University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering

Master's Thesis

EEG/ERP Portal Security in New Technologies

Pilsen, 2012 Jiří Novotný

Declaration of Authorship

Abstract

Security needs to be assured in EEG/ERP Portal for technical and legal reasons. The application stores sensitive data and has to be resistant against malicious actions. This thesis describes improving security by using features introduced in new technologies and by patching exploitable weaknesses. First, background information including legal aspects, project description and security principles are provided. Then the process of technology migration is described, including tools introduced to enable the transition. Following a security analysis, the authentication process is restructured and revealed authorization shortcomings are fixed. The final configuration is tested and evaluated to make sure the portal is suitable for wide use.

Contents

1. Introduction **1**

I. Theoretical Background **2**

2. Legal Aspects of Storing Neuroinformatic Data **3**

2.1. EU Directive . . . 3

2.1.1. Special Categories . . . 4

2.1.2. Data Controllers . . . 5

2.1.3. Security Of Processing . . . 5

2.2. Czech Law Definitions . . . 6

2.3. Personal Data Protection in the U.S. . . . 7

3. EEG/ERP Portal **8**

3.1. Features . . . 8

3.2. Users . . . 8

3.3. Architecture . . . 9

3.3.1. Data Tier . . . 9

3.3.2. Application Tier . . . 10

3.3.3. Presentation Tier . . . 10

4. Authentication and Authorization Solutions **12**

4.1. Authentication . . . 12

4.1.1. Remember Me . . . 13

4.1.2. Password Storage . . . 13

4.1.3. Third Party Login . . . 14

4.2. Authorization . . . 14

4.2.1. Page-level Security . . . 15

4.2.2. Individualized Views . . . 15

5. Frameworks Used in EEG/ERP Portal **17**
5.1. Spring . . . 17
5.1.1. New Features in Spring 3.0 . . . 18
5.1.2. Backward Compatibility . . . 18
5.2. Spring Security . . . 19
5.2.1. New Features . . . 19
5.2.2. Backward Compatibility . . . 20
5.3. Hibernate . . . 20

II. Design and Implementation **22**

6. Migration Plan **23**
6.1. Blocks to a Direct Update . . . 23
6.1.1. Library Structure . . . 23
6.1.2. Absence of Tests . . . 24
6.2. Introducing Maven . . . 24
6.2.1. Tool Description . . . 26
6.2.2. Artifact Repositories . . . 27
6.3. Migration Steps . . . 28

7. Technology Update **29**
7.1. Project Directory Structure Changes . . . 29
7.2. Unifying Dependencies . . . 30
7.3. Storing Custom Libraries . . . 31
7.4. Porting JSON Plugin . . . 33
7.5. Building in Major IDEs . . . 33
7.6. Command Line Support . . . 34

8. Analyzing Security Loopholes **35**
8.1. Password Storage . . . 35
8.2. Error Stack Traces . . . 35
8.3. Unverified Address Inputs . . . 36
8.4. Transport Layer Security . . . 36
8.5. Authorization Issues . . . 36
8.6. Cookies Accessible to Scripts . . . 37

9. Restructuring Authentication Mechanism 38
9.1. Unifying Account Manipulation 38
9.2. Securing Database Password Storage 38
9.3. Email as Primary Identification 40
9.4. Single Login Page 41
9.5. Removing Plain Text Passwords 42

10. Fixing Authorization and Mitigating Risks 43
10.1. Page Access 43
10.1.1. Proposed Design Improvement 43
10.2. Conditional Rendering 44
10.3. Protecting URL Parameter Input 45
10.4. Specifying Default Error Behavior 45
10.5. Transport Layer Security 46
10.6. HTTPOnly Cookies 46

11. Testing 47
11.1. Test Types 47
11.2. Unit Tests 48
11.2.1. Fixing Existing Tests 48
11.2.2. New JUnit Tests 50
11.3. Integration Tests 50
11.3.1. Mechanism 51
11.3.2. Testing Authentication Setup 51
11.3.3. User Authorization Tests 52

12. Evaluating Application Security 53
12.1. Web Application Security Scanners 53
12.2. Comparing with Previous Evaluation 55

13. Conclusion 57

List of Abbreviations 58

Bibliography 60

CD Contents 63

1. Introduction

Security is of key importance in web applications. That is particularly true for a publicly accessible system storing personal and medical data, like EEG/ERP Portal developed at the University of West Bohemia. The application is designed to manage measurement and experiment data created while using electroencephalography (EEG) and event related potentials (ERP) in attention research.

The project is in active development for three years. Legal compliance, security and resistance against common attacks were already evaluated in a thesis by Jiří Vlašimský [1]. However, due to time constraints and technical limitations, not all possible measures for improving security were taken and new issues were revealed since. This thesis starts where the last one left off. Proposed changes form the basis for new tasks to be done, while other objectives arise from the current needs.

Two main goals are to be met: Upgrading project technologies and enhancing security design. Update of frameworks used by EEG/ERP Portal alone would potentially increase resistance against common threats, as vulnerabilities were found in older versions of the libraries. But possibly more importantly, new technology solutions were introduced in the current versions. Those can be used to improve application security, in some cases even rendering previously dangerous attacks impractical.

The thesis will first evaluate aspects of storing neuroinformatic data under the legislation of European Union with emphasis on the Czech law, and the same in the United States. Then the EEG/ERP Portal application is introduced. Another chapter will describe authentication and authorization solutions available in frameworks utilized by the project. Also current generations of the frameworks will be introduced, emphasizing new features. The realization phase will focus on migrating to new technologies, analyzing security shortcomings, improving authentication, authorization and other security features, testing and evaluating the results.

Part I.

Theoretical Background

2. Legal Aspects of Storing Neuroinformatic Data

Large amounts of data are produced during research in the field of neurology. Restrictions and limitations exist for data analyzing and processing, the nature of which is not only technical, but also legal. As personal details and medical records are used, data handling must be compliant with personal data protection laws. EEG/ERP Portal introduced later in chapter 3 is an application storing this kind of information. It is therefore important to ensure compatibility with the legislative in personal data protection and other aspects.

A thorough legal analysis of storing medical data is both outside the thesis author's professional abilities, and outside the scope of this chapter. With that in mind, only selected paragraphs relevant for the described domain will be cited to clarify, what is defined as personal and sensitive information, what actions must be taken before such information is processed and what rules need to be followed when storing the data.

Legal aspects will be discussed using an EU directive implemented in the legal systems of member countries and by listing applicable Czech laws. Practices based in U.S. legislation will be included for comparison.

2.1. EU Directive

Directive 95/46/EC on the protection of individuals with regard to the processing of personal data and on the free movement of such data offers a definition of the personal information [2, Article 2]:

> *'Personal data' shall mean any information relating to an identified or identifiable natural person ('data subject'); an identifiable person is one*

who can be identified, directly or indirectly, in particular by reference to an identification number or to one or more factors specific to his physical, physiological, mental, economic, cultural or social identity

Personal data may be processed only if certain conditions are met. Among others, that can be when [2, Article 7]:

The data subject has unambiguously given his consent; or

Processing is necessary for the purposes of the legitimate interests pursued by the controller or by the third party or parties to whom the data are disclosed, except where such interests are overridden by the interests for fundamental rights and freedoms of the data subject which require protection under Article 1

A person can take part in the EEG research as a test subject only after signing a written consent with processing personal data (the agreement text can be found in [1]), so there is no conflict with the principle in EEG/ERP Portal.

2.1.1. Special Categories

Certain data types are, by default, not allowed to process [2, Article 8, Paragraph 1]:

Member States shall prohibit the processing of personal data revealing racial or ethnic origin, political opinions, religious or philosophical beliefs, trade-union membership, and the processing of data concerning health or sex life.

However, the restriction does not apply in several cases, one of which is again an explicitly given consent [2, Article 8, Paragraph 2]:

Paragraph 1 shall not apply where:

a) The data subject has given his explicit consent to the processing of those data, except where the laws of the Member State provide that the prohibition referred to in paragraph 1 may not be lifted by the data subject's giving his consent

2.1.2. Data Controllers

Persons or entities collecting and processing personal data are referred to as *data controllers*. It is the controller's responsibility to ensure, that the following is complied with - Personal data must be [2, Article 6, Paragraph 1]:

a) Processed fairly and lawfully;

b) Collected for specified, explicit and legitimate purposes and not further processed in a way incompatible with those purposes. Further processing of data for historical, statistical or scientific purposes shall not be considered as incompatible provided that Member States provide appropriate safeguards;

c) Adequate, relevant and not excessive in relation to the purposes for which they are collected and/or further processed;

d) Accurate and, where necessary, kept up to date; every reasonable step must be taken to ensure that data which are inaccurate or incomplete, having regard to the purposes for which they were collected or for which they are further processed, are erased or rectified;

e) Kept in a form which permits identification of data subjects for no longer than is necessary for the purposes for which the data were collected or for which they are further processed. Member States shall lay down appropriate safeguards for personal data stored for longer periods for historical, statistical or scientific use.

2.1.3. Security Of Processing

Secure data handling is not only a practical aspect. It is also a legal requirement, as [2, Article 17, Paragraph 1] states:

Member States shall provide that the controller must implement appropriate technical and organizational measures to protect personal data against accidental or unlawful destruction or accidental loss, alteration, unauthorized disclosure or access, in particular where the processing involves the transmission of data over a network, and against all other unlawful forms of processing.

2.2. Czech Law Definitions

Directive [2, Article 28] requires the member states to form supervisory authorities with the mission to protect individuals with regard to processing of personal information. In the Czech Republic, the role is fulfilled by The Office for Personal Data Protection (Czech: Úřad pro ochranu osobních údajů, ÚOOÚ).

The Czech legal system makes a clear distinction between personal and sensitive data. Stated in the Personal Data Protection Act [3, Article 4]:

> *a) "personal data" shall mean any information relating to an identified or identifiable data subject. A data subject shall be considered identified or identifiable if it is possible to identify the data subject directly or indirectly in particular on the basis of a number, code or one or more factors specific to his/her physical, physiological, psychical, economic, cultural or social identity;*
>
> *b) "sensitive data" shall mean personal data revealing nationality, racial or ethnic origin, political attitudes, trade-union membership, religious and philosophical beliefs, conviction of a criminal act, health status and sexual life of the data subject and genetic data of the data subject; sensitive data shall also mean a biometric data permitting direct identification or authentication of the data subject;*

Analogically to the directive, personal and sensitive data may be processed when given an explicit consent. Detailed in [3, Article 5]:

> *When giving his consent the data subject must be provided with the information about what purpose of processing, what personal data, which controller and what period of time the consent is being given for. The controller must be able to prove the consent of data subject to personal data processing during the whole period of processing*

A practical implication of the text is, that the test subject's written consent should also be archived.

2.3. Personal Data Protection in the U.S.

The term used to describe personal data in the context of storing them is personally identifiable information, abbreviated PII. The Office of Management and Budget defines PII [4] as:

> *any information about an individual maintained by an agency, including*
>
> *(1) any information that can be used to distinguish or trace an individual‘s identity, such as name, social security number, date and place of birth, mother‘s maiden name, or biometric records; and*
>
> *(2) any other information that is linked or linkable to an individual, such as medical, educational, financial, and employment information.*”

Protection of health information is governed by The Health Insurance Portability and Accountability Act of 1996 [5]. Unless specific privacy rule requirements are met, access to protected health information (PHI) requires written authorization. The authorization must be written in plain language, a signed copy must be provided to the individual and it must contain the following core elements [5]:

- *Description of the PHI to be used or disclosed*
- *Identity of individuals or organization who may disclose PHI*
- *Purpose of the use or disclosure*
- *Identity of person or organization to whom PHI may be disclosed*
- *Expiration date or event*
- *Signature (dated) of patient or guardian*

However, when identifying elements are stripped from the PHI, the resulting data can be used or disclosed without such authorization [5][1].

[1]Using this clause, enough data must be removed to ensure, that the chance to identify an individual is very small [5].

3. EEG/ERP Portal

A research group at the Department of Computer Sciences and Engineering is focused on research of attention, particularly in the case of drivers and injured people. EEG and ERP are used as the main research methods. Experiments produce large measurement and related data files. The EEG/ERP Portal was created as a solution allowing long-term storage and management of these experiments [6].

My enrollment in the project has been, so far, exclusively limited to activities described in this thesis - technology upgrade, improving security, fixing known issues and testing.

3.1. Features

Current portal features are, as listed in [7]:

1. *New user registration*
2. *Storage, update and download of experimental data and metadata*
3. *Sharing data a metadata between groups*
4. *Displaying download history for groups admins or system supervisors*
5. *Adding global articles for system supervisors*
6. *Adding groups articles for groups admins*
7. *Article comments*
8. *Fulltext and advanced search in EEG database*

3.2. Users

Users are categorized by their roles, which can be one of the following:

- Anonymous: Any user which is not authenticated. The only accessible pages are homepage with the login form and registration page. Anonymous is the only user category not tied to any database entity.
- Reader: User with minimal rights, created by a third person when adding experiment stakeholders. Reader can log in to the application, access public experiments, scenarios, articles and join groups.
- User: Can do everything what the Reader can, plus create or join groups and apply for a higher role. Creating and storing experiments is not permitted for accounts with this role.
- Experimenter: Has the right to insert and manage his own experiment and scenarios. An User becomes Experimenter after applying for the role in an application form, if the request gets accepted by a Supervisor.
- Group Admin: Any registered user, which created a research group, or was assigned this role by another Group Admin after joining. Can manage group users including changing user roles, write articles and see download history of experiments solved by the members.
- Supervisor (Admin): Global application administrator. Has the right to manage user accounts and has access to all data stored in the application.

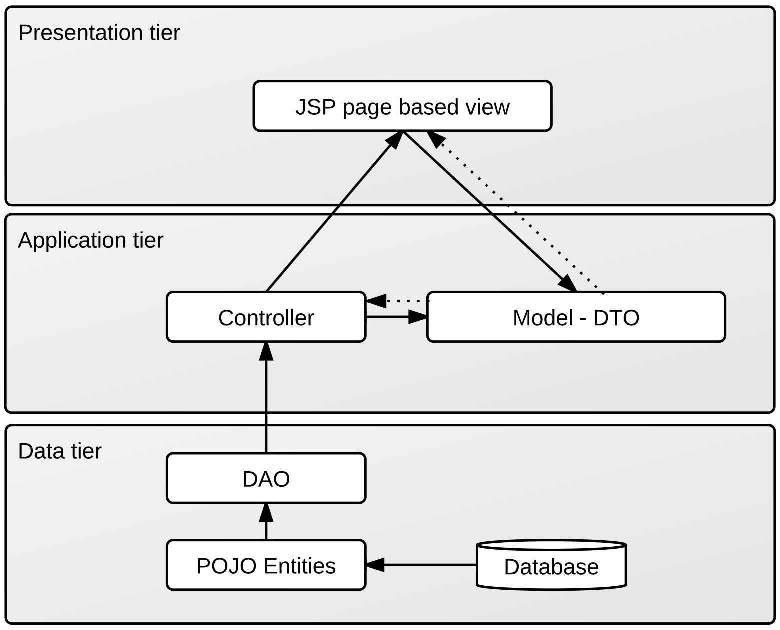
3.3. Architecture

The portal is a three tier web application written in Java. MVC (Model-View-Controller) paradigm is used for separation of concerns in processing server requests. The architecture is supported by common JavaEE (Java Platform, Enterprise Edition) and XML (Extensible Markup Language) frameworks and technologies. Figure 3.1 shows a sum up of the structure.

3.3.1. Data Tier

Storage of all application data is managed by Oracle 11g database. The database is accessed by object-relational mapping framework Hibernate. Individual tables are mapped to POJO (Plain Old Java Object) Java entities, with annotations defining

Figure 3.1.: EEG/ERP Portal Architecture



constraints and mapping properties. Entities are manipulated by DAO (Data Access Object).

3.3.2. Application Tier

Spring Framework is used to achieve loose coupling throughout the application tiers and individual classes. Beside the Spring Core, MVC, Security and AOP (Aspect Oriented Programming) modules are integrated.

In a typical flow, user actions checked by validators are handled by controller classes. A controller calls a DAO[1] to load or store data, and fills a data transfer object (DTO, in Spring called Command Object) to be displayed in the view.

3.3.3. Presentation Tier

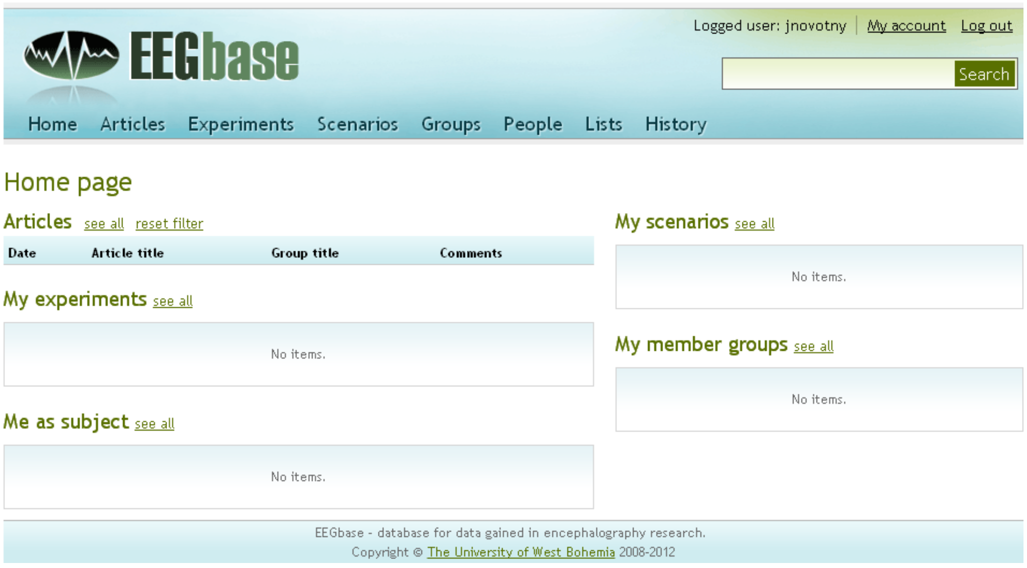
JSP (Java Server Pages) technology is used to render the view to standard HTML web pages. Thus, the application can be accessed from any location by a web browser, without any additional plugins. Embedding code in the view to prepare

[1]In case of more complex actions shared by multiple controllers, a service class is called instead, which might in turn call the respective DAO

data for display is done exclusively via JSTL (JavaServer Pages Standard Tag Library).

Figure 3.2 demonstrates the user interface, showing home page after user login with no listed data.

Figure 3.2.: Application front end



Date	Article title	Group title	Comments

No items.

No items.

No items.

No items.

EEGbase - database for data gained in encephalography research.
Copyright © The University of West Bohemia 2008-2012

4. Authentication and Authorization Solutions

Assuring proper functioning of authentication and authorization is crucial to application security. This chapter will provide a basic overview of the principles and technical solutions in this field, to be used later in the realization phase.

In each section, the concept will first be introduced. Then, several major aspects are covered, explaining solutions available in the framework used by EEG/ERP Portal, Spring Security.

4.1. Authentication

The goal of authentication is to identify and verify valid users [8]. Submitted identification is checked and paired with a record stored in the system. This can be then used to selectively display data relevant for each individual user or to access protected areas, if *authorized* to do so.

Typically, part of any system will be accessible by unauthenticated users, further on referred to as anonymous. Such public areas do not require to log in, do not display any sensitive information and do not change the overall state of the system or its data [9]. Every page displayed before logging into the system also belongs to this type.

Authentication methods in general, including form-based login[1] used in the EEG/ERP Portal was already covered in [1]. This section will therefore focus more on specific solutions and features accompanying the login.

[1]Collecting user credentials using data posted in a HTML form.

4.1.1. Remember Me

Users can avoid the need to present username and password for every login by using a *remember me* feature. A cookie created by encrypting user data is stored in the browser. Next time, instead of needing to present username and password, the cookie is read and the user will be automatically logged into the application [9].

Users authenticated with the remember me feature can be distinguished from those who provided full credentials by access rules as shown in Listing 4.1. This is suitable for situations, where the users should verify their identity in order to make important changes like account manipulation. In the EEG/ERP Portal, fully authenticated access is enforced in account creation and modification, including Facebook Connect, accessing web services and deleting lists.

Listing 4.1 Forcing user logged in with *remember-me* to reauthenticate

```
<intercept-url pattern="/account/*.jsp"
        access="IS_AUTHENTICATED_FULLY"/>
```

4.1.2. Password Storage

In case full user credentials are stored in the application, design of such system should ensure security of the login data, most importantly the password[2]. Even if the user database is compromised, passwords can be stored in a way that would require an impractical time to decode.

Chapter 4 in [9] lists general rules for storing passwords in a database:

- *Passwords must not be stored in cleartext (plain text)*
- *Passwords supplied by the user must be compared to recorded passwords in the database*
- *A user's password should not be supplied to the user upon demand (even if the user forgets it)*

Such approach is fully supported by Spring Security. Encrypting a password during the authentication process is defined by different *password encoders* and can

[2]Revealing a password can lead not only to gaining access to our application, but also to other (possibly critical) systems, should the user reuse the same password [10]

be set up directly in the configuration. Currently available implementations, listed in Table 4.1, use one-way hash functions to produce encoded strings. The framework also supports hash salting, which is a method protecting against attacks using precomputed hash values to reveal the original password.

Table 4.1.: PasswordEncoder implementations in Spring Security 3.1

Name	Hashing function
PlaintextPasswordEncoder	none (plain text)
Md4PasswordEncoder	MD4
Md5PasswordEncoder	MD5
ShaPasswordEncoder	SHA, SHA-256
LDapShaPasswordEncoder	SHA
BCryptPasswordEncoder	BCrypt

4.1.3. Third Party Login

User authentication can be delegated to a trusted third party. A so called *single sign-on* solution verifies the user to be used in multiple services. When logging in using a single sign-on, the application does not need to check user credentials - after one is identified and matched with a local record, authentication is complete.

Build-in providers supported by the security framework are listed in Table 4.2. This can be further extended by Spring Security Extensions, adding support for SAML (Security Assertion Markup Language) and Kerberos, or by specialized modules like Spring Social.

Table 4.2.: External authentication providers in Spring Security 3.1

Name	Authentication system
CASAuthenticationProvider	Central Authentication Service
LdapAuthenticationProvider	Lightweight Directory Access Protocol
JaasAuthenticationProvider	Java Authen. and Author. Service
OpenIdAuthenticationProvider	OpenID

4.2. Authorization

Authorization means allowing different levels of access to data and system resources [8]. While some authenticated users might have minimal rights to see only basic information, others with administrative rights can be set up to access the whole system configuration and dataset. This is achieved by mapping users to different *roles/authorities*. A group of similar users are typically assigned the same role. Roles or individual permissions are then used to check access to *secured resources* like internal web pages.

Following paragraphs will cover access control in the presentation tier. Business code can also be secured, a topic detailed in [9].

4.2.1. Page-level Security

With Spring Security, authorization rules based on URL patterns or individual web pages can be defined in the global XML configuration. Access rules either directly list roles needed to access a resource, or use special values for checking authorization levels like in Listing 4.1. A new feature in the current framework version is the use of an expression language[3], which can substitute the constants and provide a more flexible configuration (listed in comparison with the constant based configuration in Table 4.3[4]). Multiple rules of a single type can be combined.

4.2.2. Individualized Views

Access control is not limited to whole pages and application resources. Authorization rules can and should also be used to selectively display parts of individual views. One way to do it is to check the user authorities in business tier and conditionally render page content. But because such checks are common in a typical application and would result in a lot of repeated code, frameworks like Spring Security provide support to define access rules on page level among presentation code.

There are three possible approaches for conditionally rendering page content using *authorize* tag in JSP or similar pages. A code example is in Listing 4.2.

1. Using expressions: Rules listed in Table 4.3 can also be used to decide, whether to output code enclosed by the tag
2. Listing roles: Same as the previous, only directly listing roles needed without using expression language.

[3]Using expression language has to be explicitly enabled in the security configuration file

[4]Methods with no parameters can be accessed as *pseudo-properties*, omitting the parentheses and prefix [9]. For example *isAnonymous()* can be written as *anonymous*.

Table 4.3.: URL Access configuration

Expression Language Method	Configuration Without Expressions	Description
hasRole(role)	ROLE_NAME	User must have a granted authority matching ROLE_NAME
hasAnyRole(roles)	ROLE_1, ...ROLE_N	Any of the listed roles will be accepted
hasIpAddress(address)	-	Only a certain IP address is accepted
permitAll	IS_AUTHENTICATED _ANONYMOUSLY	Access is always granted, even to anonymous users
denyAll	specifying nonexistent role or with other mechanisms	Access is always denied
isRememberMe()	IS_AUTHENTICATED _REMEMBERED	Checks if user is authenticated using the remember-me function
isAnonymous()	-	Checks if user is anonymous
isAuthenticated()	-	Checks if user is authenticated
isFullyAuthenticated()	IS_AUTHENTICATED _FULLY	Checks, if user provided login credentials (is not anonymous or remembered)

3. Checking URL access: Part of the page will get rendered if the user has access to a certain URL. This provides a way to reuse the global configuration, for example by showing only links to pages, which one is authorized to visit.

Listing 4.2 Conditionally displaying messages by checking roles and URL access

```
<security:authorize ifAnyGranted="ROLE_USER">
  Hello user
</security:authorize>
<security:authorize url="/invoices.jsp">
  <a href="/invoices.jsp">Would you like to see company invoices?</a>
</security:authorize>
```

5. Frameworks Used in EEG/ERP Portal

The portal is built on top of several architecturally significant frameworks. This chapter will provide a basic introduction[1] to the three most important, then focusing on features introduced in the newest versions and backward compatibility. Features used by EEG/ERP Portal will be highlighted.

5.1. Spring

Spring is an open source framework aiming to simplify enterprise application development. That is achieved mainly by using loosely coupled POJO based components, instead of Enterprise Java Beans (EJB). Setting dependencies is done via a mechanism called Dependency Injection[2]. Instead of initializing dependencies in the code of each class, wiring is done by an Inversion of Control container. Individual managed classes (Spring beans) do not need to be aware of the application context and avoid boilerplate initialization code.

Spring incorporates several modules covering a range of services. The following are used in EEG/ERP Portal:

- Core - Container, that defines, how the beans in a Spring-enabled application are created, configured, and managed [11]. This part is used by all other Spring modules.
- AOP module - Support for aspect-oriented programming. By using aspects, cross-cutting concerns like logging and security can be decoupled from objects

[1]Apart from dedicated books, description of the frameworks can be also found in [1]

[2]A software pattern for setting dependencies during runtime or by configuration, instead of compile-time. Such an approach increases flexibility and enables implementation swapping for the purposes of testing, aspect-oriented programming and others.

they affect.

- Data Access - Includes support for both Java Database Connectivity (JDBC) and Object-relational mapping (ORM). Manages transactions and creates an abstraction layer on top of data access exceptions.
- Web and remoting - Servlet-based MVC framework and Java API for XML Web Services (JAX-WS) support are extensively used throughout the portal.
- Security - Described in section 5.2. Not bundled with Spring distribution by default.
- Testing - Provides support for unit tests and integration tests, which can make use of initialized application context.

5.1.1. New Features in Spring 3.0

Important features introduced in the current version are [11][12]:

- REST (Representational state transfer) support and a new namespace for simplifying configuration in Spring MVC
- New expression language
- Annotations in Spring MVC for retrieving cookies and request headers
- Declarative model validation with annotations introduced by JSR-303 (Java Specification Request 303)
- JSR-330 dependency injection specification
- Support for embedded databases
- Java based metadata allowing configuration to be stored in bean code instead of XML
- Object to XML mapping functionality was moved to Spring core

5.1.2. Backward Compatibility

Java 1.4 is no longer supported by the current Spring version. But with that exception and a trivial need to change XML namespaces, Spring 3.0 is fully backward compatible.

5.2. Spring Security

Started in 2003 and originally called *The Acegi Security System for Spring*, the project gradually became adopted as standard security solution for Spring based applications [9]. The framework can handle authentication and authorization on the web request level using servlet filters, but can also provide access control in individual methods by leveraging Spring AOP support.

Configuration is stored in XML format using a security specific namespace. Spring Security has build-in support for mapping database users to security principals, form login and logout, all of which is used for authentication in the EEG/ERP Portal. Properties like user name and authorities are accessible directly in the view.

5.2.1. New Features

Important changes were introduced in version 3.0 over the second edition [9]:

- Spring Expression Language can now be used in access declarations
- More configuration options and hooks in the authentication process
- Pre and post invocation method access declarations in annotations
- Session access management and concurrency control (handling same user logged in multiple sessions)
- Revised ACL (Access control list) module
- Added support for Kerberos and SAML single sign-on with the Extensions project, improved OpenID support

Furthermore, several features were added in Spring Security 3.1 [13]:

- Support for multiple http elements, stateless authentication, HttpOnly cookies, hasPermission expression, disabling UI security in testing and nested user switching
- Credentials can be erased after successful authentication and cookies cleared on logout
- Improved LDAP, CAS and JAAS support
- Added Basic Crypto Module

5.2.2. Backward Compatibility

Spring Security 3.1 mandates migration to Spring Core 3.0, in turn breaking compatibility with Java 1.4. Build-in support for NTLM (NT LAN Manager) authentication was removed. XML configuration was partially restructured and improved with the new namespace, but it is possible to find a setup directly matching one of Spring Security 2.

Package structure and class names were refactored in 3.0 (renaming of individual packages is detailed in [9][3]). EEG/ERP Portal utilizes directly only several Spring Security API classes, as shown in Table 5.1, so refactoring should not be an obstacle.

Table 5.1.: Imported Spring Security classes

Spring Security class / interface	Used in
AbstractAuthenticationToken	Facebook authentication
Authentication	Facebook registration
AuthenticationException	Login page
GrantedAuthority	User details retrieval, Facebook authentication
GrantedAuthorityImpl	Facebook registration
PasswordEncoder	Password manipulation
SecurityContextHolder	User details retrieval, Facebook registration
UserDetails	User details retrieval

To sum up, while Spring Security 3.1 is not fully backward compatible with the previous versions, assuring compliance after migration in EEG/ERP Portal is not expected to become a major issue.

5.3. Hibernate

Hibernate is an open source ORM framework. The project aims to be a complete solution to the problem of managing persistent data in Java [14]. By mapping database records to object-based entities, the framework creates an abstraction layer is on top of a database, allowing developers to focus on the business logic, instead of tuning data exchange with the database management system.

[3]The corresponding chapter is also available online at http://www.packtpub.com/article/migration-to-spring-security-3

Mapping between objects and tables is accomplished by XML or annotations based configuration and might include *cascade* operations ensuring data integrity on update actions. Custom queries are written using Hibernate Query Language (HQL), an SQL-like language for manipulation with entity objects.

Major features introduced between version 3.3.1 and the current release 3.6.7 include [15]:

- Support for JPA 2 (Java Persistence API 2, defined by JSR 317)
- Hibernate Annotations, Entity Manager, Envers and Java Management Extensions (JMX) became part of the core project[4]
- Infinispan was added as standard second-level cache
- Support for JDBC 4
- Improved Annotations and Type support

As for backward compatibility, relevant changes are dropping support for Java 1.4, Document Type Definition (DTD) namespace change and mapping properties configured with type="text" to JDBC LONGVARCHAR instead of CLOB type. Also, when using *materialized_clob* or *materialized_blob* properties, an environment variable *hibernate.jdbc.use_streams_for_binary* must be set to true on Oracle and false on PostgreSQL.

[4]The change simplifies version configuration, as finding a suitable combination of the modules no longer requires use of a compatibility matrix.

Part II.

Design and Implementation

6. Migration Plan

Main goal of project realization is to improve security of the EEG/ERP Portal, while using new versions of already integrated frameworks. Because an update using currently used structures faces a number of issues, upgrade to the new framework will be preceded by introducing a tool for improving project management. Reasons for such a step and relevant technologies are explained in the following sections, concluded by a list of actions planned during the realization.

6.1. Blocks to a Direct Update

As was explained in chapter 5, backward compatibility should not slow down the framework update considerably. There are however other issues, which are likely to affect complexity of the process.

6.1.1. Library Structure

Project libraries are stored in a directory structure of JAR (Java Archive) files. Apache Ant puts these on project classpath when building the project[1]. This situation became problematic as the project size increased, for two main reasons:

1. There is no central record of individual library versions. Assuring compatibility, when adding a new library to the project is difficult. Even if the library's dependencies are fully known, conflict with transitive dependencies cannot be easily detected.
2. Some libraries are stored more than once in the directory structure[2]. Therefore, it is not possible to determine, which version will be used by the Class

[1]When assembling the application, the libraries are copied to lib directory in WAR archive

[2]A typical situation causing this misconfiguration is adding a new framework with all dependencies as a directory, not checking if some of the JARs are already included in the project.

Loader during runtime. If there is an interoperability problem between a library and a specific range of other's library versions, the issue is very difficult to debug (one of the conditions informally described as Jar Hell[3]).

To avoid creating an unstable configuration, libraries will either need to be manually reviewed and compatible versions found to match with the new framework, or a new build system tracking dependency versions be deployed.

6.1.2. Absence of Tests

Although tests were previously added to the project [1][16], they are not maintained and none of Unit tests are currently working. Code has to be tested manually from the user viewpoint. That makes it difficult to test small incremental changes and scan the code base for regressions, when library setup is updated.

6.2. Introducing Maven

As explained in section 6.1, managing dependencies is a major issue complicating future expansion of the project. It is theoretically possible to manually sort out the conflicts by checking each library's compatibility and storing dependency versions matching the new frameworks, but the setup would need to be revised every time a new library is added.

A better solution would be to manage the dependencies automatically. That is possible with a variety of tools, two of the most widely used are:

1. Apache Ivy: A subproject of Apache Ant, this tool introduces dependency management, while integrating with Ant-based builds
2. Apache Maven: Build automation and project management tool, originally created to solve maintenance issues inherent to Ant-based projects [17].

A short comparison of properties relevant to EEG/ERP project framework migration is listed in Table 6.1. Ivy would enable dependency management similar to Maven, and because it leverages existing Ant build, it would be fairly easy to integrate with

[3]Derived from the so called DLL Hell, where incompatibilities between dynamically linked libraries plagued older versions of Microsoft Windows.

Table 6.1.: Comparison of build tools

	Ant + lib dir	Ant+Ivy	Maven
Standardized build process			✓
Dependency management		✓	✓
Build completely IDE-independent			✓
Simple build customization	✓	✓	
Migration effort	-	small	large

the project. Also, Ant build files are easy to customize, as they define each step of the build, following the tradition of Unix makefiles.

Maven advantages are, however, important enough for it to be chosen as the new build system:

1. Standardized build means every project and Maven build is similar. Ant and other build systems let the developer define each build step, leading to verbose and functionally different buildfiles even for projects with identical build targets (code compilation, unit testing, packaging as web application...). In contrast, Maven build file defines only project properties like name and dependencies[4]. Assuming the project has a predefined structure, the build system can automatically assemble all needed targets. Once a developer learns Maven, it is easy to understand the structure of any other Maven project, manage and build it, as it uses a common set of commands. As a result, all project members can understand and extend the configuration and build, as opposed to only the author in the case of a complex Ant/Make script, an important feature to an academic project, whose members might potentially change every semester.

2. Dependency on a particular IDE (Integrated development environment) has long been seen as a problem for the project. Build files are now tailored to one environment and IDE specific configuration files are needed to start working on the project. Developers joining the project can't use their preferred IDE, needing to wait for their license keys and then adopt the supported environment. Also, IDE vendor cannot be switched if licensing conditions cease to be acceptable. In contrast, Maven based projects can be loaded by all major IDEs and because the configuration is fully defined in the single build file, no specific project files and directories are needed to load and use the

[4]In case there is no way around executing a custom code during a build, it is possible to write plugins and hook them to the project lifecycle.

codebase.

For a longer time, migration to the new build system was considered by the project members. Framework upgrade provided a good initiative and timing to make the changes right away. The build system switch will be described in chapter 7.

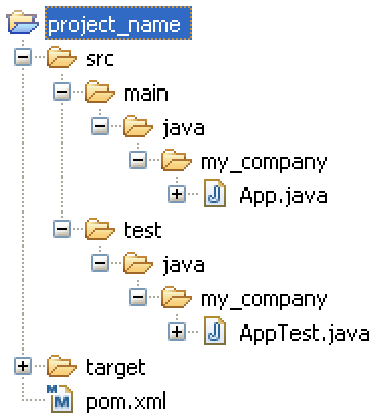
6.2.1. Tool Description

Using a formal definition from [18]:

> *Maven is a project management tool which encompasses a project object model, a set of standards, a project lifecycle, a dependency management system, and logic for executing plugin goals at defined phases in a lifecycle.*

An important concept used by Maven is *convention over configuration*: Describing the build process and configuration is not needed, if the project uses predefined setup. The tool will assume reasonable defaults, like a standard build compiling the sources in *src/main/java* (Figure 6.1), combining them with resources and placing the assembled artifact in the *target* directory[5].

Figure 6.1.: A minimalistic application following the standard directory structure



```
project_name
  src
    main
      java
        my_company
          App.java
    test
      java
        my_company
          AppTest.java
  target
  pom.xml
```

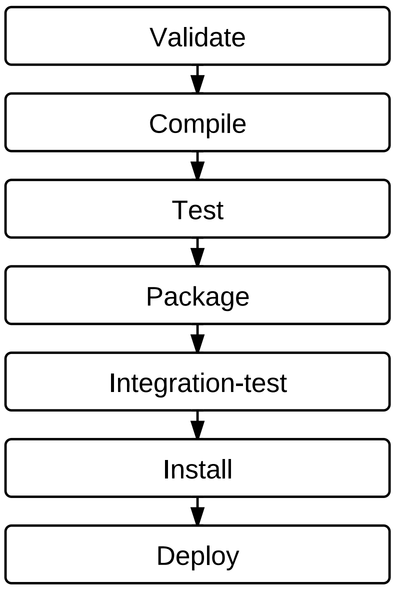
Metadata, dependencies, settings and other project properties are described in the Project Object Model (POM), which is represented by a XML file *pom.xml* in the project base directory.

[5]The complete standard directory layout can be viewed at http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html

Project compilation, assembly and distribution process is defined by the *build lifecycle*, essentially a set of sequentially executed phases. When an action is to be executed during a certain phase, all preceding phases are passed through first.

Main phases[6] of the default lifecycle are listed in Figure 6.2.

Figure 6.2.: Project Lifecycle



All the functionality beyond a few core tasks is covered by a system of plugins. These execute at defined phases in the lifecycle and fulfill goals ranging from running unit tests, to generating a website or starting a web server. Plugins can be updated individually to make use of the latest features.

6.2.2. Artifact Repositories

In Maven, build artifacts like assembled JARs are stored in repositories. In principle, repository is a locally or remotely accessible directory tree with artifacts stored on the following path:

/<groupId>/<artifactId>/<version>/<artifactId>-<version>.<packaging>

[6]The list is not exhaustive, for a complete table of default lifecycle phases including description see [18]

When a project is assembled, Maven will use repositories to fetch all needed dependencies. There are two repository types:

1. Local repository: A cache of artifacts downloaded from remote repositories, located in the filesystem of each machine[7]. Once cached in the local repository, Maven does not need to refetch the artifacts from remote repositories.
2. Remote repository: All others. A special case is the Central repository at http://repo1.maven.org/maven2/ , enabled by default and storing a large amount of public open-source project artifacts.

Locally developed and non-public libraries can be stored in a separate internal repository[8]. Publishing and managing such artifacts, together with centralized caching of the public libraries can be simplified with applications called repository managers. Such a setup increases control and stability by not relying on external resources [18].

6.3. Migration Steps

To upgrade the technologies and improve security, the following steps will be made:

- Switching to Maven and importing up-to-date frameworks
- Analyzing shortcomings from the security viewpoint
- Improving authentication and authorization based on the revealed flaws
- Fixing and extending automated testing
- Evaluating results with independent tools

Following chapters will describe the process, including reasons for choosing a particular solution, when more options are available.

[7]The local repository is stored alongside user settings in *.m2* directory

[8]From the build tool viewpoint, the repository is still remote, even when located on a local network or filesystem.

7. Technology Update

Because using any of the features introduced in new framework versions must be preceded by updating the technologies, migration was planned as the first stage of realization. Aside from introducing the new frameworks, it should do the following:

1. Change build system
2. Unify and upgrade project dependencies
3. Assure compatibility with multiple IDEs and tools

As the changes break existing build procedures, a new code branch was created in the used versioning system (Subversion, SVN) and merged with the trunk only after the project migration was complete, checked and tested[1].

7.1. Project Directory Structure Changes

One of the main differences between Maven and Ant is purpose of its configuration. Ant file defines each build target and steps in make-like fashion and is always tailored to project specifics. In Maven, the POM describes only project properties like name and dependencies, Maven uses build-in conventions to execute project goals like compiling or packaging source code.

One disadvantage is, that by specifying a recommended default behavior, migrating to Maven requires non-trivial changes to an existing project just to fit for the new build system. But, by doing so, the project structure and build process becomes standardized and easily used by any developer with previous Maven experience.

If needed, the default project setup can still be overridden. But because that would mean losing many of the core advantages of Maven approach, I chose not to do so.

[1]Testing is described in chapter 11

Maven defines a standard directory structure, clearly separating sources, tests and resources. Project directories were changed to adhere to that structure, as shown in Table 7.1. A complex Ant build file structure [2] could be then replaced by a single *pom.xml* file.

Table 7.1.: Original directory structure vs. Maven recommended layout

Logical unit	Original path	Maven path
Source codes	src/java	src/main/java
Resources	src/java	src/main/resources
Published web files	web	src/main/webapp
Test source codes	src/java/cz/zcu/kiv/eegdatabase/test/	src/test/java
Test resources	src/java	src/test/resources
Compiled classes	build	target
Build files	build.xml, build-before-profiler.xml, trunk.xml, nbproject/ant-deploy.xml, nbproject/build-impl.xml, nbproject/profiler-build-impl.xml, nbproject/project.xml, .idea/ant.xml	pom.xml

7.2. Unifying Dependencies

For each library a dependency definition was added to POM (as in Listing 7.1). When two or more versions existed for the same library (for example

- lib\hibernate\hibernate3.jar
- lib\hibernate-search-3.1.1.GA\dist\lib\hibernate-core-3.3.1.GA.jar

both contain the hibernate core, first in version 3.2.5, second 3.3.1), the more up-to-date version was used. After the project was migrated and tested with library versions matching the old build configuration, frameworks described in chapter 5 were switched to the most recent, updating the source codes where needed. In total 163 jar files were replaced by 80 dependency definitions[3].

[2]Main build file *build.xml* used several auxiliary scripts, including Netbeans-generated builds in */nbproject* directory for core tasks. When building with IDEA, the main build file was called by */.idea/ant.xml*.

[3]Maven tracks dependencies of every project/artifact. As some libraries are not directly referenced by EEG/ERP Portal, they do not need to be explicitly defined in the POM, but are loaded as transitive dependencies of the artifacts using them.

Finally, JAR files were then removed from SVN, as Maven downloads both the classes and sources automatically.

Listing 7.1 Dependency definition in *pom.xml*

```
<dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>3.6.7.Final</version>
</dependency>
```

7.3. Storing Custom Libraries

Most of the application libraries are available in one of public Maven repositories:

1. Maven central repository
2. java.net Repository for Maven
3. JBoss Public Maven Repository Group

However, libraries developed at the university are not publicly available and were not uploaded to any of the repositories. A similar situation is for libraries having a restrictive Oracle Binary License, prohibiting unlimited public distribution. Both of these types (listed in Table 7.2) thus needed to be stored in a separate repository. The last added type were libraries developed in smaller-scale project located at custom public repositories, but not the central. Because repository outage could block project checkouts, these libraries were also to be stored in the new internal repository.

A standard solution to store the custom libraries would be to use a repository manager. However, none was available at the time of project realization, neither was an option to deploy one in the university infrastructure available. A directory-based repository was therefore created instead and placed on SVN to assure consistency between all project instances. Because the Maven repository structure is followed, it can be added as any other remote repository to the build descriptor, substituting URL for relative directory path (Listing 7.2). All application libraries are thus tracked, stored and managed by Maven, independent of their location.

The solution has, however, important disadvantages:

Table 7.2.: Locally stored libraries

Group ID	Artifact ID	Reason
com.oracle	ojdbc6	Restrictive license
com.oracle	orai18n	Restrictive license
com.oracle	xdb	Restrictive license
com.oracle	xmlparserv	Restrictive license
com.jhlabs	imaging	Not available in major public repositories
owlapi	owlapi-bin	Not available in major public repositories
cz.zcu.kiv	jenabeanextension	In-house developed library
cz.zcu.kiv	eegdsp	In-house developed library

Listing 7.2 Specifying directory-based repository in pom.xml

```
<repository>
        <id>cz.zcu.kiv.eegdatabase.repo.local</id>
        <url>file://${basedir}/repo</url>
</repository>
```

1. Availability of public libraries is dependent on the uptime and free bandwidth of public repository servers. Should the central repository go down, it would be needed to distribute project libraries manually by copying local cached repositories from one machine to another[4].
2. Storing custom libraries in versioning systems is not recommended[5]. Manually storing a library in the SVN-based repository is counter-intuitive and prone to errors, because a specific structure and naming need to be respected[6]. Libraries stored in the versioning system also slow down project checkout.

For this reasons, it can be recommended to migrate to a local repository manager, as soon as one is deployed in the university environment.

[4]Using a Repository Manager would solve this issue as it caches the public libraries, needing to download the data only once and then distributing them to developer machines on request as any other repository

[5]Statement can be found at http://maven.apache.org/guides/introduction/introduction-to-repositories.html

[6]Uploading a new library version to SVN, but not updating XML annotations and failing to put it in an appropriate directory can result in Maven not recognizing the update, using the old version in all developer's machines.

7.4. Porting JSON Plugin

Part of the build process is a tool for converting messages saved in a Java property file to JSON (JavaScript Object Notation) format. The produced file becomes source of messages for AJAX (Asynchronous JavaScript and XML)-enabled pages, in particular room booking introduced by [16].

To generate JSON messages during project build, the converter code was packaged as Ant Task and called in *build.xml*. Ant build is no longer used, so the maven build needs to be customized to call the converter. As is stated in [18]:

> *While Maven 1 emphasized Jelly scripting for customizing builds, Maven 2 favors custom plugins or customization through scripting Plugins or the Maven Antrun Plugin.*

Two of the options are suitable for JSON converter:

1. To use Maven Antrun Plugin executing Ant build target, which in turn will call the existing Task. A downside to this approach is complicating the build[7] and adding a potential source of compatibility problems.
2. To write a custom Maven plug-in using existing converter code and use only standard methods to integrate the plug-in to the project life-cycle.

Because the converter algorithm was trivial and source code available, I chose the latter solution. JSonConverter was restructured to follow Maven plug-in conventions[8], set to execute in the *process-sources* phase and saved to the internal Maven repository.

7.5. Building in Major IDEs

Maven is currently supported on all major IDEs. Because the POM stores project configuration, an IDE can load Maven projects directly from the versioning system with minimal additional configuration, as shown in Table 7.3. The project has been tested on the three listed development environments.

[7]The build would become Ant/Maven combination, maven directory paths would need to be passed to Ant Task

[8]While keeping Ant build file to package it as Ant Task, if needed

Table 7.3.: Extra configuration files needed to build in IDE

IDE	Original ant-based build	Maven build
IntelliJ IDEA	*.idea* directory	-
NetBeans	*nbproject* directory	-
Eclipse + M2E plug-in	not supported	life-cycle mapping in *pom.xml*

7.6. Command Line Support

Project can be built without an IDE by using command line options of a local maven installation. This is illustrated in Table 7.4.

Table 7.4.: Examples of command-line application build and execution

command	effect
mvn package	Creates a WAR file to be deployed on application server
mvn jetty:run-war	Builds and runs project locally on an embedded Jetty server

8. Analyzing Security Loopholes

After the upgrade was complete, several security issues were identified. Most were already known in the time of writing this thesis, but were not fixed because of technical difficulties or lack of time to implement an appropriate solution.

8.1. Password Storage

User credentials are protected by hashing the password before saving to the database by MD5 message digest algorithm. In the event of a partial security compromise, when a malicious user is able to retrieve database user records, plain text passwords are not immediately visible. Although this might repel a naive attacker, it is not considered secure. A malicious user can then use a *Rainbow table*[1] (a type of pre-computed lookup table for reversing cryptographic functions) to retrieve plain text passwords. An effective method to protect against this kind of attacks is adding a pseudo-random character sequence called Salt [9][19].

Another issue is, that plain text passwords are sent by email after registration. If a hacker gets access to an user's email, he would be able to instantly log in with his credentials. But possibly even worse, he might just have revealed a password used for other services. Many users use the same passwords for multiple systems and retrieving one password can lead to a large security breach [10].

8.2. Error Stack Traces

When the application enters an unexpected state or encounters a programming error, it prints the error stack trace directly to the HTML output (default behavior). This

[1]These are publicly available on sites like http://www.freerainbowtables.com and can be used to retrieve passwords up to eight characters in length. When the character set is limited, the maximum length can be further increased.

makes it a case of information disclosure, revealing implementation details, file paths etc. to a potential attacker [8].

Unhalted exceptions can be wrapped by a generic Server Error page. Debug information would still be accessible to authorized users (developers), as the application already has an extensive log support recording such events.

8.3. Unverified Address Inputs

This issue was already addressed in [1], although subsequent code changes introduced other vulnerable URLs.

8.4. Transport Layer Security

Transport Layer Security (TLS)[2] can be used to cryptographically secure data transmitted between the browser client and web server [9]. This type of protection is not set up on the production server, meaning communication with the client is not encrypted.

When an attacker is located on the same local network as the victim, network activity can be monitored and logged by packet sniffers. Besides potentially revealing sensitive information, this becomes even worse during the login procedure, as both username and password are sent in plain text. An attacker can collect these values and use them to compromise the application [8].

8.5. Authorization Issues

Personal information of other users is accessible to any group administrator. As users are allowed to create groups, any user can become a group administrator and see other user's personal details.

[2]TLS is a successor to Secure Sockets Layer (SSL) protocol. As the term SSL became widely known, it is often used interchangeably with TLS, as the main principle stays the same.

8.6. Cookies Accessible to Scripts

Marking cookies as HttpOnly[3] is not enforced by the application, making it easy to read *remember-me* and *session ID* cookies by scripting languages. That may be used to compromise the session in XSS attacks [20].

Snippet in Listing 8.1 shows *session ID* cookie value when injected to the portal homepage and clicked. If cookies were marked as HttpOnly, the dialog would be empty[4].

Listing 8.1 Reading cookie with JavaScript

```
<input type="button" value="Show cookie"
        onclick="alert(document.cookie);">
```

Still, there are methods to bypass the limitation [21], so the setting should not be relied upon to eliminate XSS.

[3]As the name suggests, a HttpOnly cookie can only be accessed by a HTTP request

[4]Although this code is harmless, it is not hard to create one sending request including the cookie value to the attacker's site, as demonstrated in [20] using forged image source address.

9. Restructuring Authentication Mechanism

User authentication was restructured for two main reasons:

1. To fix security problems described in chapter 8.
2. To improve user experience in login, namely to use email as main identification and to use homepage as a primary login form.

This chapter describes steps needed for the improvements and fixes, as well as reasons for choosing a particular solution.

9.1. Unifying Account Manipulation

Before making any changes to authentication mechanism, code loading and changing the user credentials was identified (Table 9.1). Four classes contained a full account creation code.[1]. To simplify migration and testing, a new PersonService interface and implementation class were added to unify the account creation algorithms. Other classes were not changed.

9.2. Securing Database Password Storage

Following the evaluation in section 8.1, previous recommendation in [1] and consultation with the project members, it was agreed to change the password encoding scheme, despite the update not being transparent to project users[2]. A new mechanism will need to have:

[1] This is against the DRY (Don't repeat yourself) principle, and would increase chance to overlook or introduce errors during migration and later maintenance.

[2] Users need to reset their passwords by clicking a "Forgotten password" link

Table 9.1.: Manipulation of user credentials

Class name / configuration file	Account creation	Changing password	Password verification
AddPersonController	✓		
ChangePasswordController		✓	
ChangePasswordValidator			✓
FacebookController	✓		
ForgottenPasswordController		✓	
RegistrationController	✓		
WizardAjaxMultiController	✓		
security.xml			✓

1. A secure one-way hash function. MD5 is no longer considered secure, as practical collision attacks were demonstrated [22].
2. A suitable salt source. Any string unique for every user[3] can be used as salt source, or it may be randomly generated.

In Spring Security SHA-256, a widely used, secure universal hashing function can be used to store passwords. Due to a number of advantages, another algorithm called BCrypt was added[4] to Spring Security 3.1. Build-in mechanisms for using these two functions differ and both were considered, as shown in Table 9.2.

After this consideration, BCrypt based solution was chosen as it keeps password data in a single database column, is simple to use and has a high security margin. Importance of the first reason lies in the fact that a seemingly unrelated change (for example changing user email) would affect user authentication - block application login. Simple usage should grant easier code maintenance in comparison with the more complex SHA-256 based setup.

Directly revealing salt source to a potential attacker is mitigated by the fact that computing Bcrypt hash with the used computational cost 10 is, by design, slower by several factors [5] than SHA-256, rendering dictionary attacks impractical [19].

[3]A simplistic approach using a single salt would create an undesirable weakness, as only a single lookup table would need to be constructed for all such passwords. Also storing identical passwords will produce the same hash [9].

[4]Although addition to Spring Security is new, the algorithm itself is subject to peer review since 1999

[5]When measured on a personal notebook PC, computing a BCrypt hash of an eight character password takes 0.15 sec, as opposed to 0.000004 sec. in the case for SHA-256

Table 9.2.: Choosing password storage mechanism

	SHA-256 + custom salt	BCrypt + generated salt
Hash function	Secure Hash Algorithm by The National Institute of Standards and Technology, successor to SHA-1 and SHA, inspired by MD5 [22], in this case with digest length 256 bits	Algorithm with adaptable computational cost created specifically for storing passwords [19]
Spring Security support	Only hash is stored in database password column, another user property can be used as salt source	Random salt is generated on registration and stored along with the password
Suitable salt source	Registration date or email	(stored with password)
Usage in application	Encoder and salt source can be autowired, used together to verify passwords	BCryptEncoder can be instantiated and a single method called to verify passwords
Implementation Disadvantages	Changing value of property used as salt source disables user login, verbose setup	Plain text salt value stored with password, needs increasing database column length

9.3. Email as Primary Identification

Having custom usernames as primary user identification is practical from the developer point of view. First, when any user posts an article or experiment, his username can be shown freely to others browsing the website. Second, an advanced user or a tester can simply create multiple accounts using the same email for verification.

The problem is, users tend to forget the username[6]. This forces them to often use a password retrieval link instead of just logging in.

Using email address is thus more convenient method to log in. Registration is also simplified, as one needs to fill one text field less and doesn't have to worry about the username being already taken. Users are already familiar with this kind of identification, as it is used on major websites like Google or Facebook.

Switch to login by email was planned to be rolled out together with the updated password storage strategy (section 9.2). The following steps were made:

[6]Unlike email, which is rarely changed

1. Email was set up to become the principal username loaded by Spring Security on login and be verified and saved as username to the database on registration
2. Account manipulation forms were updated to use email exclusively
3. Usernames visible for all registered users (as in case article/experiment authors) were replaced by name and surname[7]

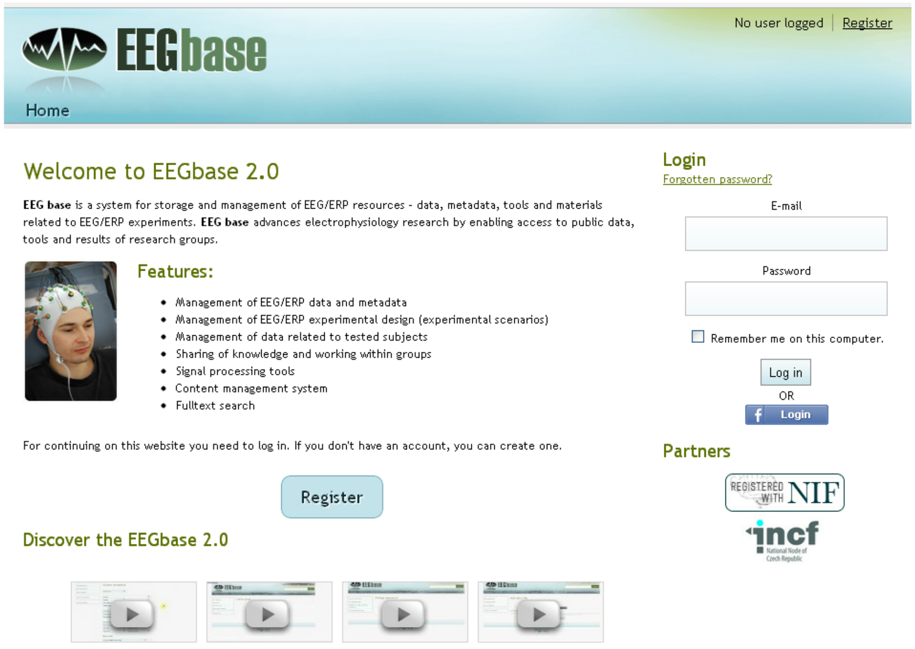
9.4. Single Login Page

Originally, two pages were used for login:

1. home.html, application homepage including a login form (Figure 9.1)
2. login.html, page presented when accessing a protected resource

The latter didn't include the option to authenticate with Facebook or to be remembered. Two login pages could also be confusing for the users. Therefore, home.html was set up as the only page for user login.

Figure 9.1.: EEG/ERP Portal homepage with login form



Welcome to EEGbase 2.0

EEG base is a system for storage and management of EEG/ERP resources - data, metadata, tools and materials related to EEG/ERP experiments. **EEG base** advances electrophysiology research by enabling access to public data, tools and results of research groups.

Features:

- Management of EEG/ERP data and metadata
- Management of EEG/ERP experimental design (experimental scenarios)
- Management of data related to tested subjects
- Sharing of knowledge and working within groups
- Signal processing tools
- Content management system
- Fulltext search

For continuing on this website you need to log in. If you don't have an account, you can create one.

[7] Name and surname are collected during the registration process, as it was before the change.

9.5. Removing Plain Text Passwords

Users can be registered by several methods:

- By pairing an account with Facebook
- By registering manually
- Being added by an experiment administrator

Except in the Facebook case, a verification message is sent to the user mail. Originally, it contained both username and password to log in. While adding a new user to the experiment generates an user with minimal rights and a random password, manually registering with the web interface sent the password just provided in the registration form, raising security issues described in section 8.1.

Password was therefore removed from the message generated while registering through the web interface.

10. Fixing Authorization and Mitigating Risks

Authorization was updated to reflect current needs for access control and steps were taken to fix problems found in chapter 8. Following sections will detail the measures and, in the case of global page access configuration, also recommend further improvements.

10.1. Page Access

Access to personal details was restricted to the user himself and the administrators. Using interceptions in the security context, permissions were updated in the following areas:

1. Viewing history
2. Viewing other users
3. Group membership editation
4. Adding people
5. Article settings

None of the listed actions is now permitted for Reader and history pages require Group Admin or administrative rights.

10.1.1. Proposed Design Improvement

The existing system uses two different mechanisms for handling user roles:

1. User authorities loaded on login by Spring Security from *user_role* database column. Those are Reader, User and Supervisor (Admin). The roles can be used in the security context or JSP views, using all features of the framework.
2. Roles acquired by recieving permission in other database tables[1]: Experimenter and Group Admin. These roles are not loaded by Spring Security and have to be manually checked when accessing a resource in the Controllers.

Because the second type does not integrate to the model offered by Spring Security, the roles can't be specified in the application context. While custom code resolving permissions on page loads does the necessary checks, it is nontransparent and verbose compared to the one-lined central definition in security context.

I propose to do the following:

1. Create an UserDetailsService implementation to intercept the authentication process and wire it to default Authentication Provider
2. In the service, create UserDetails object with roles loaded on login. But instead of using just one column, do all necesarry queries to resolve all roles and set them to the object[2].
3. All access permissions including the 'dynamic' roles can be now defined in the security context and verified using build-in Spring Framework support. Remove custom role checking code.

Advantages to such approach would include easier maintenance and being less prone to errors leading to unauthorized access. An important disadvantage is the need to reauthenticate when an user's role is changed (for example when one becomes Group Admin). That can either be done manually by logging out and in again, or by writting additional code to create new Authentication object on such actions.

10.2. Conditional Rendering

Authority based conditional rendering of page links was already in place in most protected pages. Tags from the Spring Security JSP library (introduced in subsection 4.2.2)

[1]For example being specified as owner in a Group entity grants an user the Group Admin role

[2]Although it would mean running multiple queries for each user, it is done only once on login. The current implementation needs to run the queries everytime a resource checking Experimenter or Group Admin roles is accessed.

with a defined list of accepted roles are used to hide inaccessible content and actions. This functionality was extended to strengthen access restrictions in areas listed in section 10.1.

If Experimenter and Group Admin roles were fully integrated to Spring Security as proposed in subsection 10.1.1, it would be possible to exclusively use authorization tags checking URL access. Elegance of such solution lies in the fact, that roles would no longer need to be listed in JSP pages and conditional rendering would always stay synchronized with page access defined in the security context[3].

10.3. Protecting URL Parameter Input

Pages listed in Table 10.1 were found to fail when submitting invalid URL parameter input. Parameter verification was added to each of those.

Table 10.1.: Pages with unchecked parameter input

Page	Unchecked parameter
articles/detail.html	articleId
articles/edit.html	articleId
articles/delete.html	articleId
articles/subscribeGroupArticles.html	groupId
scenarios/detail.html	scenarioId
scenarios/edit.html	id
scenarios/download-xml.html	scenarioId
experiments/choose-metadata.html	id
experiments/detail.html	experimentId
experiments/add-optional-parameter.html	experimentId
experiments/edit.html	id
people/edit.html	id

10.4. Specifying Default Error Behavior

A separate page informing about a generic server error (HTTP Error 500) was created alongside existing HTTP Error 404 page and set up to show on uncaught exceptions (Listing 10.1) in the web descriptor.

[3]Because the access rules will be inherited from the security context, no changes would need to be made to the definitions in JSPs even if the permission or role model radically changes.

Listing 10.1 Specifying server error page in web.xml

```
<error-page>
        <error-code>500</error-code>
        <location>/error500.jsp</location>
</error-page>
```

10.5. Transport Layer Security

With the help of Ing. Matejka, a certificate trusted by global authority was obtained and production server set up to use TLS. All communication between a client and the server is now encrypted, using the HTTPS protocol (Hypertext Transfer Protocol Secure).

10.6. HTTPOnly Cookies

Java Servlet 3 specification introduces an option to set up HttpOnly cookies globally (for the whole application).

A corresponding JAR file was added to the application dependency list during Maven migration, so only two changes were needed in the web descriptor:

1. Updating XML namespaces to 3.0 specification
2. Adding a snippet from Listing 10.2

A vulnerability scanner (NetSparker, detailed in section 12.1)[4] was used to verify, that HttpOnly cookies are now used exclusively.

Listing 10.2 Setting HttpOnly in web.xml

```
<session-config>
        <cookie-config>
                <http-only>true</http-only>
        </cookie-config>
<session-config>
```

[4]A preliminary scan identified absence of HttpOnly limitation as a vulnerability

11. Testing

Every project code update described in this thesis was tested before committing changes to the versioning system. But to ensure long term maintainability, the tests should be automated. That way the testing is reproducible and can be used to verify algorithm functionality, when the implementation is changed.

This chapter first describes test types, then actions done to fix existing test support and finally algorithms implemented new by tests to verify authentication and authorization setup.

11.1. Test Types

According to [23], main software testing types divided by target are:

- Unit testing: Verifies functionality of software pieces in isolation. Typically that means testing individual class methods.
- Integration testing: Tests interaction between application parts and behavior of assembled components.
- System testing: Observes behavior of the whole system. Interoperability with other applications, devices and systems is also concerned.

Tests can be further classified by objectives, techniques etc. Types relevant for this chapter are, among the previously listed:

- Performance testing: Verifies, that the application performance matches specified goals (for example being able to serve a certain amount of users).
- Smoke testing: Tests only basic, critical functionality to check, that the code is free of serious defects or misconfiguration.

Unit, integration and performance tests were introduced during development of the EEG/ERP Portal. However, no type is offering full code coverage and only integration tests were still working in late 2011. Performance tests are not very important for testing security, but the other two types can be utilized to verify project setup and the new functionality, as will be described in the following sections.

11.2. Unit Tests

The main goal of fixing unit tests was to verify project setup after Maven migration, enabling future testing and making it possible to use Spring context in integration tests. But a few new tests were also be added to aid in debugging and to verify created code.

In Maven, all tests are normally executed before creating a package (WAR). This behavior was disabled, not to block artifact assembly until all tests are working and succeed.

11.2.1. Fixing Existing Tests

Before new tests could be added, existing configuration was fixed, including test classes where possible. That was done in the following steps:

1. Moving classes and resources to Maven-defined structure, as described in section 7.1.
2. Updating context configuration to match project context and persistence setup[1]
3. Test method update to match current business and domain class implementations

In the case of tests requiring Spring context initialization, this was blocked by an interoperability issue. One of Hibernate DTD files could not be read due to XML parsing errors (for reference, the error is printed in Listing 11.1).

The EEG/ERP project uses Oracle database XML support to store structured data (experiment files in XML format). This requires a library called Oracle XDB parser as a dependency. Upon linking to the project, Oracle XDB sets itself as a default

[1]Tests use the same development database, as the dev project channel

Listing 11.1 Initializing test context with Oracle XDB parser

```
ERROR ErrorLogger - Error parsing XML (31) : http://hibernate.
    sourceforge.net/hibernate-mapping-3.0.dtd<Line 31, Column 2>:
    XML-20068: (Fatal Error) content model is not deterministic
...
Caused by: org.hibernate.InvalidMappingException: Unable to read
    XML
...
Caused by: org.dom4j.DocumentException: Error on line 31 of
    document http://hibernate.sourceforge.net/hibernate-mapping
    -3.0.dtd : http://hibernate.sourceforge.net/hibernate-mapping
    -3.0.dtd<Line 31, Column 2>: XML-20068: (Fatal Error) content
    model is not deterministic
```

XML parser. Unlike the build-in parser (Xerces/Xalan), Oracle XDB will fail while trying to parse Hibernate internal XML files.

3 possible solutions were considered:

1. Removing Oracle from test configuration
2. Finding a suitable Hibernate XML configuration conforming to the Oracle XDB accepted format
3. Explicitly switching default parser to Xerces

The first option is not practical, as XML types are an integral part of application data model. Problem with the second option lies in the fact, that the conflicting file was part of the Hibernate framework sources and could not be redefined by the framework API (Application Interface)[2] and an official suitable version is not available.

It is possible to set default XML parser before the context is initialized by setting system properties. Those can be defined:

- By setting environment variables in the operating system
- By command line parameter when starting the program[3]

[2]It would be technically possible to change the files directly inside the framework libraries. But that would introduce an incompatible fork of the framework, which could not be loaded from the central Maven repository and can never by updated to a newer version.

[3]Maven also has support for passing system properties to the test running plug-in, and this would actually solve the problem when using Maven. But it would drop unit test support for any other execution environments, like IDEs.

- Directly in Java code

As the source codes needed to be portable and tests can be executed by different applications, defining the environment variables in Java code remains the only option. A directive in Listing 11.2 was placed to superclass constructors to execute before the application context is initialized. Using this setup, the tests can be executed in any environment.

Listing 11.2 Redefining default XML parser in Java code

```
System.setProperty("javax.xml.parsers.SAXParserFactory",
        "org.apache.xerces.jaxp.SAXParserFactoryImpl");
System.setProperty("javax.xml.parsers.DocumentBuilderFactory",
        "org.apache.xerces.jaxp.DocumentBuilderFactoryImpl");
```

11.2.2. New JUnit Tests

As unit tests can be now easily constructed, several were created to check altered methods in the business and data tier.

- ConfigurationSetupTest using test context initialization to check, that the application is free of serious misconfiguration (essentially a smoke test).
- PersonDaoTest verifies user account creation and handling, including password and other properties.
- MailServiceTest can be used to evaluate email server configuration and test message sending.

11.3. Integration Tests

Integration tests were created to enable automatic access control setup verification. These tests require a fully assembled application and simulate an user interacting with the system.

11.3.1. Mechanism

Selenium Framework was used for the integration tests, as it is already included in the project[4]. This framework runs a browser instance and executes predefined commands via JavaScript.

Testing procedure requires logging in the application with different roles. One solution would be to manually create users designated for testing and hardwire the credentials in source code. That would, however, create two issues:

1. Anyone getting access to the source code would be able to read full login information of several users, including one with administrator privileges
2. Changing database table rows containing the user data or migrating to a new database would break the tests

A different solution was designed using the Spring testing support. Application context is initialized before loading Selenium, allowing any business code methods to be executed before and after running the framework. The following sequence is implemented:

1. A new user with defined role is created and saved to the database
2. His credentials are used to log in to the application front-end and perform tests
3. User is deleted when tests are done

11.3.2. Testing Authentication Setup

Correct setup of the sign-in process is verified by a few tests in the following scenarios:

- User with invalid credentials is rejected and presented a “Bad credentials” message.
- Valid credentials are accepted and the user is logged in.
- When requesting a protected resource, an unauthenticated user is redirected to the homepage with login form.
- If the user originally tried to access a resource different then the homepage, it should be displayed after signing in.

[4]Selenium is used for testing room reservation functionality [16]

11.3.3. User Authorization Tests

Page authorization is tested by automatically creating users and visiting protected resources.Table 11.1 shows checking access control rules, which do not rely on database setup[5].

Table 11.1.: Access to page as verified by integration test

Resource	Admin	User	Reader
Articles, Groups, My account	✓	✓	✓
Article settings	✓	✓	X
Room booking	✓	✓	X
Request for group role	✓	✓	X
List of people (users)	✓	✓	X
History	✓	X	X
Change role of user	✓	X	X

[5]Database checks outside user entity are needed for access control in groups. For example, group administrator can see user details of group members. Such rules have to be tested manually - no test class was created for this cases.

12. Evaluating Application Security

Vulnerability scanners were used to independently verify application security. Because the theoretical likehood of being compromised by other common attacks is already discussed in [1], it is not repeated here. Instead, improvements proposed in the thesis are compared to the actually implemented changes, marking security improvements in the covered areas.

12.1. Web Application Security Scanners

Portal security was tested by two publicly available[1] web application security scanners:

- NetSparker, an all-purpose scanner finding and exploiting multiple types of security flaws[2]
- Acunetix Free Edition, a tool specialized on finding Cross-site scripting (XSS) vulnerabilities

Deployment environment under test was the application running on localhost in Apache Tomcat 6 and Jetty 7 servlet containers[3] in two consecutive tests, using a remote testing database. Both server instances were started in default configuration and with no transport layer security (not using HTTPS).

[1]A free edition of Acunetix was used. While NetSparker also has a free community edition, it cannot be configured to login using form authentication, skipping a large part of the application. For this reason, a time limited evaluation version of the professional edition was employed instead.

[2]All types of vulnerabilities tested can be found on the company website: http://www.mavitunasecurity.com/netsparker/vulnerabilities/

[3]Jetty is used on the production server

Scanning with Acunetix did not find any flaws. Results from NetSparker analysis are listed in Table 12.1.

Table 12.1.: Test results by scanning local EEG/ERP Portal installation by NetSparker

Issue	Commentary
Probable SQL Injection in pages allowing user input (marked in Tomcat execution only)	By leveraging Spring support for Hibernate, the application uses parametrized queries (prepared statements) for all its SQL operations. This type of attack is thus not feasible.
Password transmitted over HTTP	Matches test setup. HTTPS is enforced on production server.
Password transmitted over query String	While being a false alarm for the password, session ID is indeed transmitted over query String under specific conditions (accessing the webpage with disabled cookies). Further research is needed to determine, if that can and should be addressed.
TRACE / TRACK Identified (Jetty execution only)	HTTP Trace can be used to bypass HTTP Only Cookie limitation when performing XSS attacks [21]. Should not be allowed on production server.
Auto complete enabled	Auto complete is not disabled on password field in the registration and login form. Browser might cache the value.
Server version disclosure	Target web server was identified by HTTP headers. This is standard setup in Jetty and Tomcat.
Database error message	Database connection error caused by network issues was displayed by the response page.
E-mail Address Disclosure	Email addresses were found in JQuery files. No other emails are disclosed by the portal.

From all the listed findings, the first three were marked as having medium severity, or higher, by the scanner. However, those can not be expected to succeed in the production environment.

Apart from identifying potential issues, the penetration test provided useful insight into the possibilities of automated attacks. When not addressed, critical flaws can be exploited without any human intervention by tools using similar techniques like the scanner. Also, it is clear that proper production server configuration is required to assure security of the whole system.

12.2. Comparing with Previous Evaluation

A detailed security analysis was previously performed in [1]. Section 8.3 describes possible improvements in the project as of 2011. Some are now implemented, others were not followed for various reasons. Proposed improvements were the following:

- SSL: Transport layer security is now deployed, improving resistance against session hijacking, man-in-the-middle attacks, sniffing and others
- Changing password storage: Either using an algorithm stronger than MD5, or hash salting is proposed. Using only one of these in isolation is still potentially vulnerable for two reasons:

    1. MD5 with salt can be viewed as plain MD5 with length = password + salt. If the password length is short, it can still be cracked by dictionary attacks
    2. Plain SHA-256 will produce always the same output for identical passwords. With a dictionary of common passphrase hashes, an attacker could easily find them among actual hash values.

  A combined solution was therefore implemented and deployed (section 9.2). The Insecure Cryptographic Storage vulnerability was thus removed.
- Technology innovation: Update of proposed frameworks took place during the realization phase. Only Spring Social module was not imported. While it should be possible to simplify Facebook authentication using its API, the existing process works without any regressions on the current setup. An effort to restructure the Facebook login would not add any value to the application. Migration can be done when extended functionality offered only by Spring Social is needed. Tech upgrade helped to increase application security in two aspects:

    1. Vulnerabilities were found in older versions of the frameworks[4]. Upgrading to current framework versions thus removed the issues.

[4]Spring vulnerabilities are listed at http://www.springsource.com/security/springsource-all

2. Features introduced in the new technology versions helped to improve application security. Examples are the HttpOnly cookie and BCrypt password storage.

- CSFR protection: Steps described in this thesis do not improve resistance against CSFR (Cross-site request forgery). Proposed steps to mitigate the problem are both technically challenging and detrimental to user experience with the software. Improvements in this field therefore remain available for further discussion.

13. Conclusion

Before updating technologies used by EEG/ERP Portal, a new build system based on Maven was deployed to simplify dependency and project management. The change enabled support of multiple IDEs and migration of source codes to use new versions of frameworks, most notably Spring, Spring Security, Hibernate and the Java Servlet technology. Custom developed libraries and JARs with restrictive license were stored in an internal Maven repository, created on top of existing versioning system. Ant-based plugin for message conversion to JSON format, used by AJAX components, was ported to become part of the new build system.

Security analysis of the existing code revealed issues to be addressed, like disclosing error stacks, not checking URL inputs, lack of transport layer protection, shortcomings in authorization and insecure password storage. Steps were made to address the issues and to introduce other improvements for enhancing user experience.

Storing and verifying passwords now utilizes BCrypt in producing salted hashes of the passwords, making it impractically time consuming to reveal the original values even if an attacker gets access to the database. Login dialogs were unified to use a single page. Email, which is less likely to be forgotten than a custom username, is now accepted as the main identification in login form. Passwords are no longer sent in plaintext by email on user registration. URL parameters are checked before processing and a custom error page is displayed when an uncaught exception causes the view to fail. Page access was restricted where needed and configuration was set up to mitigate potential threats. Existing tests were fixed and new ones were introduced to verify the added code. Finally, resistance against common attack scenarios was tested using a vulnerability scanner and the results were analyzed. The current security design was also compared to a previously proposed list of improvements.

While multiple aspects of application security were improved, the effort to assure security should not cease, as the need for it is continuous.

List of Abbreviations

ACL	Access control list
AJAX	Asynchronous JavaScript and XML
AOP	Aspect Oriented Programming
API	Application Interface
CAS	Central Authentication Service
CSFR	Cross-site request forgery
DAO	Data Access Object
DTD	Document Type Definition
DTO	Data Transfer Object
EEG	Electroencephalography
EJB	Enterprise Java Beans
ERP	Event Related Potentials
HQL	Hibernate Query Language
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated development environment
IP	Internet Protocol
JAAS	Java Authentication and Authorization Service
JAR	Java Archive

JAX-WS	Java API for XML Web Services
JDBC	Java Database Connectivity
JMX	Java Management Extensions
JPA	Java Persistence API
JSON	JavaScript Object Notation
JSP	Java Server Pages
JSR	Java Specification Request
JSTL	JavaServer Pages Standard Tag Library
LDAP	Lightweight Directory Access Protocol
MVC	Model–View–Controller
NTLM	NT LAN Manager
ORM	Object-relational mapping
PHI	Protected health information
PII	Personally Identifiable Information
POJO	Plain Old Java Object
POM	Project Object Model
REST	Representational state transfer
SAML	Security Assertion Markup Language
SHA	Secure Hash Algorithm
SSL	Secure Sockets Layer
SVN	Apache Subversion
TLS	Transport Layer Security
XML	Extensible Markup Language
XSS	Cross-site scripting

Bibliography

[1] Jiří Vlašimský. Access privileges in eeg/erp portal. Master's thesis, University of West Bohemia, Faculty of Applied Sciences, 2011 [cit. 2012-04-02]. Text in Czech. URL: `https://portal.zcu.cz/stag?urlid=prohlizeni-prace-detail&praceIdno=41447`.

[2] EU Directive. 95/46/EC of the european parliament and of the council of 24 october 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data. *Official Journal of the European Communities*, 281:31–50, 1995 [cit. 2012-04-17]. URL: `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:EN:NOT`.

[3] The Office for Personal Data Protection. Consolidated version of the personal data protection act (act no. 101/2000 coll.), 2000 [cit. 2012-04-17]. URL: `http://www.uoou.cz/files/101_en.pdf`.

[4] E. McCallister, T. Grance, and K. Kent. Guide to protecting the confidentiality of personally identifiable information (PII). Technical report, National Institute of Standards and Technology (US), 2009 [cit. 2012-04-17]. URL: `http://csrc.nist.gov/publications/nistpubs/800-122/sp800-122.pdf`.

[5] P.P. Gunn, A.M. Fremont, M. Bottrell, L.R. Shugarman, J. Galegher, and T. Bikson. The health insurance portability and accountability act privacy rule: a practical guide for researchers. *Medical Care*, 42(4):321, 2004 [cit. 2012-05-01]. URL: `http://www.rand.org/pubs/reprints/2005/RAND_RP1161.pdf`.

[6] P. Ježek and R. Mouček. database of eeg/erp experiments. In *Third International Conference on Health Informatics. Valencia Spain*, 2010.

[7] Jan Štěbeták. Computational tools in eeg/erp portal. Master's thesis, University of West Bohemia, Faculty of Applied Sciences, 2011 [cit. 2012-04-02]. URL: `https://portal.zcu.cz/stag?urlid=prohlizeni-prace-detail&praceIdno=41416`.

[8] Michael Cross. *Developer's guide to web application security.* Syngress Pub, Rockland, MA, 2007.

[9] Peter Mularien. *Spring Security 3 secure your web applications against malicious intruders with this easy to follow practical guide.* Packt Open Source, Birmingham, U.K, 2010.

[10] Blake Ives, Kenneth R. Walsh, and Helmut Schneider. The domino effect of password reuse. *Commun. ACM*, 47(4):75–78, April 2004 [cit. 2012-04-05]. URL: `http://doi.acm.org/10.1145/975817.975820`, `doi:10.1145/975817.975820`.

[11] Craig Walls. *Spring in action.* Manning, Shelter Island, 2011.

[12] Rod Johnson, Juergen Hoeller, Keith Donald, et al. Spring framework reference documentation, 2010 [cit. 2012-05-01]. URL: `http://static.springsource.org/spring/docs/3.0.5.RELEASE/reference/`.

[13] Ben Alex and Luke Taylor. Spring security reference documentation, 2012 [cit. 2012-05-01]. 3.1.0. URL: `http://static.springsource.org/spring-security/site/docs/3.1.x/reference/springsecurity-single.html`.

[14] Christian Bauer. *Java persistence with Hibernate.* Manning, Greenwich, Conn, 2007.

[15] Steve Ebersole. In relation to... tag: Core release, 2010 [cit. 2012-05-01]. URL: `http://in.relation.to/tag/Core+Release`.

[16] Jan Kolena. Eeg/erp portal - reservation system for eeg/erp laboratory. Bachelor thesis, University of West Bohemia, Faculty of Applied Sciences, 2011 [cit. 2012-04-02]. URL: `https://portal.zcu.cz/stag?urlid=prohlizeni-prace-detail&praceIdno=43594`.

[17] Jason van Zyl. History of maven, 2005 [cit. 2012-05-05]. URL: `http://maven.apache.org/background/history-of-maven.html`.

[18] Sonatype Company. *Maven : the definitive guide.* Oreilly, Sebastopol, Calif, 2008.

[19] Niels Provos and David Mazieres. A future-adaptable password scheme. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '99, pages 32–32, Berkeley, CA, USA, 1999. USENIX Associa-

tion. [cit. 2012-04-02]. URL: http://static.usenix.org/event/usenix99/provos/provos.pdf.

[20] Mike Shema. *Seven deadliest web application attacks.* Syngress/Elsevier Science, Amsterdam Boston, 2010.

[21] J. Grossman. Cross site tracing (xst). *WhiteHat Security White Paper*, 2003 [cit. 2012-04-29]. URL: http://www.cgisecurity.com/whitehat-mirror/whitePaper_screen.pdf.

[22] S. Indesteege. *Analysis and Design of Cryptographic Hash Functions.* PhD thesis, Katholieke Universiteit Leuven, 2010 [cit. 2012-04-06]. URL: http://www.cosic.esat.kuleuven.be/publications/thesis-171.pdf.

[23] A. Abran and P. Bourque. *SWEBOK: Guide to the software engineering Body of Knowledge.* IEEE Computer Society, 2004 [cit. 2012-04-10]. URL: http://www.computer.org/portal/web/swebok/htmlformat.

CD Contents

CD included with this thesis contains the following directories and files:

- *bin*: Contains assembled WAR with the EEG/ERP Portal
- *src*: Application source codes
- *thesis-src:* Source files for this document in LyX and TeX format
- *thesis-jnovotny.pdf*: Thesis in PDF format

Running the Application

To run the application, put the WAR from *bin* directory to a Servlet Container (tested on Apache Tomcat 6 and Jetty 7) and start the server. The application is running on localhost. Internet connection is required.

Compiling From Source Codes

To assemble the program from source codes:

1. Install Maven 2 from http://maven.apache.org/index.html
2. Copy the *src* directory to any writable location
3. Execute the following command in the new *src* folder and wait:

   ```
   mvn package
   ```

When done, a WAR file ready for deployment will be located in *target* subdirectory.