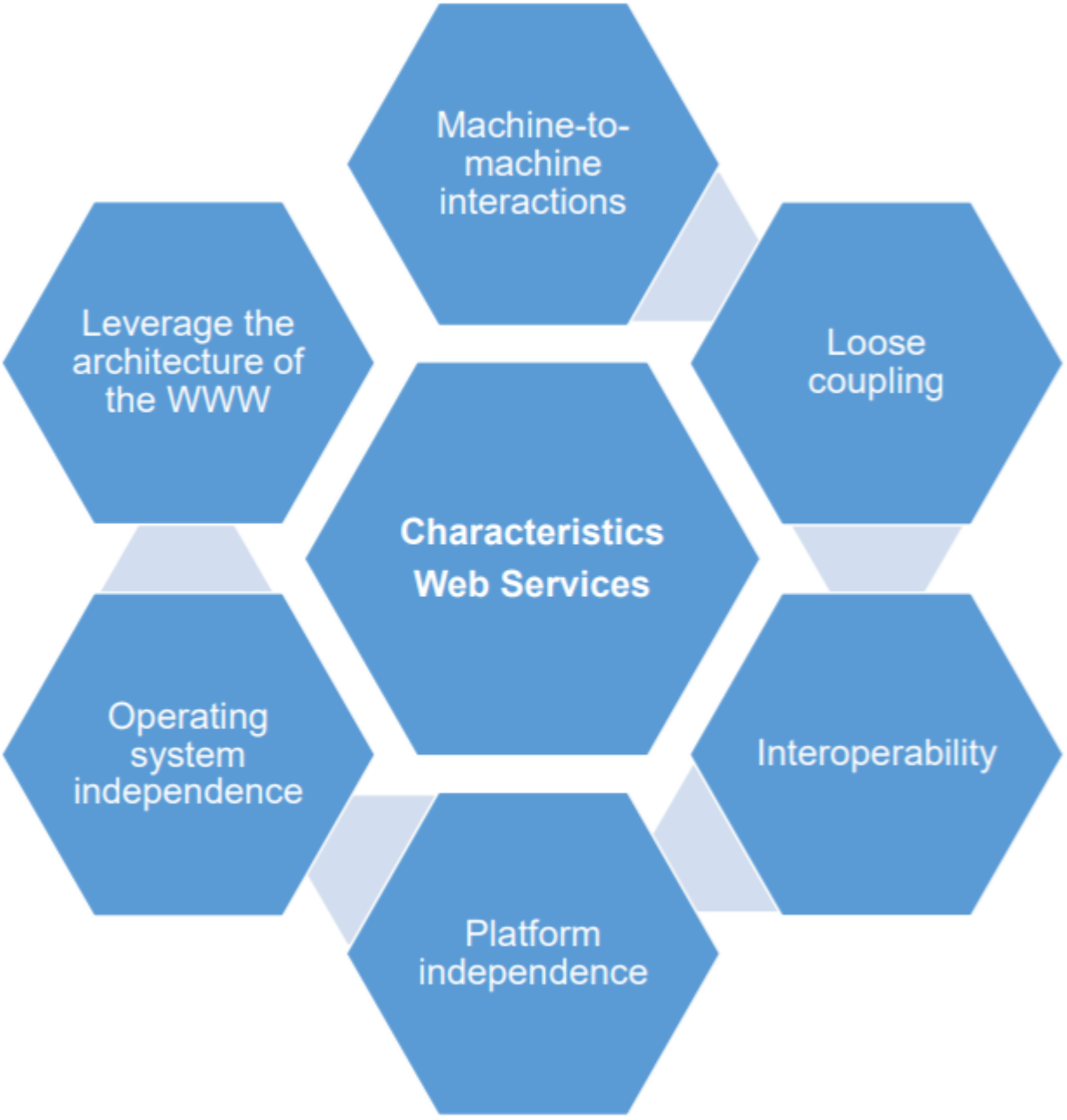
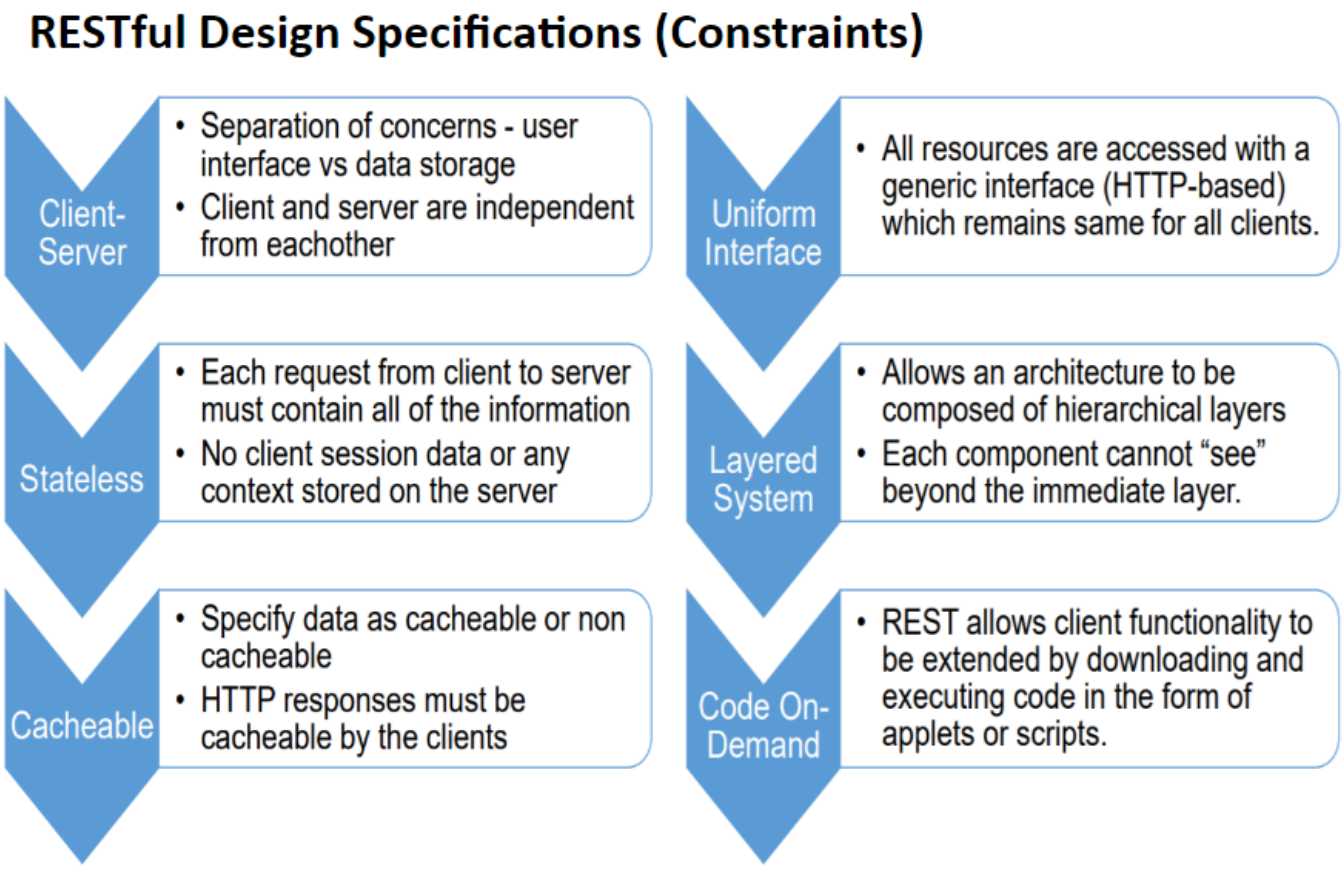


REST APIs

REpresentational **S**tate **T**ransfer

- Resource oriented architecture
- Client requests a specific resource from the server.
- The server responds to that request by delivering the requested resource.
- Server does not have any information about any client ->there is no difference between two requests from the same client.
  A model which the representations of the resources are transferred between the client and the server.

- Every resource -> uniquely addressable via URL



Design Considerations

1. Identify resources
2. Map URIs to resources
3. Apply uniform HTTP interface
4. Security considerations
5. XML/JSON representations for resources
6. Alternate resource representations
7. Provide resource metadata

Implementation

1. Parse:
    1. Use URI to identify the resource
    2. Identify URI variables and map them to resource variables
    3. Check HTTP method used in request and its validity for the resource in question
    4. Read the resource representation
2. Perform service logic
3. Return HTTP response
4.

Express

- Web application backend framework for Node

Routing

Request Objects

- req.query: Object with all query parameters
  order[status]=closed can be accessed as req.query.order.status).
- req.header(), req.get(header): Object with all headers stored as key value pairs
- req.path: Path for which the middleware func has been invoked
- req.url, req.originalURL: og proof against any modification done to regular
- req.body: key value pairs of data submitted in request body

Response Objects

- res.status(200).send(body)
- res.json({object})
- res.sendFile(path)

Middleware

A middleware is a function that takes in an HTTP request and response object, **AND THE NEXT MIDDLEWARE FUNCTION IN THE CHAIN.**

```
var router = express.Router()
router.use(function(req,res,next){
console.log("Router Middleware: "+req.body)
next()
})

// Application level middleware
 app.get("/student/:id",function(req,res,next)
{
    if(req.params.id.indexOf("2000")!=-1){
      var err = new Error("Something went Wrong!!!")
        next(err)
    }
    else{

        res.send("Everything is fine with"+req.params.id)
        next();
    }
})
app.use("/student",function(req,res){
        console.log("I am Fine")
})

// Error-handling middleware
app.use(function (err, req, res, next) {
    console.error(err.message); // Log the error message
    res.status(500).send("An error occurred: " + err.message); // Send a response to
the client
```

```
});

var server = app.listen(PORT, ()=>console.log("Server started on port: "+PORT))
```

Built in Middleware: express.static

```
const express = require('express');
const app = express();

// Serve static files from the 'public' folder
app.use(express.static('public'));

// Start the server
app.listen(3000, () => {
  console.log('Server running on port 3000');
});
```

- `express.static('public')`: This tells Express to look in the `public` folder whenever a request is made. If a file is requested, Express will serve it directly from the `public` folder.
- `app.listen(3000)`: Starts the Express server on port 3000. The server will listen for requests on this port.
  You can also serve static files under a specific URL prefix using the `express.static` middleware.

```
app.use('/static', express.static('public'));
```

Now, static files in the `public` directory can be accessed with the `/static` prefix, like this:

- `http://localhost:3000/static/images/photo.jpg`

Express Templates (Pug)

```
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');

const app = express();

// Set view engine to Pug
app.set('view engine', 'pug');
app.set('views', path.join(__dirname, 'views'));

// Middleware for parsing JSON and URL-encoded data
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

// Route to render the form
app.get('/', (req, res) => {
  res.render('form'); // Renders the `form.pug` template
});
```

```
// Route to handle form submission
app.post('/', (req, res) => {
  console.log(req.body); // Logs the submitted form data
  res.send('Received request: ' + JSON.stringify(req.body)); // Responds with the
received data
});

// Start the server
const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```

Pug file `form.pug`

```
doctype html
html
  head
    title Form Example
  body
    h1 Submit Your Details
    form(action="/" method="post")
      div
        label(for="name") Name:
        input(type="text" id="name" name="name" required)
      div
        label(for="email") Email:
        input(type="email" id="email" name="email" required)
      div
        button(type="submit") Submit
```

File Upload

```
var bodyparser = require('body-parser')
var fileUpload = require('express-fileupload')
app.set('view engine', 'pug')
app.set('views', './views')

app.get("/", (req,res,next)=>{
        res.render('form')
})

app.use(fileUpload()) // IMPORTANT: This should be before body-parser
app.use(bodyparser.json())
app.use(bodyparser.urlencoded({extended: true}))

app.post("/", (req, res)=>{
        console.log(req.body);
        var file = req.files.falala
        var filename = req.files.falala.name
```

```
        console.log(file)
        file.mv("./uploads/"+filename, (err)=>{
                if(err){
                        res.send('Received request: ' +
JSON.stringify(req.body)+"\n\nFile Failed to Upload");
                        throw err
                }
                res.send('Received request: ' + JSON.stringify(req.body)+"\n\nFile
Uploaded");
        }
        )
})

var server = app.listen(PORT, ()=>console.log("Server started on port: "+PORT))
```

form.pug

```
doctype html
html
head
  title PUG Form Example
body
  h1 Enter your details
  // IMPORTANT: DO NOT FORGET ENCTYPE
  form(method="post" enctype="multipart/form-data")
    div
    label(for="name") Name:
    input(type="text" id="name" name="name" required)
    br
    div
    label(for="falala") File:
    input(type="file" id="falala" name="falala" required)
    br
    button(type="submit") Submit!
```