Personal Website using Go, MongoDB, and Vue.js

NAMAN PARAKH

ADVISOR: PROF. JIM FAWCETT

Agenda

- Project Description and overview.
- Technologies to be used.
- Timeline for Task management.

Project Description

- Build a personal website comprising about providing comprehensive information about myself, my achievements and my views.

- Why Personal Website?
  - Easy way to reach out to you on internet.
  - Ability to express yourself in wide areas and the way you want.
  - Helps you in recruiting and shows you know Web Development ;)

Overview of Website

- The website will comprise of following sections:
  - **About me:** A brief introduction about myself along with my interests and link to download the resume.
  - **Personal projects:** A brief explanation about the projects that I have worked on and links to source code if available.
  - **Blogs:** My views on topics of my interests ranging from technical to rock music.
  - **Contact me:** Provide contact details along with a way to message me via email.

Technologies to be used

- Go

- Vue.js

- MongoDB

Go

- Developed by google and open source with great community
- C on steroids.
- Shorter code, readable code.
- No OOP :), procedural is fun over here.
- Threading is so easy with goroutines.
- Simplified concurrency using Channels.
- Web service development is breeze.

Go Continued...

```go
package main

import (
        "fmt"
        "net/http"
)

type Message struct{
        Msg string //global to any package
        local bool //local to the package
}


func local(){
        fmt.Println("I can only be called within this package as I don't start with CAPS")
}


func (msg *Message) AccessToMessage(){
        fmt.Println("I can only be called on Message pointer")
}


func Global(){
        fmt.Println("I can be called from any package as I start with CAPS. Access modifiers go to hell.")
}

func main(){
        http.HandleFunc("/", func (w http.ResponseWriter, r *http.Request){
                resp := Message{Msg:"Hello world"}
                fmt.Fprint(w, resp.Msg)
        })

        http.ListenAndServe(":8080", nil)
}
```

MongoDb

- Pros:
  - Open source and good community.
  - Schema less (NOSQL)
  - Flexible queries.
  - Scalable and managed services are available.
- Cons:
  - Heavy writes can be chaos.
  - Data redundancy due to denormalization

MongoDB continued..

- Data is organized into documents which are wrapped under collections.
- A document is represented as BSON which is binary representation of Json.
- A typical insert query looks like:

```
db.users.insertMany([
    {name : "Naman", last_name: "Parakh"},
    {name : "Pavan", last_name: "NS"}
])
```

- A typical find query looks like:

```
db.users.find({name : "Naman"})
```

Vue.js

- Open source and great community.
- A modern day Javascript framework for creating dynamic web applications.
- Component based along with the hooks into the component's lifecycle and data management.
- Provisions from routing within the application, state management to server-side rendering.
- Provisions of macros for binding a variable to element, iterating over the list, handling the if-else conditions, button clicks etc.

Vue.js continued...

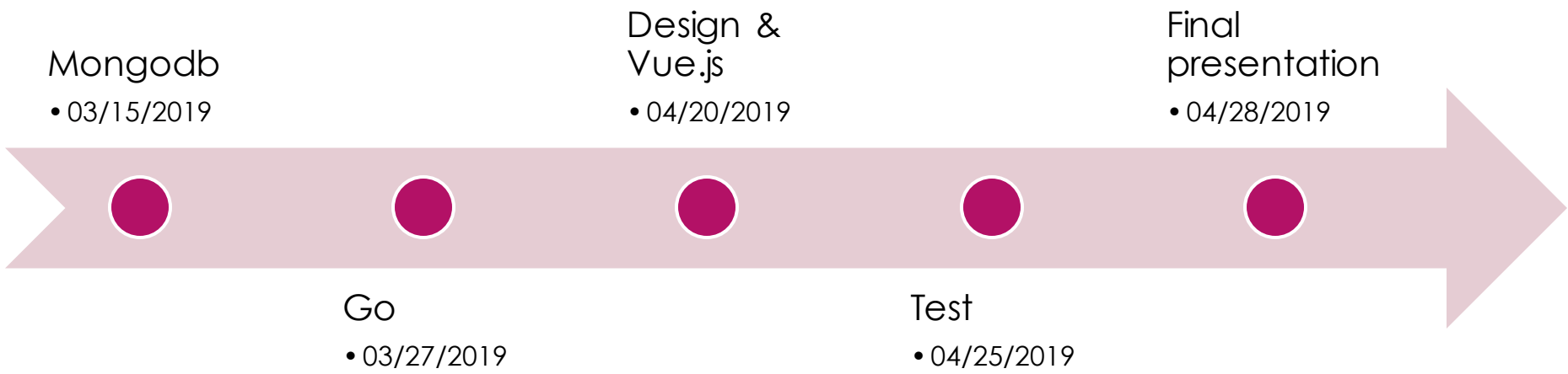
```vue
<template>
  <div class="hello">
    <h1> Message from parent: {{msgFromParent}}</h1>
    <br />
    <h2>{{ msg }}</h2>
    <h3>{{ reverseMessage }}</h3>
    <h3 v-if="shouldShow">I am visible because I am shown</h3>
    <u>
      <li v-for="hobby in hobbies">{{ hobby }}</li>
    </u>
    <button @click="changeMessage('Hello Naman!')">Change message</button>
  </div>
</template>

<script>
export default {
  name: 'HelloWorld',
  props:{
    msgFromParent: String
  },
  data: function(){
    return {
      msg: "Hello world!",
      shouldShow: false,
      hobbies:[
              "Touring",
              "Gaming",
              "Reading"
      ]
    }
  },
  methods: {
    changeMessage : function(newMsg){
      this.msg = newMsg
    }
  },
  computed: {
    reverseMessage: function(){
      return this.msg.split('').reverse().join('')
    }
  },
  watch : {
    reverseMessage: function(newMsg, oldMsg){
      return newMsg + oldMsg
    }
  }
}
</script>
```

Vue.js continued...

- State Management using Vuex
  - **Store:** A central place to serve the global state.
  - **Getters:** Like components but can be reused globally
  - **Mutations:** The only way to make change to the state of the store.
  - **Actions:** Instead of components mutating the state, actions commit mutations.
  - **Modules:** Sub module comprising of its own state, getters, mutations and actions, makes code much more readable and separate.

Timeline


Mongodb
• 03/15/2019
Go
• 03/27/2019
Design &
Vue.js
• 04/20/2019
Test
• 04/25/2019
Final
presentation
• 04/28/2019

Thank you!

NAMAN PARAKH