 3166 and ISO 639 standards, respectively.

4.8 USER REPUTATION AND CULTIVAR PROBABILITIES

The specimen/cultivar pairs have a probability that tells how probable it is for the specimen to be of that cultivar, and that probability is estimated from the users votes and their reputation.

4.8.1 USER REPUTATION

The user reputation is indicative of how much the user knows about the camellia cultivars. Each time a user votes on a cultivar or submits a quiz, their reputation will be recalculated based on all the standard specimens the user already answered through the quizzes and based on how many votes the user has on the system.

One part of the reputation is simply the number of correct answers to the standard specimens divided by the number of total answers the user had the opportunity to respond (wrong and blank responses are considered the same), the other part being the number of votes the user has in the system, over the total votes. The reputation is normalized and presented to the user as a percentage. The weight given to each part, standard specimens answers (A_W) and votes (V_W), is configurable and can be adjusted in the future, but the first values, for testing purposes, given to each, was 0.8 and 0.2, respectively.

$$R_x = A_W \times \frac{A_{Cx}}{A_{Tx}} + V_W \times \frac{V_x}{V_T}, \quad (4.1)$$

Where:

- R_x : is the reputation of user x
- A_W : is the weight given to the user answers
- A_{Cx} : is the number of correct answers given by user x
- A_{Tx} : is the number of total answers given by user x
- V_W : is the weight given to the user votes
- V_x : is the number of votes from user x
- V_T : is the number of all user votes in the system

4.8.2 CULTIVAR PROBABILITIES

When a user reputation changes (e.g. by voting on a cultivar for a specimen, by answering a quiz), the specimens cultivar probability values—for the specimens that user already voted—must be recalculated.

The probability for a specimen/cultivar pair is calculated using the mean of the users reputations,

$$\mu = \frac{\sum u_i}{n}, \quad (4.2)$$

Where:

- μ : is the cultivar probability
- $\sum u_i$: is the sum of all user reputations for this cultivar (u_1, u_2, u_3, \dots)
- n : is the total number of votes for this cultivar

4.9 INITIAL CULTIVAR DATA

The cultivar database was pre-populated with the information from the [Web Camellia Register \(WCR\)](#), that was downloaded and transformed to be inserted into the system database.

On the [WCR](#) a cultivar page is retrieved by passing a parameter called `pippo`—containing the cultivar number—to the HTTP GET request method, such as `http://camellia.unipv.it/camelliadb2/dbwin.php?pippo=123` to retrieve the cultivar 123.

Figure 4.21 shows an example of a cultivar page. The description is one whole paragraph and, often, there are characters that should not be there, such as all the backslashes at the end of the example.

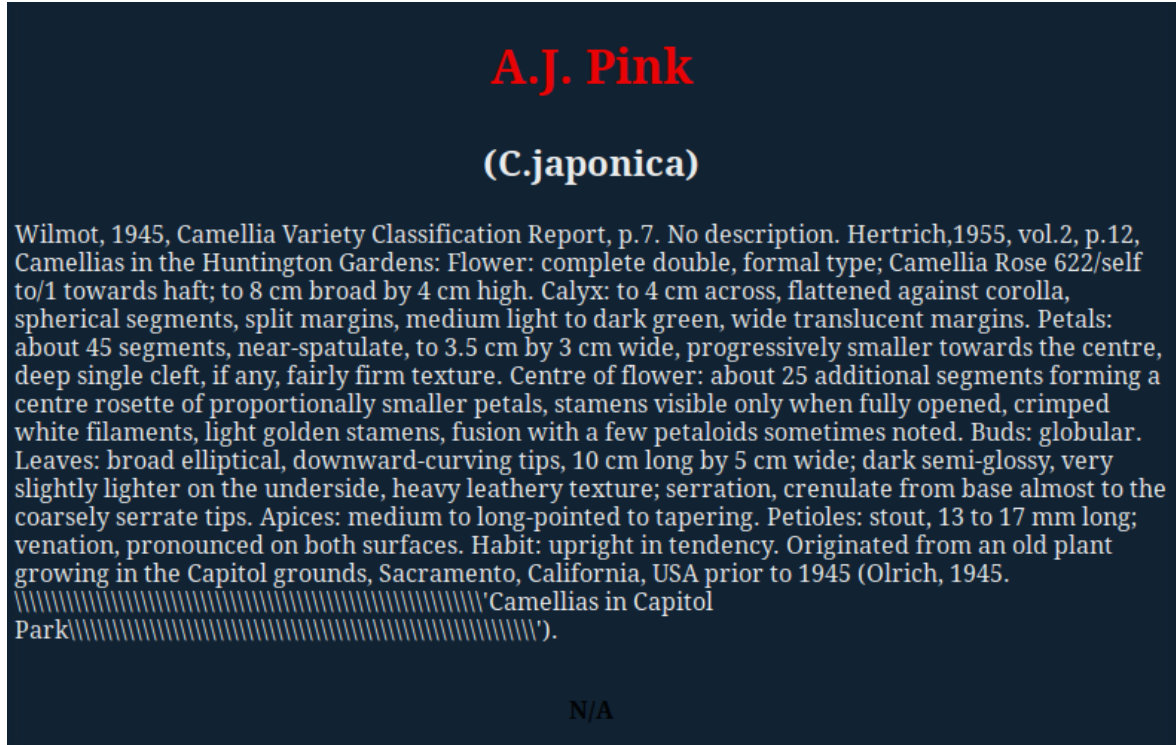


Figure 4.21: Example of a Camellia Web Register cultivar page

Listing 4.18 shows the partial source code from the same cultivar, that will be parsed later to retrieve the essential information.

Listing 4.18: Example of the code of a Camellia Web Register cultivar page

```

1           :
2 <div style='position: absolute; top: 45px; padding:0px 10px 0 10px;
  margin: 0 0 0 0; width:775px; height:509px; text-align:center;
  word-wrap: break-word; overflow:auto;'><p style='font-weight:
  bold; color:#FF0000; font-size:2em;'>A.J. Pink</p><p style='
  font-weight: bold; color:#FFFFFF; font-size:1.5em;'>(C.japonica)
  </p><p style='color:#FFFFFF; text-align:left; font-size:1.0em;'>
      Wilmot, 1945, Camellia
  Variety Classification Report, p.7. No description. Hertrich
  ,1955, vol.2, p.12, Camellias in the Huntington Gardens: Flower:
  complete double, formal type; Camellia Rose 622/self to/1 towards
  haft; to 8 cm broad by 4 cm high. ... Originated from an old
  plant growing in the Capitol grounds, Sacramento, California, USA
  prior to 1945 (Olrich, 1945.
  \\\Camellias in Capitol Park
  \\\')
  .
  </p><br /><b>N/A</b>
  </div></body>
3 </html>

```

The parsing can be done directly when downloading the page. Listing 4.19 shows the script used to parse a cultivar page. First, it saves a string ("QUOTE_HERE") to a variable so that it can be replaced at the end by real quotes, after escaping all the quotes that appear inside the cultivar description. The page is transferred from the WCR server by the *curl* tool. After downloading the page the Line Feed (\n) and Carriage Return (\r) characters are removed using the *tr* (translate or delete characters) tool, and then the useful information is extracted and structured using the *sed* (stream editor for filtering and transforming text) tool. Finally, *sed* is used again to remove all the extra spaces and backslashes, to escape the double quotes and to replace the "QUOTE_HERE" by real double quotes. Listing 4.20 on the facing page shows the parsing result. The information is clean and structured JSON-based, to be easily imported into the database.

Listing 4.19: Example of the cultivar page parsing script

```

#!/bin/sh

quote="QUOTE_HERE";

curl -s "http://camellia.unipv.it/camelliadb2/dbwin.php?pippo=3" \
| tr -d "\n" \

```

```

| tr -d "\r" \
| sed -n "s/.*font-size:2em;\x27>\(.*\)<\p>.*font-size:1.5em;\x27>\(.*\)<\p>
>.*font-size:1.0em;\x27>\(.*\)<\p><br \/><b>.*\/\{${quote}icr_id${quote}:${
quote}i${quote}, ${quote}cultivar_name${quote}:${quote}\1${quote}, ${quote}
species${quote}: ${quote}\2${quote}, ${quote}description${quote}:${quote}\3$
{quote}\n/p" \
| sed "s/\\" *\/\"/g" \
| sed "s/\" *\/\"/g" \
| sed "s/\\\\\\*\/g" \
| sed "s/\\"/\\\\\\\"/g" \
| sed "s/${quote}/\"/g";

```

Listing 4.20: Example of the cultivar page parsing result

```

1 {"icr_id": "3",
2  "cultivar_name": "A.J. Pink",
3  "species": "C.japonica",
4  "description": "Wilmot, 1945, Camellia Variety Classification Report
, p.7. No description. Hertrich,1955, vol.2, p.12, Camellias in
the Huntington Gardens: Flower: complete double, formal type;
Camellia Rose 622/self to/1 towards haft; to 8 cm broad by 4 cm
high. ... Originated from an old plant growing in the Capitol
grounds, Sacramento, California, USA prior to 1945 (Olrich, 1945.
'Camellias in Capitol Park')."}

```

The script was improved (listing 4.21) to receive two arguments, used as the first and last cultivar number to be replaced at the [Uniform Resource Locator \(URL\)](#). For example, if it is run using 50 and 53 as arguments (e.g. `sh get_cultivars.sh 50 53`), it will loop through the [URLs](#) as follows:

```

http://camellia.unipv.it/camelliadb2/dbwin.php?pippo=50
http://camellia.unipv.it/camelliadb2/dbwin.php?pippo=51
http://camellia.unipv.it/camelliadb2/dbwin.php?pippo=52
http://camellia.unipv.it/camelliadb2/dbwin.php?pippo=53

```

Listing 4.21: Example of the cultivar page parsing script using a loop

```

#!/bin/sh

quote="QUOTE_HERE";

for ((i=$1;i<=$2;i++)); do
curl -s "http://camellia.unipv.it/camelliadb2/dbwin.php?pippo=$i" \
:
| sed "s/${quote}/\"/g";
done

```

The **WCR** has 22 168 cultivars, which were saved at a file using the script. Then, using a python script, only the cultivar names were extracted to a new file just to check whether there was repeated names. It was found that 559 cultivars had more than one entry, with, at most, three cultivars with 5 entries and four cultivars with 6 entries. Listing 4.22 shows the example of the cultivar named *Akebono. (Dawn)*, which appears in 6 entries with different descriptions.

Listing 4.22: Example of the duplicate cultivar names

```

1 {"icr_id": "271", "cultivar_name": "Akebono. (Dawn)", "species": "
   C.japonica", "description": "Chinka Zufu, (before 1700), Watanabe
   , 1969, plate 40: A medium sized, open peony, white with
   irregular petals and intermixed stamens. Originated in Japan. (
   Believed extinct.)"}
2 {"icr_id": "272", "cultivar_name": "Akebono. (Dawn)", "species": "
   C.japonica", "description": "Chinka Zutu, (before 1700), Watanabe
   , 1969, plate 579: Medium size, white semi-double with two rows
   of petals and a central column of stamens with white filaments.
   Originated in Japan. (Believed extinct.)"}
3 {"icr_id": "273", "cultivar_name": "Akebono. (Dawn)", "species": "
   C.japonica", "description": "Minagawa, 1931, Chinkashû; Wada,
   1941, Japanese Garden Treasures, p.28: Single, widely opened,
   large flowers, very pale, flesh pink, slightly paler at edge and
   base, a leafy, compact grower; early blooming. ... "}
4 {"icr_id": "274", "cultivar_name": "Akebono. (Dawn)", "species": "
   C.sasanqua", "description": "Itô Ihei, 1695, Kadan Chikinshô in
   the section on sasanquas: Medium size, pale pink, red at the
   base. Originated in Japan. (Believed extinct.)"}
5 {"icr_id": "275", "cultivar_name": "Akebono. (Dawn)", "species": "
   C.sasanqua", "description": "Shirai-Bunko, 1789, Shoshiki
   Hanagatachô: White with pink edges, medium size, some stripes.
   Originated in Japan. (Believed extinct.) See: Kyôto Engei Kurabu,
   1964, Tsubaki Tokushû, No.5, p.82."}
6 {"icr_id": "276", "cultivar_name": "Akebono. (Dawn)", "species": "
   C.sasanqua", "description": "ICS., Apr.1990. Japanese Camellia
   Cultivar List, p.32: Medium size, white ground with light red
   reverse, semi-double. Originated in Kumamoto Prefecture, Japan. A
   Higo sasanqua selected and named by Kiyofusa Saitô."}

```

The disambiguation of these repeated names was made by adding a number to the title (e.g. *Akebono. (Dawn) (2)*, *Akebono. (Dawn) (3)*) when saving them to the database through the Python script shown in listing 4.23 on the next page. This script was imported and run inside the Django shell so that it makes use of the Cultivar model to save each cultivar which was read from the file containing all the cultivars downloaded. The `save()` method throws a

IntegrityError if the name already exists and that is when the script adds (or increments) the number to the cultivar name and tries to save it again.

Listing 4.23: Python script for saving the cultivars to the database

```
1 import json
2 from cultivars.models import Cultivar
3 from django.db.utils import IntegrityError
4
5 def save_cultivars():
6     with open('all_cultivars') as f:
7         for line in f:
8             json_line = json.loads(line)
9             c = Cultivar()
10            c.user_id = 1
11            c.name = json_line['cultivar_name']
12            c.icr_id = json_line['icr_id']
13            c.icr_description = json_line['description']
14            c.tmp_icr_species = json_line['species']
15
16            saved = False
17            count = 1
18            while not saved:
19                try:
20                    c.save()
21                    saved = True
22                except IntegrityError as e:
23                    error = e.__str__()
24                    if "DETAIL: Key (name)=" in error:
25                        count += 1;
26                        if count > 2:
27                            c.name = c.name[:-4] # remove the
28                                previous count
29                            c.name += " (" + count.__str__() + ")"
```

4.10 REVISION CONTROL

A revision control system is a repository of files, often the files for the source code of computer programs, with monitored access. This is important because it provides the ability to revert the files to a previous version (e.g. in case of a mistake), provides the ability to have many people working on the same project and knowing that conflicting modifications can be detected and resolved, and it also provides the ability to track the evolution and to see the

comments about the intention behind each change. Even when the development is made by only one person, like this project, the change history is an important aid to memory.

There are three models of revision control systems: local, client-server, and distributed, the last two being the most used. In the local model all developers must use the same file system. In the client-server model, developers use a shared single repository. The Apache Subversion (SVN) is one of the most known client-server model software. In the distributed model, each developer works directly with their own local repository, and the changes are shared between repositories as a separate step. The most known software for this model are Git, Mercurial, and Bazaar.

For this project was used Git, a free and open source distributed version control system. Figure 4.22 shows an example of some commits made. Each one then allows to see which files were modified and, on each file, what was modified.

Message	Date	Author
Improve pagination (don't show all the page numbers at on...	05/08/2014 05:04 PM	André Ventura
Allow the user to vote on a cultivar for a given specimen.	05/08/2014 01:27 AM	André Ventura
Allow the filtering of the cultivar list by cultivar name.	05/08/2014 01:26 AM	André Ventura
Add new model UserSpecimenCultivarVote at the specimens...	05/08/2014 01:25 AM	André Ventura
Add the latest cultivars to the main page. Reduce the site wid...	05/08/2014 01:21 AM	André Ventura
Add cultivars views, forms, and templates.	04/30/2014 03:27 AM	André Ventura
Specimen details page: add an "edit" button. Specimens list ...	04/28/2014 03:52 AM	André Ventura
Add a map to the specimen's page.	04/28/2014 03:17 AM	André Ventura
Initial work with cultivars.	04/28/2014 02:51 AM	André Ventura
Fix specimens' tests (user ID and authentication)	04/28/2014 02:50 AM	André Ventura
Require the user to be authenticated when submitting a new ...	04/19/2014 03:41 PM	André Ventura
Use a transaction while saving the specimen identification re...	04/19/2014 04:10 AM	André Ventura
Add specimen photo gallery model using django-photologu...	04/19/2014 03:21 AM	André Ventura
Fix Google Maps' map loading problems and the initial locati...	04/14/2014 02:36 AM	André Ventura

Figure 4.22: Example of Git commits

There are some web-based hosting service repositories, like GitHub, that provide access control and several collaboration features (e.g. wikis, task management, bug tracking, feature requests), and serve also as a backup. However, this project was maintained on a local Git repository with backups made with another tool.

4.11 DOMAIN NAME AND WEB HOSTING

In order to be accessible to everyone, the system needs a domain name and space on a server that supports the chosen technologies discussed in section 2.4 on page 34.

4.11.1 DOMAIN NAME

The domain name *camelias.pt* was registered by the Portuguese Association of Camellias (Associação Portuguesa das Camélias) after choosing one of the 154² existing resellers of the .pt **Top Level Domain (TLD)**, managed by the **Fundação para a Computação Científica Nacional (FCCN)**³, which registrar is Associação DNS.PT⁴. The chosen reseller was PT-Servidor⁵, a company born in 2006 and an official reseller since 2010 with positive customer reviews about the technical support and cost.

4.11.2 WEB HOSTING

Regarding the web hosting, the following types of hosting were considered:

- Home server (dedicated server at home)
- Shared hosting (virtual hosting)
- **Virtual Private Server (VPS)**
- Managed hosting
- Dedicated server

The home server approach would require, if we want the website reliably online and available to visitors, a static **Internet Protocol address (IP address)**, but most **Internet service providers (ISPs)** refuse to provide it on a home (non enterprise) plan, although it could be circumvented through the use of a **Dynamic DNS (DDNS)** service, a method used to resolve a well-known domain name to an **IP address** that may change frequently. We would also need to take into account the electricity costs to keep the system always on. For example, in Portugal the cost per month for a server with approximately 100 W of electrical power (0.1 kWh of electrical energy) would be around EUR 13.53 ($0.1 \text{ kWh} \times 24 \text{ hours} \times 30 \text{ days} \times \text{EUR } 0.1879^6 = \text{EUR } 13.53$).

On shared hosting, also called virtual hosting, the customer's website is placed on the same server as many other sites and most providers have limits to what can be installed in

²<https://www.dns.pt/en/registrars-list>

³<http://www.fccn.pt/>

⁴<https://www.dns.pt/>

⁵<https://www.ptservidor.pt/>

⁶Energy price on most suppliers of the Portuguese free market per 2014: $\text{EUR } 0.1528 \times 1.23$ (23% VAT) = EUR 0.1879

terms of resource allocation, including limits on what software may be used, although some of the shared hosting support Django.

In a managed hosting service the customers gets their own web server but have only partial control over it and, usually, are only allowed to manage their data via **File Transfer Protocol (FTP)** or other remote management tools, meaning that typically there is no access by the customers to the software configuration.

A dedicated hosting service is a web server devoted entirely to the customer. A **VPS** hosting involves a dedicated server being shared by multiple users (multiple virtual servers on the same web server), but as the server space is strictly divided and private, it is seen by the users as a dedicated server. On both, the customers access their own space and, generally, are responsible for maintaining the server software such as the required configuration and upgrades. Any **VPS** or dedicated host, by its nature, should work with Django.

The memory, hard drive storage capacity and processing power required by this project is satisfied by a shared hosting service but it would be necessary to exist some freedom managing the database and configuration both during development and afterwards (possible on a **VPS** or dedicated server) so the chosen type of hosting was a mixture of the shared hosting with the dedicated hosting. The WebFaction⁷ company offers a shared hosting service that has many of the benefits of a **VPS** or dedicated server (**VPS** like features for shared hosting prices), thus having the best ratio of prices to service.

⁷<https://www.webfaction.com/>

SUMMARY & FURTHER WORK

5.1 TESTS AND VALIDATION

The system was populated with a small-scale dummy data set with the main goal of validating and comparing the statistical metrics applied, and is being fed with real data through the collaboration of volunteer Botany students filling-in identification requests as completely as possible. Contacts were made with some top camellia gardens to try and import a core of reliable data, including as many standard specimens as possible (especially of Portuguese origin), but this validation with real data is still on-going work.

5.2 SUMMARY

The goal of this project, to develop a crowdsourcing information system for camellia cultivar identification, given the project requirements outlined in chapter 3 on page 37, was achieved, whilst leaving room for improvement and development upon the foundation that has been laid.

5.3 FURTHER WORK

The system was implemented according to the design and requirements but there is always room for improvement. It has to be tested with more users and the reputation metrics must be analysed and adjusted as needed. Some of other possible areas for expansion are detailed in this section.

5.3.1 FORUM

Cultivar identification is not black and white and generates a lot of discussion. For this reason, an Internet Forum—or discussion board—would be helpful and could push the users to collaborate more. There are some free and professional grade forum software packages, such as the Simple Machines¹ that allows to configure a forum easily and allows the integration with the existing users on this system.

5.3.2 MOBILE APPLICATION

Although the website is designed to be mobile friendly, a native mobile app would take advantage of the mobile phone camera and GPS location to help the user send identification requests more easily. It would also enable the users—visiting a garden—to use the GPS real-time location and be guided through the specimens on that place improving their visit and satisfaction.

5.3.3 AUTO-IDENTIFICATION OF CULTIVARS

As the system is populated with specimens and when there are plenty of standard specimens, a possibility of auto-identification arises. If it is known—for the standard specimens—which characteristics they have, then when a user fills the characteristics of a new specimen the system would automatically rule out the cultivars which would be an impossible match due to the differences in the characteristics.

¹<http://www.simplemachines.org/>

BIBLIOGRAPHY

- [1] B. Robson, “On the naming of 19th century camellias”, *International Camellia Journal*, no. 40, pp. 44–49, 2008.
- [2] I. C. Society. (). Scheme for the recognition of international camellia gardens of excellence, [Online]. Available: <http://www.internationalcamellia.org/> (visited on 10/28/2012).
- [3] —, (). International Camellia Register, [Online]. Available: <http://www.internationalcamellia.org/international-camellia-register> (visited on 02/13/2014).
- [4] —, (). Web Camellia Register, [Online]. Available: <http://camellia.unipv.it/camelliadb2/> (visited on 02/13/2014).
- [5] J. L. Couselo, P. Vela, C. Salinero, and M. J. Sainz, “Characterization and differentiation of old *Camellia japonica* cultivars using single sequence repeat (SSRs) as genetic markers”, *International Camellia Journal*, no. 42, pp. 117–122, 2010.
- [6] M. F. Stoner, “Identification, history, cultivation, and conservation of heritage camellias in Hawaii”, *International Camellia Journal*, no. 42, pp. 46–49, 2010.
- [7] F. S. Crowder, “A personal search for pre-1900 camellia cultivars and their preservation”, *International Camellia Journal*, no. 43, pp. 45–46, 2011.
- [8] U. Universitet. (2010). Linné on line, [Online]. Available: <http://www.linnaeus.uu.se/online/index-en.html> (visited on 10/18/2014).
- [9] L. D.-V. Ryssel and R. D. Herdt, *De camellia. Een’aristocratische roos (The Camellia. An Aristocratic Rose)*. Gent: MIAT - Museum over industrie, arbeid en textiel, 2008.
- [10] I. C. Society. (). Otomo Endowment Research Fund, [Online]. Available: <http://www.internationalcamellia.org/the-ics-otomo-fund> (visited on 10/18/2014).
- [11] Camellias.pics. (). Camellia’s history, [Online]. Available: <http://www.camellias.pics/histoire-gb.php> (visited on 10/18/2014).
- [12] I. C. Society. (). The International Camellia Society Portugal Region, [Online]. Available: <http://www.internationalcamellia.org/ics-portugal> (visited on 10/18/2014).
- [13] T. Nursery and G. I. A. of New Zealand. (). Science behind your garden - Ornamental - Camellias, [Online]. Available: <http://www.gardenscience.co.nz/ornamental/TGuides/camellias.htm> (visited on 10/17/2014).
- [14] J. Howe. (Jun. 2, 2006). Crowdsourcing: a definition, [Online]. Available: http://crowdsourcing.typepad.com/cs/2006/06/crowdsourcing_a.html (visited on 01/02/2014).
- [15] (2014). Treezilla - the monster map of trees, [Online]. Available: <http://treezilla.org> (visited on 02/13/2014).

- [16] (2014). OpenTreeMap Cloud - tree map for engaging communities and managing urban ecosystems, [Online]. Available: <https://www.opentreeemap.org/> (visited on 02/13/2014).
- [17] F. R. Farmer and B. Glass, *Building Web Reputation Systems*. O'Reilly Media, Inc., 2010.
- [18] O. Roy Sarkar. (). Open source is secure, but only with proper tools and strategies, [Online]. Available: <http://www.openlogic.com/resources/enterprise-blog/november-2014/open-source-is-secure,-but-only-with-proper-tools> (visited on 10/12/2014).
- [19] P. Katherine Noyes. (). 10 reasons open source is good for business, [Online]. Available: www.pcworld.com/article/209891/10_reasons_open_source_is_good_for_business.html (visited on 10/12/2014).
- [20] D. S. Foundation. (). Django, the web framework for perfectionists with deadlines, [Online]. Available: <https://docs.djangoproject.com/en/1.7/intro/overview/> (visited on 08/21/2014).
- [21] Instagram. (). What Powers Instagram: Hundreds of Instances, Dozens of Technologies, [Online]. Available: <http://instagram-engineering.tumblr.com/post/13649370142/what-powers-instagram-hundreds-of-instances> (visited on 12/13/2014).
- [22] P. Sciarra. (). What is the technology stack behind Pinterest?, [Online]. Available: <http://www.quora.com/What-is-the-technology-stack-behind-Pinterest-1> (visited on 12/13/2014).
- [23] T. P. G. D. Group. (). About PostgreSQL, [Online]. Available: <http://www.postgresql.org/about/> (visited on 08/21/2014).
- [24] PostgreSQL. (). PostgreSQL: Featured Users, [Online]. Available: <http://www.postgresql.org/about/users/> (visited on 12/13/2014).
- [25] I. U. for the Protection of New Varieties of Plants. (Oct. 20, 2011). Guidelines for the conduct of tests for distinctness, uniformity and stability - Camellia L. - TG/275/1, [Online]. Available: <http://www.upov.int/edocs/tgdocs/en/tg275.pdf> (visited on 02/13/2014).
- [26] J. Li, S. Ni, X. Li, X. Zhang, and J. Gao, "Developing the international test guideline of distinctness, uniformity and stability for ornamental camellia varieties", *International Camellia Journal*, no. 40, pp. 112–118, 2008.
- [27] A. Hunt and D. Thomas, *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional, 1999, ISBN: 0-201-61622-X.
- [28] D. S. Foundation. (). Django at a glance, [Online]. Available: <https://docs.djangoproject.com/en/1.7/intro/overview/> (visited on 08/21/2014).
- [29] S. Burbeck, *How to use Model-View-Controller (MVC)*. Smalltalk Webring, 1992.
- [30] A. Godwin. (). About South, [Online]. Available: <https://south.readthedocs.org/en/latest/about.html> (visited on 08/21/2014).
- [31] D. S. Foundation. (). Introduction to class-based views, [Online]. Available: <https://docs.djangoproject.com/en/1.7/topics/class-based-views/intro/> (visited on 08/21/2014).
- [32] —, (). The Django template language, [Online]. Available: <https://docs.djangoproject.com/en/1.7/topics/templates/> (visited on 08/21/2014).
- [33] Bootstrap. (). Bootstrap, [Online]. Available: <http://getbootstrap.com/> (visited on 08/21/2014).
- [34] D. S. Foundation. (). Working with forms, [Online]. Available: <https://docs.djangoproject.com/en/1.7/topics/forms/> (visited on 08/21/2014).