HIPAA Compliant Patient Interactive Website

Customer: Dr. Yoon H. Kang, DMD, PhD

Development Team

Satyanandana S Vadapalli

Aadish Shah

Harsha Vardhan Reddy Puli

Manoj Chappidi

Mohammad Hadian

Pritesh patel

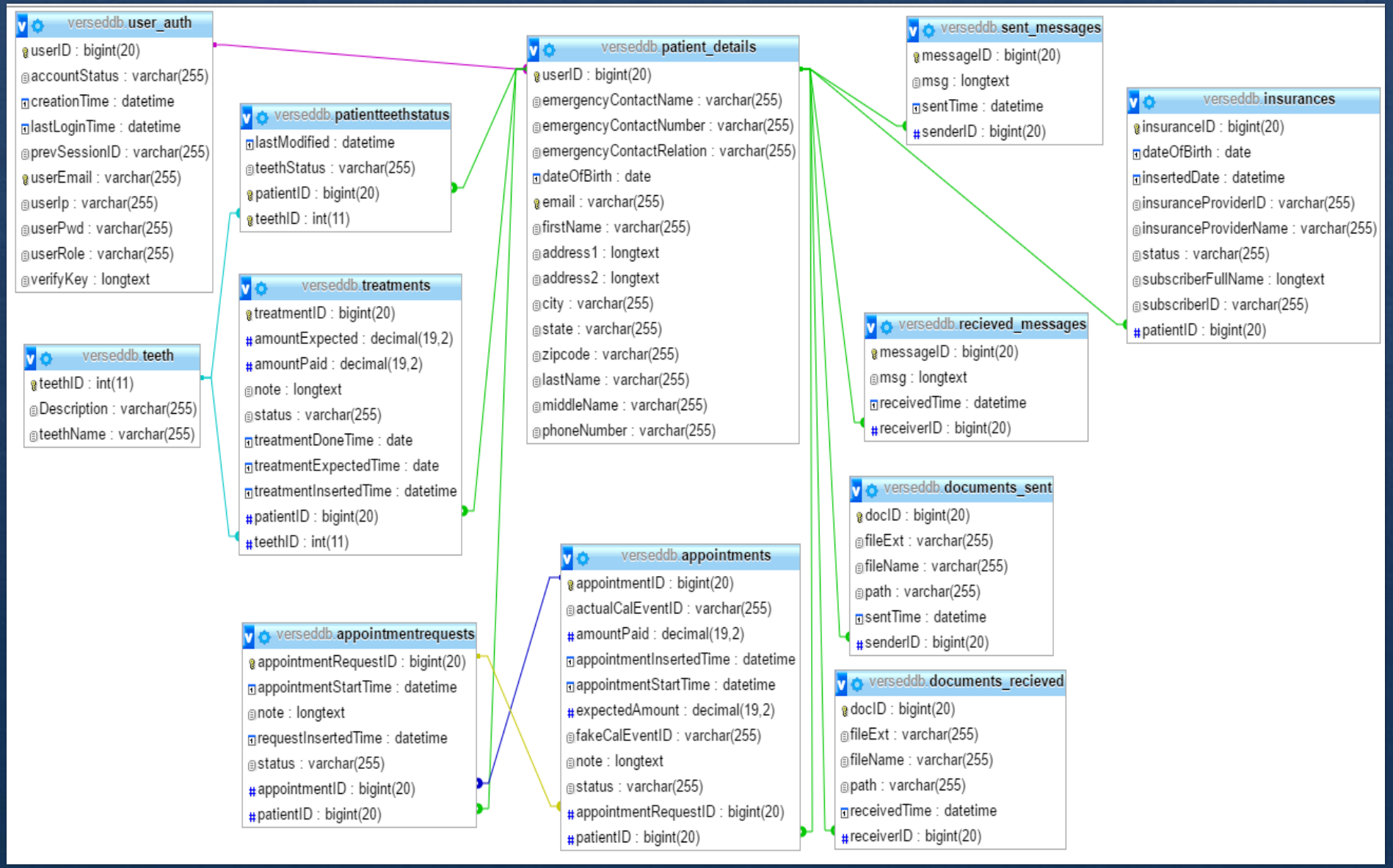
Customer Requirements

- Personalized Dental Health Management System
  - Interactive appointment Scheduling with the Google Calendar interface to display available slots.
  - Provide an option for the user to submit or update information like
    - Personal Information
    - Health Insurance Information
    - Contact information etc.
  - Provide an interface to send and receive messages securely between patient and doctor.
  - Provide an interface to exchange documents between patient and doctor.
  - Provide an interface to manage appointments like cancellation and changing the status.
  - Create some static pages like aboutus,FAQ,contactus,services etc.
  - Interactive Dental Treatment Status Page
    - History of Visits and Services records.
    - Dental Records.
  - Custom email notifications etc.
- MUST Comply with HIPAA Regulations
  - the “Health Insurance Portability and Accountability Act”
  - Any company that deals with protected health information must ensure that all the required physical, network, and process security measures are in place and followed.

Project Stack

- Framework : Spring 4
- ORM : Hibernate 4
- Other libraries : Log4j,Joda Time, Velocity Template Engine, Jasypt, Bouncy Castle, Calendar API Client, Geo Lite, Jackson etc.
- Frontend : HTML,CSS,Bootstrap,Jquery,Mustache Js
- IDE : Eclipse Mars
- Language : JAVA 7
- Server : Tomcat 7
- Version control : GitHub
- Build Tool : Maven
- Database : MySQL
- Hosting : Godaddy VPS

ER DIAGRAM


verseddb.user_auth
userID : bigint(20)
accountStatus : varchar(255)
creationTime : datetime
lastLoginTime : datetime
prevSessionID : varchar(255)
userEmail : varchar(255)
userIp : varchar(255)
userPwd : varchar(255)
userRole : varchar(255)
verifyKey : longtext
verseddb.patientteethstatus
lastModified : datetime
teethStatus : varchar(255)
patientID : bigint(20)
teethID : int(11)
verseddb.treatments
treatmentID : bigint(20)
amountExpected : decimal(19,2)
amountPaid : decimal(19,2)
note : longtext
status : varchar(255)
treatmentDoneTime : date
treatmentExpectedTime : date
treatmentInsertedTime : datetime
patientID : bigint(20)
teethID : int(11)
verseddb.teeth
teethID : int(11)
Description : varchar(255)
teethName : varchar(255)
verseddb.appointmentrequests
appointmentRequestID : bigint(20)
appointmentStartTime : datetime
note : longtext
requestInsertedTime : datetime
status : varchar(255)
appointmentID : bigint(20)
patientID : bigint(20)
verseddb.patient_details
userID : bigint(20)
emergencyContactName : varchar(255)
emergencyContactNumber : varchar(255)
emergencyContactRelation : varchar(255)
dateOfBirth : date
email : varchar(255)
firstName : varchar(255)
address1 : longtext
address2 : longtext
city : varchar(255)
state : varchar(255)
zipcode : varchar(255)
lastName : varchar(255)
middleName : varchar(255)
phoneNumber : varchar(255)
verseddb.appointments
appointmentID : bigint(20)
actualCalEventID : varchar(255)
amountPaid : decimal(19,2)
appointmentInsertedTime : datetime
appointmentStartTime : datetime
expectedAmount : decimal(19,2)
fakeCalEventID : varchar(255)
note : longtext
status : varchar(255)
appointmentRequestID : bigint(20)
patientID : bigint(20)
verseddb.sent_messages
messageID : bigint(20)
msg : longtext
sentTime : datetime
senderID : bigint(20)
verseddb.recieved_messages
messageID : bigint(20)
msg : longtext
receivedTime : datetime
receiverID : bigint(20)
verseddb.documents_sent
docID : bigint(20)
fileExt : varchar(255)
fileName : varchar(255)
path : varchar(255)
sentTime : datetime
senderID : bigint(20)
verseddb.documents_recieved
docID : bigint(20)
fileExt : varchar(255)
fileName : varchar(255)
path : varchar(255)
receivedTime : datetime
receiverID : bigint(20)
verseddb.insurances
insuranceID : bigint(20)
dateOfBirth : date
insertedDate : datetime
insuranceProviderID : varchar(255)
insuranceProviderName : varchar(255)
status : varchar(255)
subscriberFullName : longtext
subscriberID : varchar(255)
patientID : bigint(20)

IMPLEMENTATION

Spring Framework

We used the following modules of spring frame work.

- 1. Spring Context.
- 2. Spring Core.
- 3. Spring Beans.
- 3. Spring Web MVC
- 4. Spring Security
- 5. Spring Mail

- We used purely java based configuration with annotations to configure the spring application instead of using xml based configuration.
- Integrated the Spring Security feature with Spring web MVC.
- Integrated Hibernate 4 with Spring Framework.

- We used log4j for logging.
- We used Joda Time library instead of using java.util.Date
- Integrated Jasypt and Bouncy Castle with Spring Framework and Hibernate 4.
- Integrated Google API client. To insert/delete google calendar events through java code.
- Integrated Spring Mail instead of using javax.mail.
- Integrated Velocity Template Engine to send transactional emails.
- Integrated Geo Lite to get the location of client based on Ip address.
- Configured Jackson to serialize domain objects to JSON in Rest controllers.
- We used @Async and @EnableAsync to to send emails, create and delete google calendar events asynchronously.
- We used @Scheduled annotation to schedule batch tasks like sending remainder emails.

Spring Web MVC

- **org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer**
  - This is a base class for org.springframework.web.WebApplicationInitializer implementations that register a DispatcherServlet configured with annotated classes, e.g. Spring's @Configuration classes.
  - Concrete implementations are required to implement getRootConfigClasses() and getServletConfigClasses() as well as getServletMappings(). Further template and customization methods are provided by AbstractDispatcherServletInitializer.
- **org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter –**

  An implementation of WebMvcConfigurer with empty methods allowing sub-classes to override only the methods we're interested in.
  - Added static views to ViewControllerRegistry.
  - Configured InternalResourceViewResolver.
  - Configured ResourceHandlerRegistry for serving static resources such as images, css files and others through Spring MVC including setting cache headers optimized for efficient loading in a web browser.
  - Configured Message converters to serialize joda LocaleDate.class and DateTime.class with custom fromats while using Jackson in rest.
  - Configured Spring device detection.

- We used the following annotations.

  @Controller, @ResponseBody, @RestController, @EnableAsync, @Transactional, @RequestMapping, @PathVariable, @RequestParam, @CookieValue, @ModelAttribute etc.

Spring Security

- We implemented custom authentication mechanism by setting the SecurityContextHolder contents directly instead of using Spring Security defaults like Authentication Manager.
- Required beans for authentication and session management.

- **org.springframework.security.web.context.AbstractSecurityWebApplicationInitializer** –

Registers the DelegatingFilterProxy to use the springSecurityFilterChain before any other registered Filter. When used with AbstractSecurityWebApplicationInitializer(Class), it will also register a ContextLoaderListener. When used with AbstractSecurityWebApplicationInitializer(), this class is typically used in addition to a subclass of AbstractContextLoaderInitializer.

Note : we need to add SpringSecurityFilterChain to the DispatcherServlet.

- **org.springframework.security.core.session.SessionRegistryImpl –**

Bean required to maintains the registry of sessions. In order to make this bean work register HttpSessionEventPublisher listener with Spring DispatcherServlet.

- **org.springframework.security.web.authentication.SavedRequestAwareAuthenticationSuccessHandler –**

When a request is intercepted and requires authentication, the request data is stored to record the original destination before the authentication process commenced, and to allow the request to be reconstructed when a redirect to the same URL occurs. This bean is responsible for performing the redirect to the original URL if appropriate.

- **org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter –**

Provides a convenient base class for creating a WebSecurityConfigurer instance. The implementation allows customization by overriding methods.

SPRING SECURITY CONFIGURATION

```java
@Configuration
@EnableWebSecurity
@ComponentScan(basePackages = "com.dentist.webapp")
public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private SessionRegistry sessionRegistry;

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
                .antMatchers("/resources/**", "/signup/*", "/aboutme", "/askme", "/gallery", "/services", "/contactus", "/login/*", "/", "/home")
                .permitAll()
                .anyRequest().authenticated().and()
                .formLogin().loginPage("/login/form").permitAll().and()
                .logout()
                        .invalidateHttpSession(true)
                        .clearAuthentication(true)// instead of using .deleteCookies("JSESSIONID","USER")
                        .addLogoutHandler(new ProperCookieClearingLogoutHandler("JSESSIONID", "USER"))
                        .permitAll().and()
                .sessionManagement()
                                .maximumSessions(1)
                                .maxSessionsPreventsLogin(true).expiredUrl("/accessDenied")
                                .sessionRegistry(sessionRegistry).and()
                .and().headers()
                                .httpStrictTransportSecurity()
                                .includeSubDomains(true)
                                .maxAgeInSeconds(31536000).and()
                .and().csrf().ignoringAntMatchers("/doc/**");

    }
```

```
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Content-Type-Options: nosniff
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block
```

SINGLE SESSION PER USER

```
public static void handleSession(SessionRegistry sessionRegistry, AuthenticationSuccessHandler successHandler, HttpServletRequest request,
        HttpServletResponse response, UserAuthentication user, HibernatePBEStringEncryptor encryptor, Patient patient)
        throws IOException, ServletException {

    List<GrantedAuthority> authorities = AuthorityUtils.createAuthorityList(user.getUserRole().toString());
    UserDetails userDetails = new CustomUserDetails(user);
    Authentication auth = new UsernamePasswordAuthenticationToken(userDetails, user.getUserPwd(), authorities);

    Iterator<SessionInformation> i = sessionRegistry.getAllSessions(auth.getPrincipal(), true).iterator();
    while (i.hasNext()) {
        SessionInformation si = i.next();
        si.expireNow();
    }
    sessionRegistry.registerNewSession(request.getSession().getId(), auth.getPrincipal());
    LOGGER.debug("added user to the spring session registry");
    user.setPrevSessionID(request.getSession().getId());
    LOGGER.debug("updated new sessionID in the database");
    SecurityContextHolder.getContext().setAuthentication(auth);
    LOGGER.debug("added user to the spring security context holder");
    request.getSession().setAttribute("user", user.getUserEmail());
    request.getSession().setAttribute("role", user.getUserRole());
    request.getSession().setAttribute("name", patient.getFirstName() + " " + patient.getLastName());
    LOGGER.debug("added user to the http servlet session");
    Cookie cookieUserId = new Cookie("USER", encryptor.encrypt(user.getUserEmail() + "-" + request.getSession().getId()));
    cookieUserId.setMaxAge(24 * 60 * 60); // 24 hours.
    cookieUserId.setComment("www.kangdentalnewton.com");
    cookieUserId.setHttpOnly(true);
    cookieUserId.setPath(request.getContextPath() + "/");
    // uncomment the below lines during production
    // cookieUserId.setDomain("https://www.kangdentalnewton.com");
    // cookieUserId.setSecure(true);
    response.addCookie(cookieUserId);
    LOGGER.debug("added user to the cookie");
    successHandler.onAuthenticationSuccess(request, response, auth);
    LOGGER.debug("Redirecting to where the request came from i.e " + request.getPathTranslated());
}
```

Expression Based Access Control

- **@EnableGlobalMethodSecurity(prePostEnabled = true)**

Note : This annotation should be applied on web MVC configuration i.e org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter.And it requires an AuthenticationManager bean.

- **@PreAuthorize("hasRole('ROLE_ADMIN')")**

```
@Controller
@EnableAsync
@Transactional
@RequestMapping(value = "/admin")
public class AdminController {

    private static final Logger LOGGER = Logger.getLogger(AdminController.class);

    @PreAuthorize("hasRole('ROLE_ADMIN')")
    @RequestMapping(value = "/dashboard", method = RequestMethod.GET)
    public String getAdminDashbord(Model model) {
        LOGGER.debug("processing GET request to /admin/dashbord ....");
        return "admindashboard";
    }
```

SSL

We installed the SSL certificate on Godaddy server to make sure that all data passed between the web server and browsers remain private and integral.

On GoDaddy hosting tomcat 7 is not installed as a standalone application. Instead, it runs on apache server module. So the SSL certificates are installed on apache instead of tomcat. So we can't configure the tomcat 7 to run on secure port 8443 by editing the server.xml file. Instead, we should rewrite all the request URL to HTTPS using a .htaccess file with the following code.

```
SetHandler jakarta-servlet
SetEnv JK_WORKER_NAME ajp13

RewriteEngine On
RewriteCond %{HTTPS} off
RewriteRule ^(.*) https://%{HTTP_HOST}/$1 [L,R=301]
```

File Uploads

We are renaming the file on upload to ensure the correct file extension. We are preventing direct access to uploaded files altogether. This way, any files uploaded to our web site are stored in a folder outside of the web root. We changed the file permissions to 0666 so it can't be executed.

Hibernate 4

- **org.springframework.orm.hibernate4.LocalSessionFactoryBean**

This is the usual way to set up a shared Hibernate SessionFactory in a Spring application context; the SessionFactory can then be passed to Hibernate-based data access objects via dependency injection. This variant of LocalSessionFactoryBean requires Hibernate 4.0 or higher.it is closer to AnnotationSessionFactoryBean since its core purpose is to bootstrap a SessionFactory from package scanning. Configure this bean by setting DataSource,Hibernate and packages to scan.

```
@Bean
public LocalSessionFactoryBean sessionFactory() {
    LocalSessionFactoryBean sessionFactory = new LocalSessionFactoryBean();
    sessionFactory.setDataSource(dataSource());
    sessionFactory.setPackagesToScan(new String[]{"com.dentist.*"});
    sessionFactory.setHibernateProperties(hibernateProperties());
    return sessionFactory;
}
```

- **org.springframework.orm.hibernate4.HibernateTransactionManager.HibernateTransactionManager**

Configuration of Hibernate Transaction Manager. This is important to make Spring Transaction work i.e. @Transactional. Set LocalSessionFactory to transaction manager. Binds a Hibernate Session from the specified factory to the thread, potentially allowing for one thread-bound Session per factory. SessionFactory.getCurrentSession() is required for Hibernate access code that needs to support this transaction handling mechanism, with the SessionFactory being configured with SpringSessionContext.

```
@Bean
@Autowired
public HibernateTransactionManager transactionManager(SessionFactory s) {
    HibernateTransactionManager txManager = new HibernateTransactionManager();
    txManager.setSessionFactory(s);
    return txManager;
}
```

- We applied relational mappings between entities like One to One, One to Many and Many to Many. And working around shared primary key and composite keys with the help of @MapsId and @Embedded, @Embeddable annotations
- We used Hibernate Criteria API to perform CRUD operations on database.

Encryption

- **Encryption and Decryption of Hibernate Entity String properties using Jasypt.**

  We defined @TypeDefs in package-info.java in the package containing Hibernate entities.

```
@TypeDefs({@TypeDef(name = "encryptedString", typeClass = EncryptedStringType.class, parameters = {
        @Parameter(name = "encryptorRegisteredName", value = "HibernateStringEncryptor")})})
```

  and apply @Type(type = "encryptedString") annotation to String properties of Hibernate entities. So that the annotated properties will be automatically encrypted while writing to the database and decrypted while reading from the database.

- URL Safe Encryption
  - Since we are using Base64 as the output string format for jasypt encryptor. We need to make sure that encrypted string is properly encoded.

    we wrote a custom class to replace all '/' with '_' and all '+' with '-' before passing it as URL.

    And in order to decrypt an URL Safe encrypted string we need to replace all '_' with '/' and all '-' with '+'
  - Another alternative is to use Hexadecimal as the output string format for jasypt encryption.
- Encryptable Properties

```
1 #Hibernate properties
2
3 jdbc.driverClassName = com.mysql.jdbc.Driver
4 jdbc.url = jdbc:mysql://localhost:3306/verseddb
5 jdbc.username = versedfi
6 # for base64
7 jdbc.password = ENC(UYn4ArGxOtogtM0+1Hm0tu29PoDysC8d2yFzo8VdLu0=)
8
```

USER INTERFACE

- We used Bootstrap for styling and layout.
- We used JQuery for DOM manipulation.
- We used Ajax to fire most of the GET/POST requests to the server.
- We implemented module revealing pattern while writing JavaScript code.
- For dental icons we used FLATICON.css
- For font effects and icons we used font-awesome.css
- For multi language support we are using google translator.
- For templating we are using Mustache.js
- To provide filter/sort support on tables at admin side , we are using datatable.js

THANK YOU