var gulp = require("gulp");

var sass = require("gulp-sass");

var CleanCSS = require("gulp-clean-css");

var concat = require("gulp-concat-css");

var fs = require("fs");

var json = JSON.parse(fs.readFileSync("./config/config.json"));

/\*var scss\_css = function(done) {

  gulp

    //.src(json.scss\_css.scss\_src)

    .src("src/styles/\*.scss")

    .pipe(sass())

    .pipe(gulp.dest("src/styles"));

    done();

};\*/

var minify\_css = function(done) {

  files = [json.scss\_css.css\_src];

  fs.exists(files, function(exists) {

    gulp

      .src(files)

      .pipe(CleanCSS())

      .pipe(gulp.dest(json.scss\_css.dest\_mini\_css));

    done();

  });

};

var concate\_css = function(done) {

  return gulp

    .src(json.scss\_css.con\_css\_src)

    .pipe(concat(json.scss\_css.con\_css\_file))

    .pipe(gulp.dest(json.scss\_css.con\_css\_dest));

};

const cssTasks = gulp.series(minify\_css, concate\_css);

exports.cssTasks = cssTasks;

var gulp = require("gulp");

var spritesmith = require("gulp.spritesmith");

var imagemin = require("gulp-imagemin");

var fs = require("fs");

var path = require('path');

var json\_img = JSON.parse(fs.readFileSync("./config/config.json"));

function minify\_images(done){

  for(var i=0;i<json\_img.image.length;i++)

  {

      for(var j=0;j<json\_img.image[i].src\_img.length;j++)

      {

        const path=json\_img.image[i].src\_dir + json\_img.image[i].src\_img[j];

        console.log(path);

        gulp.src(path)

        .pipe(imagemin())

        .pipe(gulp.dest(json\_img.image[i].temp\_min\_img))

      }

  }

  done();

}

function dist\_image(done){

  gulp.src(json\_img.image[0].temp\_min\_img + "/\*.\*")

  .pipe(gulp.dest(json\_img.image[0].dist\_img))

  done();

}

function image\_sprite(done){

  for(var i=0;i<json\_img.image.length;i++)

  {

    for(var j=0;j<json\_img.image[i].src\_img.length;j++)

    {

      const path=json\_img.image[i].src\_dir + json\_img.image[i].src\_img[j];

      var spriteData=gulp.src(path).pipe(

        spritesmith({

          imgName:json\_img.image[i].temp\_img,

          cssName:json\_img.image[i].temp\_scss,

          algorithm: 'diagonal'

        })

      );

      spriteData.img.pipe(gulp.dest(json\_img.image[i].temp\_img\_dir));

      spriteData.css.pipe(gulp.dest(json\_img.image[i].temp\_scss\_dir));

      done();

    }

  }

}

/\*function minify\_image(done) {

  files = [json\_img.images.src];

  fs.exists(files, function(exists) {

    gulp

    .src(files)

    .pipe(imagemin())

    .pipe(gulp.dest(json\_img.images.dest\_mini\_img));

    done();

  });

}

function image\_sprite(done) {

  var spriteData = gulp.src(json\_img.images.src).pipe(

    spritesmith({

      imgName: json\_img.images.output\_img,

      cssName: json\_img.images.output\_scss

    })

  );

  spriteData.img.pipe(gulp.dest(json\_img.images.dest\_img));

  spriteData.css.pipe(gulp.dest(json\_img.images.dest\_scss));

  done();

}

const imageTasks = gulp.series(minify\_image,image\_sprite);

exports.imageTasks = imageTasks;\*/

const imageTasks=gulp.series(minify\_images,image\_sprite,dist\_image);

exports.imageTasks=imageTasks;

//gulp.task("sprites", image\_sprite);

//gulp.task("optimize", minify\_image);

var gulp = require("gulp");

var jsonMinify = require("gulp-jsonminify");

var fs = require("fs");

var json = JSON.parse(fs.readFileSync("./config/config.json"));

function json\_minify(done) {

  files = [json.mini\_json.json\_src];

  fs.exists(files, function(exists) {

    gulp

    .src(files)

    .pipe(jsonMinify())

    .pipe(gulp.dest(json.mini\_json.json\_dest));

    done();

  });

}

const jsonTasks = gulp.series(json\_minify);

exports.jsonTasks = jsonTasks;

//gulp.task("minify\_json", json\_minify);

(function () {

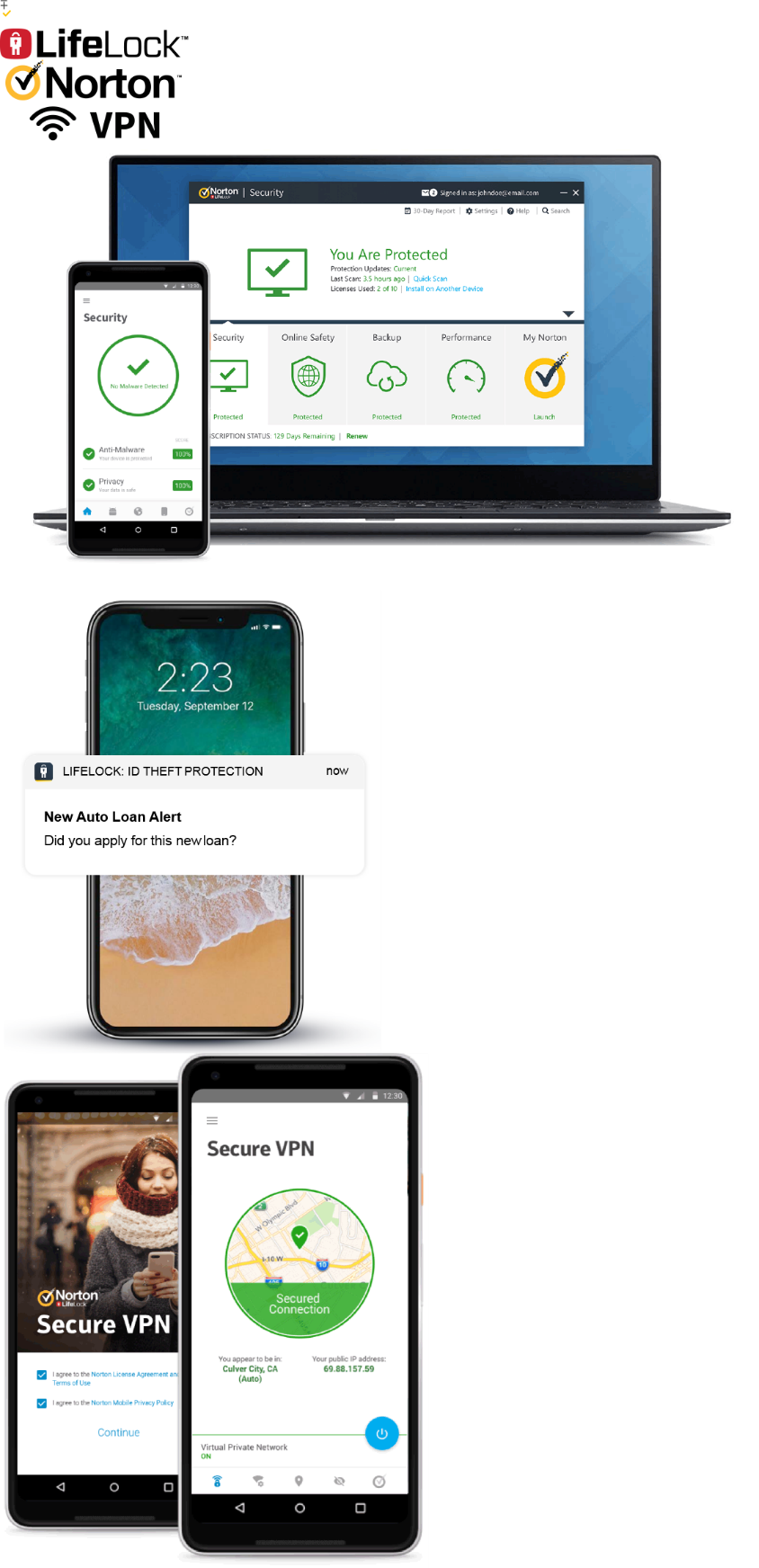
    $('head').append('<style><%= head %></style>');

    $('.background-egi').replaceWith('<%= body %>');

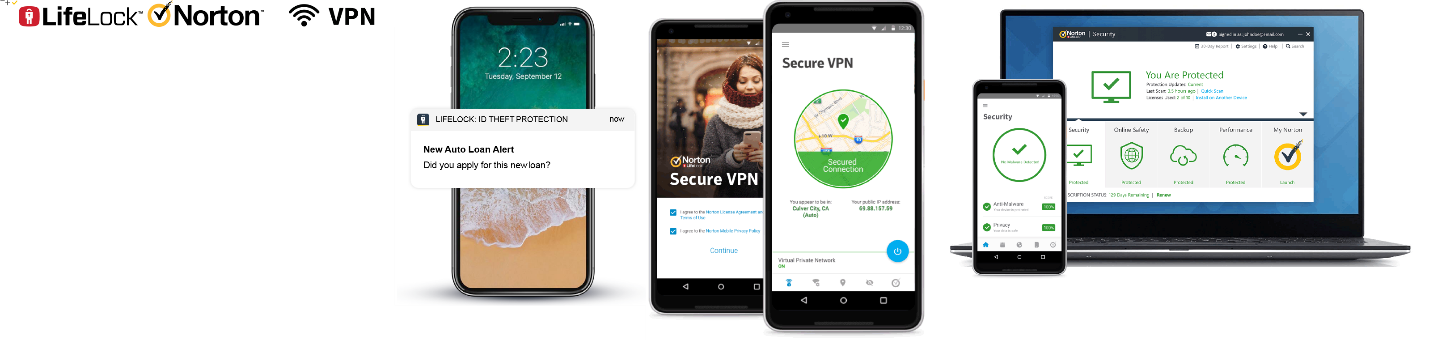
    <%= javascript %>

}).call(this);

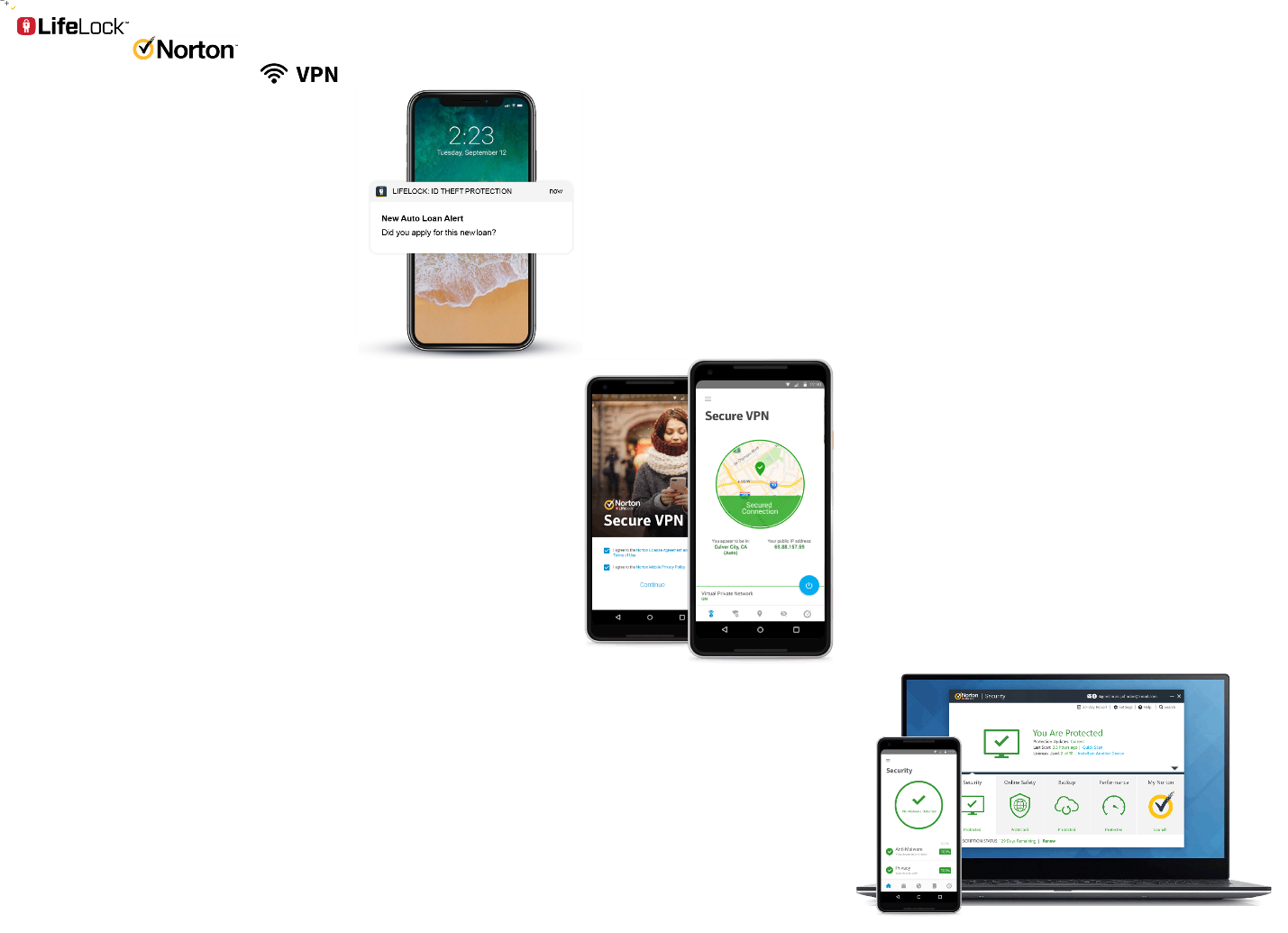
**Top-down**



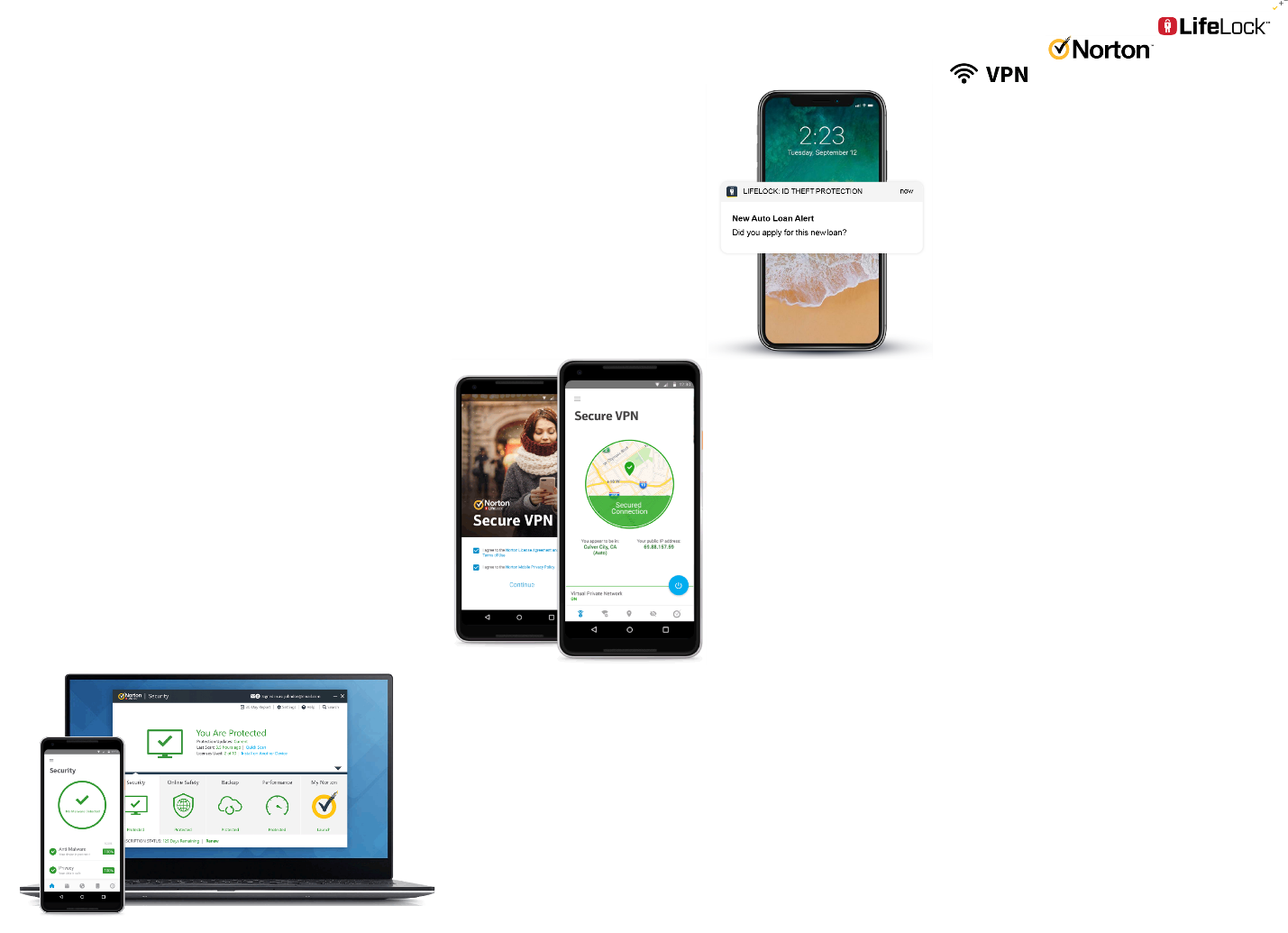
**Left-right**



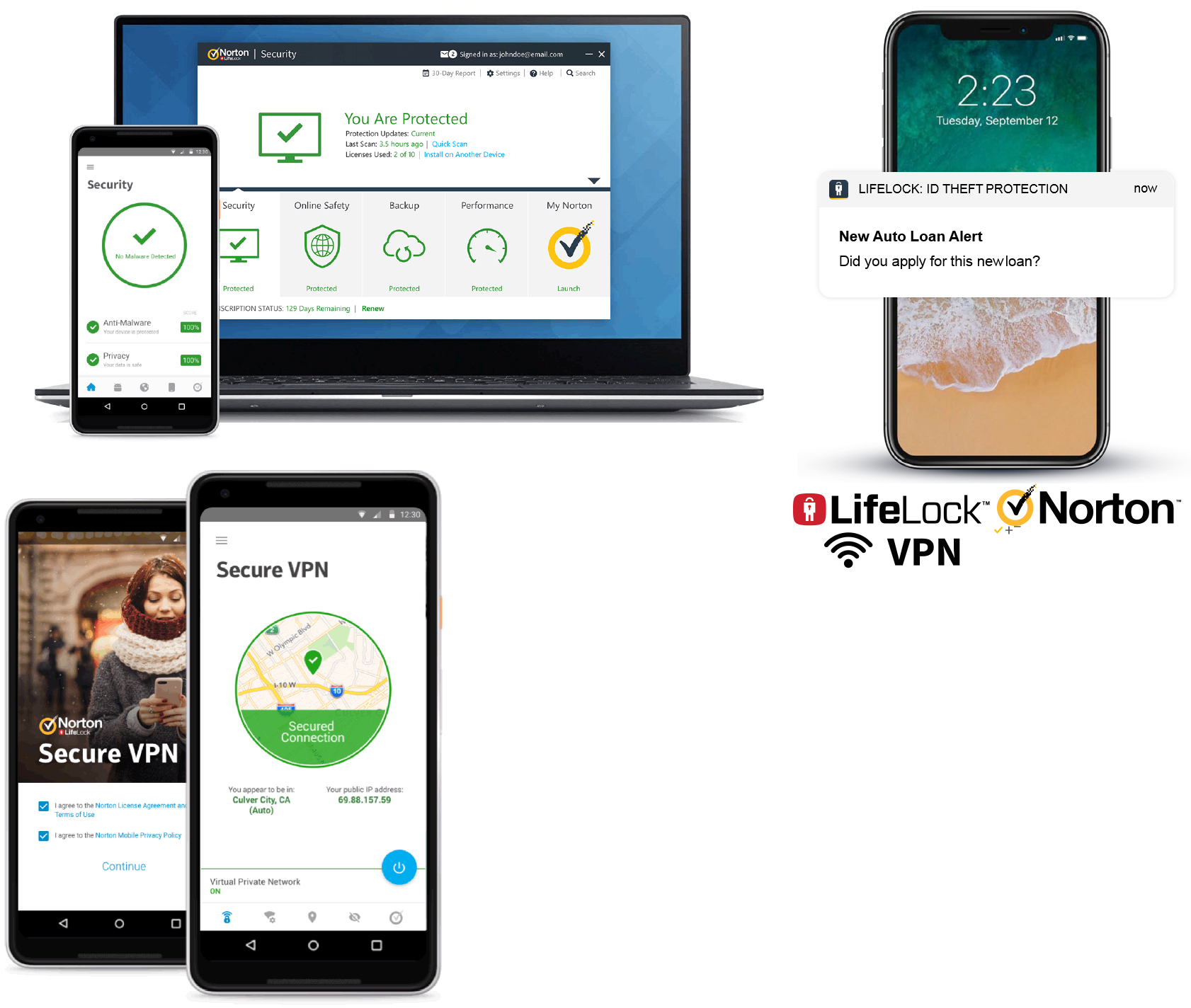
**diagonal**



**Alt-diagonal**



**binary-tree**



HTML CONCATE

var HtmlInputArray = [

  {

    path: "src/components/header/header.html",

    extractFrom: "#header",

    css\_path:"../src/components/styles/header.css"

  },

  {

    path: "src/components/body/body.html",

    extractFrom: "#image",

    css\_path:"../src/components/styles/body.css"

  },

  {

    path: "src/components/footer/footer.html",

    extractFrom: "#footer",

    css\_path:"../src/components/styles/footer.css"

  }

];

gulp.task("copy-html", function(done) {

  var HTMLContentList = [];

  //var HTMLSubContentList = [];

  for (var i = 0; i < HtmlInputArray.length; i++) {

    dom.JSDOM.fromFile(HtmlInputArray[i].path).then(

      function(config, htmlContentList, d) {

        htmlContentList.push(

          d.window.document.querySelector(config.extractFrom).innerHTML

        );

      }.bind(undefined, HtmlInputArray[i], HTMLContentList)

    );

  }

  dom.JSDOM.fromFile(output).then(

    function(htmlContentList, d) {

      d.window.document.querySelector("body").innerHTML += htmlContentList.join(

        "\n"

      );

      saveOutput(d.serialize(d.window.document.documentElement.innerHTML));

    }.bind(undefined, HTMLContentList)

  ).then(done);

});

function saveOutput(data) {

  fs.writeFile(output, data, function(error) {

    if (error) {

      throw error;

    }

    console.log("Copied portion to output successfully.");

  });

}

gulp.task("include-css-js", function(done) {

  return gulp

    .src("src/index.html")

    .pipe(

      cheerio(function($) {

        $("head").append(

          '<script src="../src/components/header/header.js"></script>'

        );

        $("head").append(

          '<link rel="stylesheet" href="../src/components/body/body.css">'

        );

        $("head").append(

          '<link rel="stylesheet" href="../src/components/header/header.css">'

        );

        $("head").append(

          '<link rel="stylesheet" href="../src/components/footer/footer.css">'

        );

      })

    )

    .pipe(gulp.dest("dest"))

});

Js file Creation in which append the html, css and javascript through gulp

function js\_createFn() {

  var body = fs.readFileSync(json.read\_file.read\_html).toString();

  var js = fs.readFileSync(json.read\_file.read\_js).toString();

  var css = fs.readFileSync(json.read\_file.read\_css).toString();

  return gulp

    .src(json.create\_js.js\_src)

    .pipe(template({ head: css, body: body, javascript: js }))

    .pipe(rename(json.create\_js.rename))

    .pipe(gulp.dest(json.create\_js.js\_dest));

}

function html\_contentFn() {

  return gulp

    .src(json.html\_content.html\_src)

    .pipe(querySelector(json.html\_content.querySelect))

    .pipe(replaceQuotes())

    .pipe(htmlmin({ collapseWhitespace: true }))

    .pipe(rename(json.html\_content.rename))

    .pipe(gulp.dest(json.html\_content.html\_dest));

}

function cssFn() {

  return gulp

    .src(json.scss\_css.append\_css\_src)

    .pipe(CleanCSS({ compatibility: "ie8" }))

    .pipe(rename(json.scss\_css.rename))

    .pipe(gulp.dest(json.scss\_css.css\_dest));

}

Run the task in gulp-

gulp.task(

   "default",

  gulp.series("final", "copy-html","include-css-js")

);

Koa js and Mongodb

1 Registration API

Plugins-

const koa = require("koa");

const app = new koa();

const koaRouter = require("koa-router");

const router = new koaRouter();

const serve = require("koa-static");

const bodyParser = require("koa-body");

require("dotenv").config();

const database = require("./database\_connect.js");

const bcrypt = require("bcryptjs");

const datalize = require("datalize");

const field = datalize.field;

const User = require("./models/user.js");

var formidable = require("koa2-formidable");

const Login = require("./models/login.js");

app.use(formidable({}));

app.use(router.routes()).use(router.allowedMethods());

Server start code-

app.listen(3000, () => {

  console.log("server start");

});

HTTP request- POST METHOD with validation

router.post(

  "/signup",

  datalize([

    field("email")

      .required()

      .email(),

    field("password")

      .required()

      .length(8, 15)

      .trim()

      .custom(function(value, result, ctx) {

        console.log(ctx.request.body.confirm\_password);

        console.log(value);

        if (value !== ctx.request.body.confirm\_password) {

          throw new Error("Passwords don't match");

        } else {

          return value;

        }

      }),

    field("confirm\_password")

      .required()

      .length(8, 15)

      .trim()

  ]),

  async ctx => {

    if (!ctx.form.isValid) {

      ctx.status = 400;

      ctx.body = {

        status: "error",

        message: "Form Validation Error",

        errors: ctx.form.errors

      };

      return;

    }

    ctx.body = {

      status: "success",

      message: "Correct data entered",

      data: ctx.form

    };

Password hashing

var hashedPassword = bcrypt.hashSync(ctx.request.body.password, 10);

    console.log(hashedPassword);

    console.log(ctx.request.body.email);

    const temp = new User();

    temp.email = ctx.request.body.email;

    temp.password = hashedPassword;

    await temp

      .save()

      .then(data => {

        ctx.body = data;

      })

      .catch(err => {

        ctx.body = "error: " + err;

      });

  }

STORE DATA IN DATABASE

const temp = new User();

    temp.email = ctx.request.body.email;

    temp.password = hashedPassword;

    await temp

      .save()

      .then(data => {

        ctx.body = data;

      })

      .catch(err => {

        ctx.body = "error: " + err;

      });

  }

Login API-

POST method - validation in form

router.post(

  "/login",

  datalize([

    field("email")

      .required()

      .email(),

    field("password")

      .required()

      .length(8, 15)

      .trim()

  ]),

  async ctx => {

    if (!ctx.form.isValid) {

      (ctx.status = 400),

        (ctx.body = {

          status: false,

          message: "Form Validation Error",

          errors: ctx.form.errors

        });

      return;

    }

    ctx.body = {

      status: "success",

      message: "Correct data entered",

      data: ctx.form

    };

Read the data from Database using MONGODB

await User.findOne({ email: ctx.request.body.email }, function(

      error,

      result

    ) {

      if (error) {

        return (ctx.body = {

          status: false,

          message: "User does not exist"

        });

      }

      if (result) {

        const isMatch = bcrypt.compareSync(

          ctx.request.body.password,

          result.password

        );

        if (isMatch) {

          ctx.body = {

            status: true,

            message: "User exist"

          };

          var hashedPassword = bcrypt.hashSync(ctx.request.body.password, 10);

          const temp = new Login();

          temp.email = ctx.request.body.email;

          temp.password = hashedPassword;

          temp

            .save()

            .then(data => {

              ctx.body = data;

            })

            .catch(err => {

              ctx.body = "error: " + err;

            });

        } else {

          return (ctx.body = {

            status: false,

            message: "Password does not match"

          });

        }

      } else {

        ctx.body = {

          status: false,

          message: "User does not exist"

        };

      }

    });

  }

);