shopizer-ecommerce / **shopizer** Public

Code · Issues 171 · Pull requests 209 · Discussions · Actions · Projects · ...

3.0.1

shopizer / sm-shop / src / main / java / com / salesmanager / shop / store / api / v1 / customer /

AuthenticateCustomerApi.java / Jump to

shopizer-ecommerce Customer api ✓ History

1 contributor

Executable File | 273 lines (209 sloc) | 11.9 KB

```java
package com.salesmanager.shop.store.api.v1.customer;

import java.util.stream.Collectors;
import java.util.stream.Stream;

import javax.inject.Inject;
import javax.servlet.http.HttpServletRequest;
import javax.validation.Valid;

import org.apache.commons.lang3.Validate;
import org.apache.http.auth.AuthenticationException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

import com.salesmanager.core.model.customer.Customer;
import com.salesmanager.core.model.merchant.MerchantStore;
import com.salesmanager.core.model.reference.language.Language;
import com.salesmanager.shop.constants.Constants;
import com.salesmanager.shop.model.customer.PersistableCustomer;
import com.salesmanager.shop.store.api.exception.GenericRuntimeException;
import com.salesmanager.shop.store.api.exception.ResourceNotFoundException;
import com.salesmanager.shop.store.api.exception.UnauthorizedException;
import com.salesmanager.shop.store.controller.customer.facade.CustomerFacade;
import com.salesmanager.shop.store.controller.store.facade.StoreFacade;
import com.salesmanager.shop.store.controller.user.facade.UserFacade;
import com.salesmanager.shop.store.security.AuthenticationRequest;
import com.salesmanager.shop.store.security.AuthenticationResponse;
import com.salesmanager.shop.store.security.JWTTokenUtil;
import com.salesmanager.shop.store.security.PasswordRequest;
import com.salesmanager.shop.store.security.user.JWTUser;
import com.salesmanager.shop.utils.AuthorizationUtils;

import io.swagger.annotations.Api;
import io.swagger.annotations.ApiImplicitParam;
import io.swagger.annotations.ApiImplicitParams;
import io.swagger.annotations.ApiOperation;
import io.swagger.annotations.SwaggerDefinition;
import io.swagger.annotations.Tag;
import springfox.documentation.annotations.ApiIgnore;

@RestController
@RequestMapping("/api/v1")
@Api(tags = {"Customer authentication resource (Customer Authentication Api)"})
@SwaggerDefinition(tags = {
    @Tag(name = "Customer authentication resource", description = "Authenticates customer, registe
})
public class AuthenticateCustomerApi {

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

    @Value("${authToken.header}")
    private String tokenHeader;

    @Inject
    private AuthenticationManager jwtCustomerAuthenticationManager;

    @Inject
    private JWTTokenUtil jwtTokenUtil;

    @Inject
    private UserDetailsService jwtCustomerDetailsService;

    @Inject
    private CustomerFacade customerFacade;

    @Inject
    private StoreFacade storeFacade;

    @Autowired
    AuthorizationUtils authorizationUtils;

    @Autowired
    private UserFacade userFacade;

    /**
     * Create new customer for a given MerchantStore, then authenticate that customer
     */
    @RequestMapping( value={"/customer/register"}, method=RequestMethod.POST, produces ={ "applica
    @ResponseStatus(HttpStatus.CREATED)
    @ApiOperation(httpMethod = "POST", value = "Registers a customer to the application", notes =
        @ApiImplicitParams({ @ApiImplicitParam(name = "store", dataType = "string", defaultValue =
                @ApiImplicitParam(name = "lang", dataType = "string", defaultValue = "en") })
    @ResponseBody
    public ResponseEntity<?> register(
                @Valid @RequestBody PersistableCustomer customer,
                        @ApiIgnore MerchantStore merchantStore,
                        @ApiIgnore Language language) throws Exception {



            customer.setUserName(customer.getEmailAddress());

                        if(customerFacade.checkIfUserExists(customer.getUserName(),  merchantStore
                                //409 Conflict
                                throw new GenericRuntimeException("409", "Customer with email [" +
                        }

            Validate.notNull(customer.getUserName(),"Username cannot be null");
            Validate.notNull(customer.getBilling(),"Requires customer Country code");
            Validate.notNull(customer.getBilling().getCountry(),"Requires customer Country code");

            customerFacade.registerCustomer(customer, merchantStore, language);

            // Perform the security
            Authentication authentication = null;
            try {

                authentication = jwtCustomerAuthenticationManager.authenticate(
                        new UsernamePasswordAuthenticationToken(
                                customer.getUserName(),
                                customer.getPassword()
                        )
                );

            } catch(Exception e) {
                return new ResponseEntity<>(HttpStatus.NOT_FOUND);
            }

            if(authentication == null) {
                return new ResponseEntity<>(HttpStatus.NOT_FOUND);
            }

            SecurityContextHolder.getContext().setAuthentication(authentication);

            // Reload password post-security so we can generate token
            final JWTUser userDetails = (JWTUser)jwtCustomerDetailsService.loadUserByUsername(cust
            final String token = jwtTokenUtil.generateToken(userDetails);

            // Return the token
            return ResponseEntity.ok(new AuthenticationResponse(customer.getId(),token));


    }

    /**
     * Authenticate a customer using username & password
     * @param authenticationRequest
     * @param device
     * @return
     * @throws AuthenticationException
     */
    @RequestMapping(value = "/customer/login", method = RequestMethod.POST, produces ={ "applicati
    @ApiOperation(httpMethod = "POST", value = "Authenticates a customer to the application", note
    @ResponseBody
    public ResponseEntity<?> authenticate(@RequestBody @Valid AuthenticationRequest authentication

        //TODO SET STORE in flow
        // Perform the security
        Authentication authentication = null;
        try {


                //to be used when username and password are set
                authentication = jwtCustomerAuthenticationManager.authenticate(
                        new UsernamePasswordAuthenticationToken(
                                authenticationRequest.getUsername(),
                                authenticationRequest.getPassword()
                        )
                );

        } catch(BadCredentialsException unn) {
                return new ResponseEntity<>("{\"message\":\"Bad credentials\"}",HttpStatus.UNAUTHO
        } catch(Exception e) {
            return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }

        if(authentication == null) {
            return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }

        SecurityContextHolder.getContext().setAuthentication(authentication);

        // Reload password post-security so we can generate token
        // todo create one for social
        final JWTUser userDetails = (JWTUser)jwtCustomerDetailsService.loadUserByUsername(authenti

        final String token = jwtTokenUtil.generateToken(userDetails);

        // Return the token
        return ResponseEntity.ok(new AuthenticationResponse(userDetails.getId(),token));
    }

    @RequestMapping(value = "/auth/customer/refresh", method = RequestMethod.GET, produces ={ "app
    public ResponseEntity<?> refreshToken(HttpServletRequest request) {
        String token = request.getHeader(tokenHeader);

        String username = jwtTokenUtil.getUsernameFromToken(token);
        JWTUser user = (JWTUser) jwtCustomerDetailsService.loadUserByUsername(username);

        if (jwtTokenUtil.canTokenBeRefreshed(token, user.getLastPasswordResetDate())) {
            String refreshedToken = jwtTokenUtil.refreshToken(token);
            return ResponseEntity.ok(new AuthenticationResponse(user.getId(),refreshedToken));
        } else {
            return ResponseEntity.badRequest().body(null);
        }
    }

    @RequestMapping(value = "/private/customer/password", method = RequestMethod.PUT, produces ={
    @ApiOperation(httpMethod = "PUT", value = "Change customer password", notes = "Change password
    public ResponseEntity<?> setPassword(
                @RequestBody @Valid AuthenticationRequest authenticationRequest,
                @ApiIgnore MerchantStore merchantStore,
                @ApiIgnore Language language) {


                        String authenticatedUser = userFacade.authenticatedUser();
                        if (authenticatedUser == null) {
                                throw new UnauthorizedException();
                        }

                        userFacade.authorizedGroup(authenticatedUser, Stream.of(Constants.GROUP_SU


            Customer customer = customerFacade.getCustomerByUserName(authenticationRequest.getUser

            if(customer == null){
                return ResponseEntity.notFound().build();
            }


            customerFacade.changePassword(customer, authenticationRequest.getPassword());
            return ResponseEntity.ok(Void.class);

    }


    @RequestMapping(value = "/auth/customer/password", method = RequestMethod.POST, produces ={ "a
    @ApiOperation(httpMethod = "POST", value = "Sends a request to reset password", notes = "Passw
    public ResponseEntity<?> changePassword(@RequestBody @Valid PasswordRequest passwordRequest, H


        try {

            MerchantStore merchantStore = storeFacade.getByCode(request);

            Customer customer = customerFacade.getCustomerByUserName(passwordRequest.getUsername()

            if(customer == null){
                return ResponseEntity.notFound().build();
            }

            //need to validate if password matches
            if(!customerFacade.passwordMatch(passwordRequest.getCurrent(), customer)) {
              throw new ResourceNotFoundException("Username or password does not match");
            }

            if(!passwordRequest.getPassword().equals(passwordRequest.getRepeatPassword())) {
              throw new ResourceNotFoundException("Both passwords do not match");
            }

            customerFacade.changePassword(customer, passwordRequest.getPassword());
            return ResponseEntity.ok(Void.class);

        } catch(Exception e) {
            return ResponseEntity.badRequest().body("Exception when reseting password "+e.getMessa
        }
    }
```

```
}
```